QTor : Une approche communautaire pour l'évaluation de requêtes

Sébastien Dufromentel-Fougerit

▶ To cite this version:

Sébastien Dufromentel-Fougerit. QTor : Une approche communautaire pour l'évaluation de requêtes. Réseaux et télécommunications [cs.NI]. Université de Lyon, 2016. Français. NNT : 2016LYSEI132 . tel-01715623

HAL Id: tel-01715623
https://theses.hal.science/tel-01715623

Submitted on 22 Feb 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N°d'ordre NNT : 2016LYSEI132

Thèse de Doctorat de l'Université de Lyon

opérée au sein du

Laboratoire d'informatique en image et systèmes d'information

Ecole Doctorale N° accréditation
Informatique et Mathématiques

Spécialité de doctorat : Informatique

Soutenue publiquement le 09/12/2016, par :

Sébastien Dufromentel

QTor : une approche communautaire pour l'évaluation de requêtes continues

Devant le jury composé de :

HAMEURLIN, AbdelKader Professeur des Universités Président

HAMEURLIN, AbdelKader	Professeur des Universités	Rapporteur
DEFUDE, Bruno	Professeur des Université	Rapporteur
SKAF, Hala	Maître de Conférence	Examinatrice
RONCANCIO, Claudia	Professeur des Universités	Examinatrice
LAMARRE, Philippe	Professeur des Université	Directeur de thèse
LESUEUR, François	Maître de Conférence	Co-directeur de thèse

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Département FEDORA – INSA Lyon - Ecoles Doctorales – Quinquennal 2016-2020

SIGLE	ECOLE DOCTORALE	NOM ET COORDONNEES DU RESPONSABLE
CHIMIE	**CHIMIE DE LYON** **http://www.edchimie-lyon.fr** Sec : Renée EL MELHEM Bat Blaise Pascal 3e etage secretariat@edchimie-lyon.fr Insa : R. GOURDON	M. Stéphane DANIELE Institut de Recherches sur la Catalyse et l'Environnement de Lyon IRCELYON-UMR 5256 Équipe CDFA 2 avenue Albert Einstein 69626 Villeurbanne cedex directeur@edchimie-lyon.fr
E.E.A.	**ELECTRONIQUE, ELECTROTECHNIQUE, AUTOMATIQUE** **http://edeea.ec-lyon.fr** Sec : M.C. HAVGOUDOUKIAN Ecole-Doctorale.eea@ec-lyon.fr	M. Gérard SCORLETTI Ecole Centrale de Lyon 36 avenue Guy de Collongue 69134 ECULLY Tél : 04.72.18 60.97 Fax : 04 78 43 37 17 Gerard.scorletti@ec-lyon.fr
E2M2	**EVOLUTION, ECOSYSTEME, MICROBIOLOGIE, MODELISATION** **http://e2m2.universite-lyon.fr** Sec : Safia AIT CHALAL Bat Darwin - UCB Lyon 1 04.72.43.28.91 Insa : H. CHARLES Safia.ait-chalal@univ-lyon1.fr	Mme Gudrun BORNETTE CNRS UMR 5023 LEHNA Université Claude Bernard Lyon 1 Bât Forel 43 bd du 11 novembre 1918 69622 VILLEURBANNE Cédex Tél : 06.07.53.89.13 e2m2@ univ-lyon1.fr
EDISS	**INTERDISCIPLINAIRE SCIENCES-SANTE** **http://www.ediss-lyon.fr** Sec : Safia AIT CHALAL Hôpital Louis Pradel - Bron 04 72 68 49 09 Insa : M. LAGARDE Safia.ait-chalal@univ-lyon1.fr	Mme Emmanuelle CANET-SOULAS INSERM U1060, CarMeN lab, Univ. Lyon 1 Bâtiment IMBL 11 avenue Jean Capelle INSA de Lyon 696621 Villeurbanne Tél : 04.72.68.49.09 Fax :04 72 68 49 16 Emmanuelle.canet@univ-lyon1.fr
INFOMATHS	**INFORMATIQUE ET MATHEMATIQUES** **http://infomaths.univ-lyon1.fr** Sec :Renée EL MELHEM Bat Blaise Pascal 3e etage infomaths@univ-lyon1.fr	Mme Sylvie CALABRETTO LIRIS – INSA de Lyon Bat Blaise Pascal 7 avenue Jean Capelle 69622 VILLEURBANNE Cedex Tél : 04.72. 43. 80. 46 Fax 04 72 43 16 87 Sylvie.calabretto@insa-lyon.fr
Matériaux	**MATERIAUX DE LYON** **http://ed34.universite-lyon.fr** Sec : M. LABOUNE PM : 71.70 –Fax : 87.12 Bat. Saint Exupéry Ed.materiaux@insa-lyon.fr	M. Jean-Yves BUFFIERE INSA de Lyon MATEIS Bâtiment Saint Exupéry 7 avenue Jean Capelle 69621 VILLEURBANNE Cedex Tél : 04.72.43 71.70 Fax 04 72 43 85 28 Ed.materiaux@insa-lyon.fr
MEGA	**MECANIQUE, ENERGETIQUE, GENIE CIVIL, ACOUSTIQUE** **http://mega.universite-lyon.fr** Sec : M. LABOUNE PM : 71.70 –Fax : 87.12 Bat. Saint Exupéry mega@insa-lyon.fr	M. Philippe BOISSE INSA de Lyon Laboratoire LAMCOS Bâtiment Jacquard 25 bis avenue Jean Capelle 69621 VILLEURBANNE Cedex Tél : 04.72 .43.71.70 Fax : 04 72 43 72 37 Philippe.boisse@insa-lyon.fr
ScSo	**ScSo*** **http://recherche.univ-lyon2.fr/scso/** Sec : Viviane POLSINELLI Brigitte DUBOIS Insa : J.Y. TOUSSAINT viviane.polsinelli@univ-lyon2.fr	Mme Isabelle VON BUELTZINGLOEWEN Université Lyon 2 86 rue Pasteur 69365 LYON Cedex 07 Tél : 04.78.77.23.86 Fax : 04.37.28.04.48

*ScSo : Histoire, Géographie, Aménagement, Urbanisme, Archéologie, Science politique, Sociologie, Anthropologie

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Remerciements

TODO.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Table des matières

1 Introduction **9**
1.1 Contexte et objectifs généraux . . . 10
1.1.1 Flux et requêtes . . . 10
1.1.2 Enjeux sociétaux . . . 11
1.2 Motivations, cas d'utilisation . . . 12
1.2.1 Flux d'informations connus et recherche Web . . . 12
1.2.2 Création collective . . . 14
1.2.3 Veille scientifique . . . 14
1.3 Approche QTor : intuitions . . . 15
1.3.1 Communautés d'intérêt . . . 16
1.3.2 Mise en relation des communautés . . . 16

2 Étude bibliographique **19**
2.1 Critères d'analyse . . . 19
2.1.1 Éléments de classification . . . 19
Généricité vis-à-vis du langage de requêtes . . . 20
Ressources utilisées . . . 22
2.1.2 Éléments d'évaluation . . . 25
2.2 Modélisation . . . 29
2.2.1 Composants de base d'un système générique . . . 29
2.2.2 Représentation logique : graphes et hypergraphes . . . 31
2.2.3 Structures reconnaissables . . . 34
2.3 Systèmes guidés par la diffusion . . . 36
2.3.1 Systèmes non-basés sur le langage . . . 36
Systèmes non-portés par les utilisateurs·trices . . . 36
Mise à contribution des utilisateurs·trices . . . 37
Modélisation commune . . . 38
2.3.2 Systèmes basés sur l'équivalence des requêtes . . . 40
2.4 Systèmes guidés par les calculs . . . 42
2.4.1 Systèmes à organisation spécifique à un langage particulier . . . 42
Systèmes centralisés et organisation locale des calculs . . . 43
Répartition du calcul sur des nœuds système . . . 43
Mise à contribution des participants exprimant les requêtes . . . 44
Modélisation et analyse . . . 45
2.4.2 Systèmes basés sur l'extraction des opérateurs . . . 46
2.4.3 Systèmes basés sur les réécritures . . . 48
Notion de réécritures de requêtes . . . 48
Échanges de flux et réécritures locales . . . 51
Semantic Peer-to-Peer Overlays for Publish/Subscribe Networks . 52

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Delta . . . 53
Modélisation commune . . . 56
Synthèse bibliographique . . . 57

3 Proposition : système à torrent de requêtes **61**
3.1 Formalisation du problème . . . 61
3.1.1 Fonctions d'organisation . . . 61
3.1.2 Contraintes et objectifs . . . 63
3.2 Approche QTor . . . 64
3.2.1 Système communautaire basé sur les requêtes . . . 65
3.2.2 Relations entre communautés : réécritures et modèle de coût . . . 67
3.2.3 Échange de ressources entre communautés . . . 69
3.3 Mise en œuvre incrémentale . . . 71
3.3.1 Abstraction : gestion de l'hypergraphe des communautés . . . 72
Maintien de l'ensemble des communautés . . . 73
Modification des relations existantes entre communautés . . . 76
Distribution de l'organisation abstraite . . . 78
3.3.2 Concrétisation : relations entre les participants . . . 79
Mise en relation des communautés . . . 80
Concrétisation de la diffusion des résultats . . . 83
Distribution de l'organisation concrète . . . 84
3.4 Propriétés et analyse . . . 85
3.4.1 Étude des coûts de l'organisation . . . 85
3.4.2 Répartition des calculs et conditions d'optimalité . . . 87
3.4.3 Cas de l'absence d'un mécanisme de réécritures . . . 91
3.4.4 Adaptabilité de la proposition . . . 92
Adaptabilité à la popularité des requêtes . . . 92
Adaptabilité à la connexité des requêtes . . . 93
Adaptabilité aux évolutions des flux . . . 94
Adaptabilité aux capacités des participants . . . 95
Synthèse de l'évaluation théorique . . . 97

4 Évaluation expérimentale **99**
4.1 Environnement expérimental . . . 99
4.1.1 Éléments comparés . . . 99
4.1.2 Notre démonstrateur, FleDDi . . . 101
4.1.3 Jeux de test type . . . 103
4.2 Évaluation dans le cadre général . . . 105
4.2.1 Coût d'organisation du système . . . 106
4.2.2 Observation de l'organisation du système . . . 107
4.2.3 Évaluation du fonctionnement du système . . . 110
4.3 Influence de la variation de popularité . . . 111
4.3.1 Étude en popularité homogène . . . 111
4.3.2 Cas limites de l'homogénéité . . . 113
4.3.3 Variations dynamiques de la popularité . . . 114
4.4 Influence de variation des capacités . . . 117
4.4.1 Variations côté source . . . 117
4.4.2 Variations côté requêteurs . . . 119

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

5 Conclusion **125**
5.1 Synthèse des travaux réalisés . . . 125
5.1.1 Contexte et état d'avancement . . . 125
5.1.2 Communication et usages . . . 127
5.2 Pistes d'améliorations et perspectives . . . 127
5.2.1 Généralisation du modèle . . . 127
5.2.2 Extensions de l'organisation . . . 128
5.2.3 Intimité numérique et contrôle d'accès . . . 130
Synthèse générale . . . 132

Annexes **133**

1 Organisation communautaire **135**
1.1 Modélisation interne d'une communauté . . . 135
1.1.1 Répartition des tâches . . . 136
1.1.2 Conséquences sur les propriétés du système . . . 137
1.2 Diffusion tenant compte des communautés enfants . . . 138
1.2.1 Notion de poids d'une unité . . . 138
1.2.2 Effet sur la latence globale . . . 139
1.2.3 Problématiques de mise en place . . . 141

2 Langages et jeux de données **143**
2.1 Filtres par mots-clefs et combinaisons conjonctives . . . 143
2.1.1 Analyse des requêtes et réécritures . . . 143
2.1.2 Requêtes et données utilisées . . . 145
2.2 Filtres par mots-clefs et combinaisons disjonctives . . . 145
2.2.1 Analyse des requêtes et réécritures . . . 146
2.2.2 Requêtes et données utilisées . . . 147
2.3 Langage pseudo-SQL pour flux issus de capteurs . . . 148
2.3.1 Données utilisées et syntaxe des requêtes . . . 148
2.3.2 Mécanisme de réécritures . . . 149

3 Démonstrateur **151**
3.1 Analyse et traitement des requêtes . . . 151
3.1.1 Interfaces . . . 151
3.1.2 Paramétrage des implémentations . . . 154
3.2 *Tracker* et organisation du système . . . 155
3.2.1 Mécanismes généraux . . . 155
3.2.2 Exemple simple : Unicast . . . 156
3.2.3 Multicast et arbres de diffusion . . . 157
3.2.4 SemPO et mise en cascade . . . 157
3.2.5 QTor, FQTor et communautés . . . 158
3.3 Échanges réseau réels ou simulés . . . 158
3.3.1 Mécanismes généraux . . . 159
3.3.2 Communication directe entre objets Java . . . 161
3.3.3 Simulateur basé sur PeerSim . . . 161
3.3.4 Prototype pour déploiement réel . . . 162
3.4 Tests, classes principales et instrumentation . . . 162
3.4.1 Classe centrale du simulateur . . . 162

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

3.4.2 Points d'entrée pour l'interpréteur 163
3.4.3 Interface graphique 164
3.4.4 Gestion des logs 165

4 Documentation fonctionnelle **167**
Bootstrap 167
QTor 167
Traitement des requêtes 167
Gestion du graphe 168
Query 171
Participant 171
Gestion des unités 171
Communication avec le système 172
Unit 173
Informations sur les fonctions de l'unité 173
Informations de mise en relation 174
Informations sur les ressources disponibles 174
Community 175
Informations publiques sur la communauté 175
Communication avec le système 175
Gestion du placement dans le graphe 177
Organisation interne 179

5 Ma thèse... en BD ! **183**

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Chapitre 1

Introduction

Amorcé à la fin du siècle dernier, l'essor du numérique et de l'accès à Internet induit des bouleversements notables dans nos sociétés. L'un des exemples les plus flagrants de ces dernières années a été celui des mouvements regroupés sous le terme de *printemps arabe*, où l'usage des réseaux sociaux a conduit à des changements de gouvernance majeurs dans certains pays. D'une manière générale, ces bouleversements liés au numérique sont comparés[85] à ceux amenés par l'invention de l'écriture marquant la fin de la Préhistoire, et par celle de l'imprimerie durant la Renaissance, leur conférant le statut de troisième grande révolution dans le rapport au savoir de l'humanité.

L'un des changements majeurs apportés par Internet, par rapport aux outils précédents (imprimerie, puis radiophonie et télévision), est que sa structure est beaucoup moins verticale : là où seuls quelques privilégiés disposaient des moyens requis pour faire imprimer et diffuser leurs idées, Internet permet d'apporter cette possibilité à l'ensemble de la population[1]. On assiste donc à une augmentation massive de la production de données (*Big Data*), avec toutes les problématiques que cela entraîne ; mais également à une prise en compte de plus en plus importante de la possibilité, pour les utilisateurs·trices, de contribuer au bon fonctionnement des systèmes.

Dans ce contexte, ce document décrit les travaux réalisés dans le cadre de la thèse intitulée « Approche communautaire sur l'évaluation de requêtes », portant sur la mise en place d'un système distribué de traitement de requêtes continues sur des flux de données, dont l'objectif est d'être simple à mettre en place, adaptable et respectueux des intérêts de ses participants.

Ce chapitre introductif est consacré à poser les bases nécessaires à la bonne compréhension tant de nos motivations que des éléments principaux ayant guidé notre façon de procéder, lesquels sont repris et formalisés dans les chapitres ultérieurs. Pour cela, nous présentons en premier lieu les objectifs généraux de nos travaux, compte tenu du contexte, afin de poser les premières intuitions du problème que notre approche vise à résoudre. Nos proposons ensuite diverses situations pouvant requérir l'usage de la solution que nous proposons. Enfin, la dernière section de ce chapitre présente, de manière succincte, les grandes lignes de cette solution, et montre en quoi son utilisation est adaptée à ces différents cas.

[1] Il convient toutefois de noter que les problématiques d'accès à Internet sont encore importantes au moment de la parution de ce document, pour des raisons qui ne sont pas uniquement financières. En particulier, le passage à la version 6 de l'*Internet Protocol* est un des chantiers majeurs à terminer actuellement, entre autres pour l'augmentation de l'espace d'adressage qu'elle apporte.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

1.1 Contexte et objectifs généraux

Les outils numériques de plus en plus puissants, y compris et surtout au domicile des particuliers, ainsi que l'augmentation des débits, en particulier de téléversement[2], permettent donc aux usagers de prendre en charge le fonctionnement des systèmes distribués, qui ne pouvaient jusqu'alors fonctionner que grâce à des ressources spécialisées.

1.1.1 Flux et requêtes

Dans ce contexte, le problème général qui nous préoccupe ici porte sur les requêtes *exprimées* les utilisateurs·trices pour spécifier quelles informations doivent être obtenues à partir de *flux continus d'information*. Ces requêtes peuvent s'avérer potentiellement assez complexes à traiter, surtout si l'on tient compte des caractéristiques de vitesse et de volumétrie propres aux masses de données. Organiser un système constitué d'un grand nombre de ces requêtes s'avère donc une tâche délicate.

Ce problème général, d'importance croissante depuis le début de l'essor d'Internet[10], est abordé dans la littérature de différentes manières. Dans le domaine général des bases de données, cela s'apparente au cas des requêtes continues (CQ, pour « *Continuous Queries* » en anglais), ainsi qu'aux systèmes de gestion et de traitement de flux (DSMS et DSPS, pour, respectivement, « *DataStream Management Systems* » et « *Distributed Stream Processing Systems* »). Le problème est également abordé dans les domaines réseau, et notamment dans le monde des échanges pair-à-pair, particulièrement par les systèmes de souscriptions aux publications (Pub/Sub, pour « *Publish/Subscribe* »).

D'une manière générale, les systèmes constitués dans ce cadre sont construits autour de *sources primaires de données*, qui génèrent et émettent les flux d'informations sur lesquels sont exprimées les requêtes des utilisateurs·trices. Organiser un tel système demande de déterminer comment connecter entre eux les différents participants, que sont les sources, les utilisateurs·trices et d'éventuelles ressources supplémentaires, en tenant compte des capacités de chacun et des relations existant entre les différentes requêtes, afin de permettre à chaque utilisateur·trice de recevoir les informations demandées.

Naturellement, un système dans lequel les résultats obtenus ne correspondraient pas aux requêtes exprimées est à éviter absolument. Le fait qu'un des participants du système voit ses capacités surchargées entraînerait par ailleurs des dysfonctionnements notables, rendant également l'utilisation problématique. À ces contraintes majeures s'ajoutent usuellement d'autres critères, tels que le fait que les résultats puissent être obtenus dans des délais raisonnables, et que la participation, si elle est requise, n'aille pas à l'encontre du souhait des utilisateurs·trices.

La mise en place d'un système fonctionnel présente en outre d'autres difficultés, dont en particulier le nombre potentiel d'utilisateurs·trices exprimant des requêtes, qui peut s'avérer particulièrement important, et donc entraîner de graves problèmes de passage à l'échelle s'il n'est pas correctement pris en compte. La complexité des requêtes et de l'hétérogénéité des capacités tendent d'ailleurs à accentuer ces difficultés.

[2] L'augmentation des débits s'est d'abord faite de manière asynchrone, très forte pour le téléchargement et très modérée pour le téléversement, ce qui, compte tenu de l'augmentation des besoins[71], limite la capacité des utilisateurs·trices à émettre des données et entraîne donc une reproduction des modèles verticaux précédents. Le déploiement à grande échelle de la fibre optique devrait permettre de pallier ce problème... du moins pour les opérateurs qui ne brident pas leurs abonnements de manière arbitraire.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

1.1.2 Enjeux sociétaux

En complément des aspects purement techniques, la localisation de la prise en charge du fonctionnement des systèmes (chez les utilisateurs·trices ou sur des ressources tierces) présente des aspects sociétaux majeurs, en grande partie liés à la verticalité ou non des modes de diffusion.

Historiquement, les situations dans lesquelles le fonctionnement du système n'était pas porté par ses usagers présentaient en premier lieu des problèmes relatifs au passage à l'échelle, un tiers centralisateur unique représentant un goulot d'étranglement, se trouvant rapidement débordé dès que le nombre de participants devient trop important. Si les progrès techniques réalisés en matière de traitements distribués ont depuis permis aux acteurs majeurs du domaine de consolider leur infrastructure pour fortement réduire ces problèmes, cela n'a pu se faire qu'au prix d'un coût financier trop important pour être accessible à toute personne souhaitant jouer ce rôle. Ceci entraîne alors un retour à une situation dans laquelle seules quelques entités privilégiées peuvent disposer des nouvelles formes de presses à imprimer, tandis que l'ensemble de la population n'a que la possibilité de lire ce qui est produit par d'autres, perdant l'un des apports majeurs du numérique.

Par ailleurs, il convient de noter que ces solutions nécessitent des parcs de machines suffisamment importants pour soutenir une charge d'autant plus accrue que cela entraîne une concentration des services et une augmentation du nombre d'échanges. Les besoins énergétiques pour alimenter et refroidir les machines sont notables, et l'impact environnemental provoqué par la multiplication des *datacenter* risque donc de s'avérer tout sauf négligeable (quand l'influence humaine sur le climat est déjà problématique). Un ensemble de machines géré directement par l'ensemble de la société permettrait un gain sensible à ce sujet, du fait que ces machines, mieux réparties, seraient moins sollicitées.

De plus, la présence d'un tiers centralisateur, en charge à la fois de collecter les informations des sources, si celles-ci sont nombreuses, et de réaliser seul le traitement des requêtes, afin d'envoyer directement aux utilisateurs·trices les résultats désirés, présente également des problèmes de respect de l'intimité numérique[3]. Ce tiers centralisateur dispose en effet d'une connaissance à la fois exclusive et exhaustive de l'ensemble du système : toute l'information produite par les sources de données doit passer par lui ; et il doit connaître les intentions précises de chaque utilisateur·trice. Ces données peuvent ensuite être collectées à des fins commerciales (dont le tiers centralisateur, en position de monopole, est le seul à tirer bénéfice) ; mais également à des fins de surveillance des populations (les révélations récentes concernant les pratiques de la NSA, par exemple, ont mis en lumière un système massif d'espionnage des populations, fort problématique, pour un résultat plus que limité en matière de sécurité).

Assez naturellement, les situations de ce type, communes à de nombreux domaines, entraînent, au sein des groupes militants concernés, un mouvement de rejet[42], incitant les citoyen·ne·s à reprendre le contrôle de leur informatique personnelle. L'*auto-hébergement*, c'est-à-dire le fait de disposer de ses propres ressources en charge de gérer les tâches numériques plutôt que de les déléguer à de tels tiers, est généralement perçu comme un élément de solution majeur à ce problème.

L'auto-hébergement a longtemps présenté des difficultés de mises en place notables pour certains domaines (notamment pour ce qui concerne le courrier électronique), le réservant de fait à un public plutôt qualifié techniquement. Des progrès majeurs ont cependant été réalisés dans ce domaine au cours de ces dernières années, par exemple par

[3] À ce terme est parfois préféré celui de « vie privée ». Quoique les deux se rattachent au concept parfois également désigné par le terme anglais *privacy*, la notion d'intimité nous semble ici plus représentative.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

la conception de la Brique Internet[93, 56], un mini-serveur de faible consommation conçu spécifiquement pour être aisément pris en main sans connaissances particulières.

En parallèle, les publications scientifiques des domaines concernés s'orientent de plus en plus vers la prise en compte des utilisateurs·trices comme participants actifs au système, et non plus simplement comme récepteurs passifs des données demandées. La conjonction de ces deux tendances permet d'envisager la mise en place de réseaux massivement distribués, conçus pour pallier les difficultés présentées ci-dessus, avec une pérennité assurée par le fait que ce sont directement les personnes ayant besoin du système qui en assurent le bon fonctionnement, quand les tiers centralisateurs peuvent finir par disparaître.

Quoique la différence éthique soit d'importance cruciale, on peut, en un sens, voir une continuité technique entre la distribution du fonctionnement sur des nœuds systèmes, étape qui a permis à ces tiers centralisateurs de gérer le nombre toujours accru d'utilisateurs·trices de leurs services, et la distribution pair-à-pair incluant ces dernièr·es. Toutefois, comme cela a été évoqué, le pas technique entre ces deux étapes peut s'avérer aussi important que celui amenant du pur centralisé au (semi-)distribué, compte tenu du fait qu'un réseau pair-à-pair est nettement plus hétérogène sur tous les aspects, avec des contraintes de capacités nettement plus strictes (quoiqu'elle suffise amplement pour des usages classiques tels que le mail ou le blog, une Brique Internet n'a naturellement pas la puissance d'une machine de *datacenter*).

1.2 Motivations, cas d'utilisation

Conscients de ces enjeux, nous avons choisi de porter notre attention sur le domaine de la diffusion des données et du traitement distribué des requêtes pour plusieurs raisons majeures : d'une part, ce domaine est d'importance cruciale en ce qui concerne l'accès à l'information. L'activité journalistique, par exemple, se manifeste naturellement sous la forme de flux continus, l'impossibilité pour l'humain·e moyen·ne de s'intéresser simultanément à l'ensemble des sujets existants et la nécessité de croiser les sources d'information entraînant le besoin d'exprimer des requêtes potentiellement complexes sur ces flux. La possibilité, pour les participants, de partager leurs résultats et de se regrouper autour de thématiques communes est par ailleurs une étape permettant de mieux s'approprier l'information, et ainsi de sortir d'un rôle de spectateur passif, vertical, tel que celui imposé par les médias précédents.

D'autre part, enfin, une approche s'inspirant des bonnes pratiques développées dans le domaine des bases de données, qui est notre domaine initial, nous a semblé en mesure de répondre à ce problème, par le fait que certains des éléments considérés, notamment en ce qui concerne la complexité des requêtes, y sont précisément assez étudiés. Afin de préciser cet aspect, nous proposons quelques exemples de cas d'utilisation présentant les problèmes auxquels nous envisagions apporter une réponse utile.

1.2.1 Flux d'informations connus et recherche Web

Assez naturellement, le premier cas d'utilisation que nous considérons concerne donc la problématique de l'accès à l'information. On peut envisager, par exemple, la présence d'un ou de plusieurs sites de journaux en ligne[4], diffusant leurs articles et dépêches au fur et à mesure de leur parution, par l'intermédiaire par exemple de flux RSS.

[4] Mais également de nombreux autres médias tels que les blogs et forums, l'une des conséquences des changements sociétaux étant le fait que l'activité journalistique (en particulier le travail de vérification et de mise à disposition des informations) n'est plus nécessairement réservée aux seuls journalistes de profession.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Dans ce contexte, les utilisateurs·trices effectuent leur revue de presse personnelle, en commençant en premier lieu par filtrer les articles portant sur des sujets qui les intéressent. Ce filtrage peut parfois se faire à l'aide des mots-clefs renseignés dans les différentes entrées du flux RSS ; mais ces mots-clefs n'étant pas toujours bien spécifiés, et la description fournie pouvant être trop résumée, il est plus vraisemblablement nécessaire, pour un résultat plus pertinent, d'aller inspecter le corps du document.

Chaque journal ayant sa propre ligne éditoriale, les différents articles publiés sont le fruit d'une grille de lecture particulière, et certain·e·s internautes, soucieux·ses de se faire une idée aussi objective que possible de la situation sur les sujets qui leur tiennent a cœur, ont tendance à croiser les sources d'informations, et donc à lire des articles issus de plusieurs journaux différents. Pour d'autres internautes, en revanche, les sites d'informations partageant des points de vue particulièrement divergents sont d'intérêt bien moindre, et seuls les résultats de journaux proches seront à prendre en compte.

Un tel contexte amène donc des requêtes assez variées et d'expressivité assez importante, mais avec néanmoins des sujets phares qui se retrouveront communs à de nombreuses requêtes sur des périodes plus ou moins longues, en fonction des aléas de l'actualité (par exemple, la combinaison « volcan » et « avion » au moment des perturbations de trafic aérien entraînées par l'éruption de l'Eyjafjöll[5] en 2010 ; ou bien « état d'urgence » depuis les attentats du 13 novembre 2015 en France).

Ce cas peut en fait assez aisément être généralisé à celui de la *découverte* d'information, permettant de trouver des sources nouvelles et non-encore connues, comme peuvent le proposer certains moteurs de recherche avancés. Il suffit dans ce cas d'envisager que le robot d'indexation chargé de parcourir le Web pour référencer ces nouveaux sites publie lui-même un flux indiquant ses découvertes, et se mette donc en position de source. Ce flux devrait contenir un volume de données assez conséquent, la plupart d'entre elles n'étant pas pertinentes, mais des requêtes suffisamment ciblées permettraient d'en extraire les nouveaux sites correspondant aux attentes des utilisateurs·trices.

Alternativement, on peut considérer que les personnes mettant à disposition ces nouveaux sites, elles-mêmes soucieuses de rencontrer leur public, fassent la démarche de se faire référencer dans le système sans la présence de robots d'indexations, ce qui requiert donc la présence d'un mécanisme permettant d'ajouter dynamiquement de nouvelles sources primaires au système.

Dans tous les cas, ces participants peuvent se voir demander de partager les articles récupérés (et identifiés comme correspondant aux requêtes exprimées) afin de permettre aux autres utilisateurs de les obtenir. Cela sous-entend généralement un partage fonctionnant de manière dynamique (un article pertinent est diffusé aussitôt qu'il est identifié) et sans mémoire. Il serait cependant, à titre de perspective, envisageable de l'étendre pour demander aux participants de conserver une copie de ces articles aussi longtemps que possible, ce qui pourrait permettre à des utilisateurs·trices ne participant pas nécessairement au système d'émettre des requêtes ponctuelles pour obtenir de l'information, sans avoir pour cela besoin de recourir aux énormes bases de données que doivent mettre en place les moteurs de recherche actuels.

Toutefois, si une telle situation suppose des utilisateurs·trices actif·ve·s du point de vue du strict fonctionnement du système (requêtage, mise à disposition de ressources de traitement et partage des résultats), leur attitude demeure globalement passive vis-à-vis de la production d'information.

Quelques services centralisés, tels que Google Alerts ou TalkWalker, proposent déjà des cas d'utilisation relativement similaires ; mais présentent toutefois les inconvénients

[5] « Eyjafjallajökull » est en fait le nom du glacier surplombant ce volcan islandais.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

décrits plus tôt en termes d'enjeux sociétaux.

1.2.2 Création collective

Il est également possible d'envisager des cas où le traitement de requêtes et la diffusion d'informations, plutôt qu'autonomes, seraient partie intégrante d'un système communautaire plus vaste, dans lequel les usagers contribuent également à cette production. À titre d'exemple, considérons un réseau dédié à l'observation de la faune et de la flore. Les membres de ce réseau se chargent de publier manuellement leurs observations sous la forme de messages courts, contenant par exemple l'espèce observée et le lieu d'observation (coordonnées GPS).

Un tel réseau, alimenté en faits par ses utilisateurs·trices, peut être utilisé avec des objectifs variés. Ainsi, les passionné·e·s d'ornithologie peuvent l'utiliser pour identifier les espèces d'oiseaux qu'il est possible d'observer à proximité de leur lieu de résidence ou de villégiature, et les lieux où leurs chances d'effectuer ces observations seront les plus élevées ; tandis que les autorités sanitaires peuvent suivre la progression d'animaux porteurs de maladie, ou au contraire la régression d'espèces sensibles.

Alternativement, on peut considérer le cas de documents collaboratifs de taille plus importante tels que, par exemple, ceux que l'on peut produire à l'aide de CRATE[72], développé dans le cadre du projet SocioPlug[6] dans lequel s'inscrit également une partie des travaux présentés dans ce document.

Ce projet vise en effet à permettre la mise en place d'un ensemble de services de type dit de « *Cloud computing* » sur un réseau de mini-serveurs (ou *plugs*) chacun possédé par un·e utilisateur·trice du réseau. Dans ce cadre, CRATE est un outil d'édition collaborative de documents, comme peuvent l'être des logiciels comme Gobby ou Etherpad, mais qui est conçu pour fonctionner entièrement en pair-à-pair, sans serveur intermédiaire. Une communauté utilisant un tel outil a donc également besoin de faire appel à un mécanisme de recherche distribuée pour gérer la diffusion des documents ainsi produits.

Dans un cas comme dans l'autre, nous avons donc des utilisateurs·trices qui sont en charge de produire les données qui seront diffusées dans le système, et qui ont donc d'autant plus intérêt à permettre aux autres personnes intéressées d'accéder aux informations, et, pour ce faire, à fournir au système de requêtage et de diffusion les ressources qui lui seront nécessaires pour fonctionner. Une difficulté majeure est, dans ce cas, que ces participants basent essentiellement leur participation au réseau sur des machines de relativement faible puissance (*plugs* et ordiphones), faisant de la façon de répartir les ressources un aspect critique.

1.2.3 Veille scientifique

Un troisième cas d'usage à envisager est celui de la recherche scientifique, avec son flot de publications nouvelles. Ces publications se font par des sources variées, mais clairement identifiées, que sont les différents journaux, conférences, *workshops* et sites d'archives. Les chercheur·se·s réalisent leurs travaux, rédigent des articles en présentant les résultats, et soumettent ces articles à la relecture de leurs pairs. Une fois les articles validés, ils sont publiés pour permettre à l'ensemble de la communauté scientifique d'en prendre connaissance.

Ce cas d'utilisation correspond au domaine que nous considérons, dans la mesure où les travaux scientifiques doivent tenir compte de l'état de l'art de leur domaine, ce qui requiert

[6] Projet de l'Agence Nationale de la Recherche française, N^o ANR-13-INFR-0003.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

de réaliser une veille pour se tenir informé·e·s des nouvelles publications portant sur des sujets proches des leurs. Ces veilles peuvent donc prendre la forme de requêtes plus ou moins complexes permettant d'identifier les papiers pertinents. Quoique la communauté scientifique soit globalement partitionnée en différents domaines, les frontières en sont loin d'être hermétiques, entraînant la nécessité de consulter des sources multiples. Par exemple, des travaux autour d'un système de requêtage et de diffusion peuvent requérir de suivre les publications aussi bien dans le domaine des bases de données distribuées que dans celui des échanges pair-à-pair, et un papier à ce sujet peut avoir besoin de citer des auteurs de philosophie et de micro-économie.

Extraire les informations demandées du document (distinguer, par exemple, les articles externes référencés de la contribution principale) demande une analyse poussée, et donc des ressources, qui ne doivent naturellement pas empiéter sur celles requises pour les travaux scientifiques eux-mêmes. Les chercheur·se·s ont cependant tout intérêt à partager et à faire connaître leurs propres travaux ; mais également ceux des travaux de leurs collègues qui s'appuient sur les leurs – un bon exemple de requête possible dans un tel cas est d'ailleurs celui de chercher les publications qui vont référencer l'un de nos articles, ce qui n'a *a priori* pas de limites dans le temps (des articles datant des années 1980 sont encore cités dans des documents de 2016).

Une des limites majeures au partage d'articles scientifiques est actuellement le fait qu'une proportion importante des publications ne sont pas encore diffusées en *OpenAccess*[27], mais sont au contraire soumises à des contraintes d'accès telles qu'un abonnement (potentiellement onéreux) à la revue dans laquelle ils sont parus. Un mécanisme de traitement des requêtes complexes, permettant d'assurer que les papiers demandés correspondent le mieux possible aux attentes, permet de diminuer les risques de devoir payer pour accéder à des articles qui s'avèrent finalement n'avoir aucune pertinence vis-à-vis du problème examiné. Par ailleurs, il est possible d'envisager, par certaines contraintes sur l'organisation, d'éviter de demander aux participants ne disposant pas des accès requis de devoir manipuler et partager les documents réservés.

1.3 Approche QTor : intuitions

Afin de pouvoir répondre efficacement au problème considéré, notamment dans ces différents cas d'utilisation, nous avons choisi de baser nos travaux sur un aspect qui nous semble essentiel : celui de la popularité des requêtes. En effet, dans chacun de ces cas (sujets d'actualités impactant de nombreuses personnes, zones géographiques plus ou moins peuplées, domaines scientifiques porteurs…), on peut s'attendre à ce que de nombreux participants expriment des requêtes proches, ce qui a tout avantage à être pris en compte dans l'organisation des requêtes. Il est connu[68] que cette popularité se retrouve, à importances variables, dans la plupart des recherches effectuées sur Internet.

Ces variations de popularité des requêtes entraînent une autre forme d'hétérogénéité pouvant elle aussi contribuer à l'importante complexité du problème. Toutefois, il est également possible de considérer que, correctement prises en compte, elles peuvent au contraire s'avérer une clef permettant de désamorcer grandement cette complexité. En effet, celle-ci est en particulier due au fait que les relations qui seront déterminées entre les participants dépendent à la fois d'aspects logiques (l'utilisabilité des données envoyées) et physiques (les contraintes de capacités des participants concernés). Nous avons choisi de considérer que la popularité des requêtes pouvait être utilisée afin de séparer ces deux aspects, et donc de diviser le problème en deux tâches plus simples.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

1.3.1 Communautés d'intérêt

La première étape, pour cela, est de considérer les requêtes qui, qu'elles soient rigoureusement identiques ou formulées de manière différente, vont s'avérer être strictement *équivalentes* au sens mathématique du terme, c'est-à-dire qu'à situation identique, elles produiront systématiquement les mêmes résultats, quels que soient les sources primaires de données et les flux qu'elles produisent.

Les participants partageant ces requêtes équivalentes peuvent donc, assez naturellement, être regroupés au sein de *communautés d'intérêt*, chaque communauté étant dédiée à une classe d'équivalence particulière de requêtes. Au sein de ces communautés, tous les participants présents ont des intérêts convergents (chacun d'entre eux a besoin de recevoir les mêmes résultats), et peuvent donc être amenés à collaborer pour permettre l'accomplissement de ce but commun.

En particulier, le problème de l'acquisition des données est simplifié par le fait qu'une communauté comptant un millier de participants n'aura pas besoin de récupérer un millier d'exemplaires de chaque élément des flux d'entrée ; mais pourra au contraire travailler de concert à partir d'un nombre restreint de données partagées. La charge des sources, ou des autres participants chargés de transférer les données, peut donc être d'autant réduite.

Quoique ce point ne soit pas l'objet de ce document, car un travail séparé[54] (s'inscrivant également dans le cadre du projet SocioPlug) y est consacré, l'organisation en communautés peut aussi permettre de distribuer la charge de calcul : dans le cas où la requête à traiter serait hors de portée de chaque intervenant pris individuellement, en raison de la complexité de la requête et/ou du volume de flux d'entrée, il est tout à fait possible à ces participants de se répartir le travail, divisant le flux initial ou se partageant les différents opérateurs, afin que la charge de traitement ne dépasse pas les capacités de chacun.

Enfin, les membres de ces communautés, une fois les résultats de leur requête obtenus, peuvent s'en partager la diffusion, s'assurant que chaque participant reçoit bien les informations requises. La prise en compte des restrictions d'accès peut être envisagée à ce niveau, considérant qu'elles conduisent à des classes d'équivalence de résultats différentes, même pour des requêtes par ailleurs similaires.

En d'autres termes, chaque communauté forme un sous-système autonome, de nombre de participants réduit, et dans lequel ces participants peuvent être organisés entre eux sans avoir à tenir compte du reste du système, les seules considérations à prendre en compte sont celles de capacités et de latence, les autres aspects étant communs.

1.3.2 Mise en relation des communautés

Mais les variations de popularité ne concernent pas uniquement la stricte équivalence des requêtes : la plupart du temps, il est possible de trouver plusieurs requêtes qui, quoique différentes l'une de l'autre, s'intéressent à des sujets communs et présentent donc certaines similitudes dans leurs résultats. Le regroupement des participants en communautés n'est donc qu'une première étape, ces communautés pouvant ensuite être reliées entre elles pour pouvoir bénéficier des travaux préalables effectués par les autres.

Ainsi, chaque fois que les résultats d'une requête donnée permettent de calculer une autre requête plus efficacement qu'en allant chercher le flux brut initial, les communautés concernées peuvent être mises en relation, ce qui permet ainsi d'éviter un grand nombre de traitements redondants. Nous retombons donc ici sur un problème de prise en compte des relations entre requêtes et de mise en relation de différents intervenants ; mais ce problème est d'autant plus réduit que les requêtes sont populaires, car il n'y a à considérer que les différentes communautés et non l'ensemble complet des participants.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

L'usage de communautés forme ainsi une *abstraction* guidant l'organisation : dans un premier temps, les relations entre requêtes sont utilisées pour déterminer les communautés et liens à former entre elles ; puis les décisions prises au niveau des communautés sont *concrétisées* sur les participants, par l'organisation interne de ces communautés. Cela conduit donc à diviser le problème en deux aspects complémentaires, chacun étant de taille réduite. De plus, de la même manière que le problème de l'organisation interne permettait de s'affranchir des problématiques de relations entre requêtes, le problème de la mise en relation des communautés permet de s'affranchir des problématiques de capacités, dans la mesure où les participants, tous tributaires du bon fonctionnement du système général, peuvent être amenés à envoyer des ressources (que l'organisation globale et la diminution de la redondance des calculs leur permettent d'économiser) vers les autres communautés.

En d'autres termes, une approche basée sur des communautés d'intérêt mises en relations les unes avec les autres permet un système collaboratif, ouvert et respectueux des intérêts de chaque intervenant, dont le fonctionnement ira « de chacun, selon ses moyens, à chacun, selon ses besoins. »

Ce document présente nos travaux relatifs à la mise en place d'un système de traitement de requêtes et de diffusion de données qui soit ouvert, porté par les utilisateurs et respectueux de leurs intérêts comme des limites de leurs capacités. Il est articulé en cinq chapitres principaux : la présente *introduction*, présentant le contexte et les intuitions, suivie d'une *étude bibliographique* incluant une modélisation commune aux différentes propositions de l'état de l'art, puis une présentation de l'approche que nous proposons, la définition d'un *système à torrent de requêtes*. Nous présentons ensuite les *évaluations expérimentales* réalisées ainsi que leurs résultats, avant de présenter nos *conclusions*.

Plusieurs annexes viennent compléter ces chapitres principaux sur les points relatifs à l'*organisation communautaire*, aux *langages et jeux de données* utilisés, au fonctionnement de notre *démonstrateur* et aux *algorithmes et protocoles* sur lesquels repose notre approche. Enfin, l'annexe intitulée *Ma thèse... en BD !* présente un résumé étendu de nos travaux sous format graphique.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Chapitre 2

Étude bibliographique

Ce second chapitre est consacré à l'étude des différents travaux présents dans la littérature qui répondent à des problèmes analogues à celui auquel nous nous intéressons. Comme mentionné au chapitre précédent, on trouve de telles propositions dans divers domaines, relevant autant du monde des bases de données que de celui des échanges réseau. Pour autant, passer en revue ces propositions en les regroupant selon leur domaine d'origine ne nous semble pas le plus pertinent, dans la mesure où les différences et proximités entre ces approches transcendent les domaines de publication. Nous proposons donc une nouvelle catégorisation des approches existantes.

Puisque nous proposons une nouvelle classification, il est nécessaire de présenter les critères sur lesquels celle-ci repose. La première section de ce chapitre est donc dédiée à ce but, ainsi qu'à présenter les critères sur lesquels nous évaluons ces propositions. La seconde section présente ensuite une modélisation générique que nous proposons pour désigner l'ensemble des propositions répondant à un problème proche du nôtre dans un formalisme commun. Enfin, nous appuyant sur cette modélisation, nous classons et évaluons ces différentes propositions en deux grandes catégories : les approches guidées par la diffusion et celles guidées par les calculs, chacune de ces grandes catégories étant l'objet d'une section spécifique.

2.1 Critères d'analyse

Cette section présente les critères structurant notre évaluation. Nous regroupons ceux-ci en deux catégories : d'abord, les éléments de classification, puis les éléments d'évaluation. Les premiers, en effet, nous semblent mettre en lumière des proximités et des différences marquées entre les modes de fonctionnement des différentes propositions, et donc de former plusieurs familles d'approches. Les seconds sont liés aux choix d'organisation effectués par les auteur·e·s de ces propositions, en fonction des aspects spécifiques du problème étudié. Ils ne sont donc pas constitutifs de liens de parenté entre les différentes approches, mais montrent plutôt comment les différentes familles peuvent se décliner en fonction du contexte étudié et des objectifs à atteindre.

2.1.1 Éléments de classification

Notre problème général étant le traitement de requêtes sur des flux, les possibilités admises concernant l'expression et la prise en compte de ces requêtes forment naturellement un critère essentiel sur lequel nous appuyer pour examiner l'état de l'art. Dans le cadre

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

d'un système ouvert, dans lequel de nombreux·es utilisateurs·trices sont disposé·e·s à participer au bon fonctionnement général, la façon dont le système tire profit des ressources à sa disposition est un second critère d'importance similaire. Nous abordons ici successivement ces deux points en présentant, pour chacun d'entre eux, les éléments de contexte à prendre en compte dans leur étude.

Généricité vis-à-vis du langage de requêtes

Les systèmes de gestion de base de données relationnels classiques (SGBDr) considèrent généralement deux aspects complémentaires : ceux ayant trait aux données, et ceux ayant trait aux requêtes.

L'organisation des données vise à créer les conditions nécessaires à faciliter l'accès à celles-ci. Pour cela, à la dimension physique, qui correspond à la façon dont le SGBDr dispose en pratique les données sur le disque et gère les ajouts et suppression, s'ajoute une dimension logique, correspondant pour sa part à la structure théorique envisagée pour ces données (tables, index…).

Les requêtes, pour leur part, sont compilées en un plan d'exécution puis opérées pour obtenir et renvoyer les résultats. Ce plan d'exécution tient compte des données à disposition et de la façon dont elles sont organisées ; de même que des possibilités d'optimisation dues au traitement coordonné d'un grand nombre de requêtes. Là encore, il est possible de distinguer une phase logique, correspondant à l'analyse de la requête et à la mise en place du plan d'exécution, se basant sur l'algèbre relationnelle, et une phase physique, correspondant à l'application des opérateurs et à la mise à disposition des résultats.

Organisations des données et des requêtes correspondent donc aux deux faces d'une même pièce, et sont utilisées de manière complémentaire. Ainsi, le langage SQL (« *Structured Query Language* »), utilisé par les principaux SGBDr, contient à la fois[45] des instructions servant à définir l'organisation logique des données (LDD, pour « Langage de Définition de Données »), et des instructions servant à spécifier précisément les données auxquelles on peut vouloir accéder ou que l'on peut vouloir modifier (LMD, pour « Langage de Manipulation de Données »).

Bien sûr, une telle dimension logique suppose d'avoir été spécifiquement mise en place : pour l'analyse de masses d'informations de sources diverses, comme on peut la pratiquer en fouille de données (*data mining*), il est généralement délicat, voire impossible, de considérer *a priori* une telle structuration. Dans le domaine qui nous intéresse, en revanche, il est possible de considérer une certaine forme d'organisation logique : les flux d'information sont généralement au minimum *semi-structurés*, c'est-à-dire qu'ils contiennent des marqueurs permettant une analyse hiérarchique de leur contenu.

Cette caractéristique suffit à envisager la possibilité d'exprimer des requêtes, plus ou moins expressives, pour extraire l'information des flux. Dans les systèmes Pub/Sub, par exemple, les organisations dites *topic-based* sont celles dans lesquelles les sources primaires de données (« *publishers* ») spécifient différents sujets dans lesquels s'inscriront leurs publications. Les utilisateurs·trices (« *subscribers* ») peuvent ainsi choisir les sujets qui les intéressent, et auxquels iels sont donc inscrits, ce qui forme plusieurs canaux de diffusion distincts. Cette spécification des sujets est une forme de structuration minimale[1], suffisante pour exprimer une certaine forme de requêtes, qui peut cependant s'avérer d'une expressivité limitée.

[1] Ce qui, cependant, n'empêche bien sûr pas que les données diffusées à travers ces canaux soient davantage structurées ; mais cette possible structuration n'a pas besoin d'être connue pour assurer une organisation de ce type.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Dans les organisations Pub/Sub dites *content-based*, en revanche, les souscriptions des utilisateurs·trices permettent une analyse spécifique des contenus, suivant des critères précis, qui possèdent donc une expressivité bien plus importante, mais nécessitent que les informations contenues dans les flux correspondent à une structure mieux spécifiée pour permettre cette analyse. Nous pouvons au passage noter que, contrairement à ce que leur nom pourrait sembler indiquer, les organisations Pub/Sub « basées sur le contenu » sont orientées sur le traitement des requêtes, et non sur l'organisation des données.

La possibilité d'exprimer des requêtes sur des flux suppose l'existence de *langages de requête*, permettant aux utilisateurs·trices d'exprimer les résultats attendus d'une manière qui sera comprise par le système. Étant le langage de requêtes le plus utilisé dans le domaine des SGBDr classiques, SQL a été exporté à d'autres domaines, et parfois augmenté pour prendre en compte certaines caractéristiques ne correspondant pas à son environnement d'utilisation d'origine. Ainsi, l'université de Stanford en a proposé une extension baptisée *Continuous Query Language*, ou CQL[8], apportant notamment la gestion des fenêtres temporelles.

Toutefois, ce langage n'est pas le seul à pouvoir s'appliquer pour les données relationnelles classiques. Ainsi, Datalog[15], basé sur le langage de programmation Prolog, permet d'exprimer des requêtes sous une forme déductive. Pour d'autres façons de structurer les données, il existe par ailleurs des langages de requêtes spécifiques : aux documents XML, par exemple, sont associés des langages d'interrogations tels que XQuery[11] et XPath[25]. Pour ce qui concerne le Web sémantique, le W3C propose SPARQL[81], un langage d'interrogation et de modification basé sur RDF.

L'existence de ces multiples langages, souvent liés à un format de données particulier, entraîne d'évidentes problématiques de généricité pour l'approche proposée. En effet, si le calcul de l'organisation dépend de propriétés avancées spécifiques à un langage donné, appliquer cette approche à l'utilisation d'un autre langage ne pourra se faire, lorsqu'il est possible, qu'au prix d'importantes modifications. Le langage étant dépendant de la forme des données, cela signifie qu'une approche de ce type aura des domaines d'application fortement restreints.

Étant donné l'importance de cette limitation, nous avons choisi d'utiliser ce critère comme élément principal de catégorisation, déterminant ainsi l'organisation générale de notre étude bibliographique. Nous avons déterminé cinq catégories distinctes, réunies en deux grandes familles.

En premier lieu, nous considérons les approches conçues pour être totalement indépendantes du langage de requêtes utilisé. Cette totale indépendance ne peut se concevoir qu'en excluant le calcul des requêtes de l'organisation du système : celle-ci porte uniquement sur les aspects de diffusion, avec pour tâche de permettre à chaque requêteur d'obtenir les flux initiaux tels qu'ils sont produits par les sources. Les traitements désirés sont ensuite effectués par chaque participant, en dehors du système. Toute autre manière d'aborder les choses nécessite de prendre en compte certaines informations sur la façon dont les calculs vont se faire, ce qui entraîne une dépendance vis-à-vis de certains aspects du langage.

Toutefois, cette relation au langage peut se limiter à des aspects externes plutôt qu'à la façon dont les traitements sont effectués. Ainsi, la seconde catégorie d'approches repose sur la possibilité de déterminer les requêtes qui renverront des résultats identiques, et de partitionner l'ensemble des participants en fonction de ces différentes *classes d'équivalences*. Déterminer quelle requête est équivalente à quelle autre nécessite bien évidemment un certain degré de connaissance du langage, mais on peut envisager cette fonction comme une propriété fournie par le langage lui-même : une telle approche sera alors suffisamment

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

générique pour être mise en place sans modification sur tout langage disposant de la possibilité d'évaluer l'équivalence ou non entre deux requêtes, indépendamment de la façon dont ce test est effectué en pratique.

Dans un cas comme dans l'autre, les calculs sont donc relégués à la périphérie du système, et l'organisation des différents participants ne se fait (après partitionnement dans le second cas) qu'en tenant compte des aspects liés à la diffusion. L'organisation peut alors être déchargée de certaines contraintes : tous les participants (au sein du système global ou de chacune de ses parties) doivent recevoir les mêmes informations, ce qui diminue le nombre de différenciations à prendre en compte. Nous avons donc choisi de regrouper ces deux catégories dans la famille des approches « guidées par la diffusion », présentée en section 2.3. Pour les autres approches, ces contraintes sont au contraire majeures, et les liens entre participants ne peuvent être faits qu'en s'assurant que les flux échangés (pouvant différer des flux initiaux) permettent bien le calcul des requêtes considérées. Elles forment donc une famille d'approches « guidées par les calculs », qui sont l'objet de la section 2.4.

Dans cette section, les trois catégories d'approches sont spécifiques à certains aspects du langage. Comme pour la fonction de vérification d'équivalence, ces aspects peuvent cependant être accessibles via une API (« interface de programmation applicative », un ensemble de fonctions servant à interagir avec un système informatique indépendamment de la façon dont il est mis en œuvre) que plusieurs langages sont en mesure de fournir. Les approches concernées pourront alors être déployées sur tout langage fournissant cette API, et conserver ainsi une certaine forme de généricité.

C'est ce point qui distingue les trois catégories que nous proposons au sein de cette famille : deux des catégories considérées reposent sur de telles API, l'une portant sur l'extraction de la séquence d'opérateurs utilisés par les requêtes, l'autre portant sur un mécanisme de réécritures, qui sont toutes deux explicitées dans la section concernée. La catégorie restante correspond aux systèmes qui, pour leur part, dépendent d'aspects très spécifiques à un langage en particulier, et présentent donc un niveau de généricité assez faible.

Ressources utilisées

Les réseaux informatiques de première génération étaient conçus de manière essentiellement centralisée : un serveur principal gérait l'ensemble du fonctionnement ; quand les autres machines avaient essentiellement une fonction de terminaux passifs. Ce mode d'organisation présentant des inconvénients assez notables, les réseaux plus récents reposent sur davantage de répartition. Ainsi, Internet lui-même repose sur un fonctionnement entièrement dénué de serveur central, dans lequel toute machine peut théoriquement[2] être joignable presque directement, les différents serveurs relais étant conçus pour être transparents et interchangés.

La philosophie de conception d'Internet[24] repose entièrement sur le principe de distribuer le fonctionnement, et préconise d'éviter autant que possible la centralisation. Pour autant, il s'agit-là du fonctionnement du réseau de base, et les applications construites sur ce réseau peuvent pour leur part être plus ou moins centralisées. En effet, les approches entièrement centralisées, malgré les problématiques évoquées au chapitre précédent, apportent certains avantages. En particulier, elles sont généralement plus simples à mettre en place et à maintenir, et permettent certaines optimisations des calculs qui sont difficiles

[2] Ce principe a cependant été remis en cause par la mise en place de NAT[41], qui causent des difficultés de déploiement pour les réseaux pairs-à-pairs.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

à répartir. Pour autant, les nécessités de passage à l'échelle entraînent souvent le besoin de mobiliser davantage de ressources, par exemple sur un parc de machines dédié. On parle alors de « systèmes répartis », quoique l'anglicisme « systèmes distribués » soit plus courant et puisse également être trouvé au sein de ce document.

De la même manière qu'elles forment deux aspects complémentaires vis-à-vis du langage de requêtes, données et requêtes peuvent être envisagées de manière complémentaire vis-à-vis de la répartition. Dès lors que la diffusion est partiellement distribuée, certaines techniques sont mises en place pour faciliter leurs transferts. La mise en cache[23] des informations est ainsi très utile dans les systèmes où certaines données devront être récupérées à plusieurs reprises, comme notamment les CDN[88]. Évidemment, son intérêt est beaucoup plus grand en ce qui concerne les données statiques que pour la diffusion de flux continus, où il est impossible de mettre en cache les tuples non-encore produits. Cependant, les techniques de mise en cache peuvent potentiellement s'avérer utiles si certains participants doivent maintenir l'état des flux quelque temps, par exemple pour réaliser des requêtes agrégatives portant sur une large fenêtre d'informations, voire pour rediffuser les informations à d'autres participants ayant subi des interruptions réseau.

Dans les cas dans lesquels il est nécessaire de stocker des informations sur plusieurs machines, il est possible d'organiser le réseau de telle sorte que les données soient regroupées par thématiques, afin de faciliter certaines recherches. C'est le concept général des *Semantic Overlay Networks*[29], ou « SON », dont le principe est d'abord de proposer une classification arborescente des données (un SON dédié, par exemple, à la musique pourra ainsi contenir une catégorie « Rock », contenant elle-même une sous-catégorie « Grunge »). Les différentes catégories sont ensuite confiées à des machines différentes, par une répartition tenant compte des intérêts des participants concernés.

Lorsqu'une telle catégorisation des données ne s'avère pas pertinente, les tables de hashage distribuées (ou DHT, acronyme du terme anglais « *Distributed HashTable* »), telles que Pastry[83] ou Chord[87], permettent de répartir un ensemble de combinaisons clefs/valeurs dans un réseau de participants n'étant pas nécessairement différenciés par leurs intérêts. Une DHT accorde notamment une importance particulière à la pérennité des données, chaque partie de l'annuaire complet étant distribué sur plusieurs machines. Les participants désirant obtenir une information interrogent l'une des machines responsables, qui sera capable, si elle ne dispose pas des bonnes informations, de désigner une autre machine possédant des informations plus précises. Les DHT sont habituellement utilisées dans divers domaines tels que les systèmes Pub/Sub, les réseaux d'échange pair-à-pair ponctuels de génération récente[57], ou encore d'autres sortes de réseaux collaboratifs, comme le moteur de recherche YaCy[51].

Mais les réseaux collaboratifs peuvent également avoir d'autres objectifs que celui de stocker de l'information. Ainsi, BOINC[7] est un outil distribué permettant aux personnes intéressées de fournir une partie des ressources de traitement de leurs machines à la recherche scientifique, en fonction de leurs intérêts (l'utilisateur·trice choisit les projets auxquels il lui semble important de participer). De tels réseaux de calcul entièrement basés sur les utilisateurs·trices peuvent s'avérer très intéressants en termes de puissance disponible : comme le précise l'étude[7], ce mode de fonctionnement, reposant sur le volontariat, offre des capacités de calcul s'avérant comparables, sinon supérieures, à celles des supercalculateurs conventionnels.

Dans le cadre général du traitement réparti, les réseaux de clusters, permettant une parallélisation importante, sont un aspect assez étudié, notamment pour ce qui concerne l'allocation de ressources dans une organisation en grille[39, 44]. MapReduce[30], proposé par Google en s'inspirant de principes issus de la programmation fonctionnelle, permet

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

de mettre en place aisément un tel système par la définition de deux fonctions principales (*map*, pour générer une série intermédiaire à partir des données initiales, et *reduce*, pour combiner ces valeurs intermédiaires). Un certain nombre des propositions de calcul distribué présentées ci-après reposent sur ce principe, certaines d'entre elles utilisant Hadoop[86], un *framework* Java sous licence libre implémentant ce concept.

En contrepartie, ce fonctionnement réparti est plus délicat à mettre en place, et nécessite donc davantage de travaux. Les réseaux d'échanges pair-à-pair ponctuels sont ici un bon exemple illustratif : bien que leur fonctionnement de base soit distribué, l'organisation du système se faisait, dans les premiers temps, de manière centralisée, les évolutions successives[57] ayant amené de plus en plus de distribution.

Ainsi, Napster[13], réseau d'échange pair-à-pair de première génération, reposait sur un *tracker* centralisé qui, s'il n'intervenait pas dans les échanges, était nécessaire pour mettre les participants en relation les uns avec les autres. En conséquence, la disparition du *tracker* a suffi à amener la fin du réseau, aucun nouvel échange ne devenant possible.

Une génération intermédiaire de réseaux d'échange pair-à-pair, comprenant notamment FastTrack[60], s'est dotée du concept de *super-pairs*, des nœuds particuliers chargés de guider l'organisation. Les réseaux à super-pairs distinguent donc deux classes de participants, avec la nécessité que les super-pairs doivent disposer d'une puissance suffisante et d'une bonne connexion.

Dans les générations plus récentes, dont fait notamment partie BitTorrent[26], un fonctionnement basé par exemple sur l'usage de DHT permet de maintenir une structure solide sans requérir de certains nœuds qu'ils soient en mesure de prendre en charge une part essentielle de l'organisation.

Un système entièrement géré par ses participants doit cependant présenter quelques caractéristiques particulières pour pouvoir fonctionner sur le long terme. Il doit, notamment, être capable de s'organiser de lui-même (*self-organizaion*), de se configurer lui-même (*self-configuration*), et de se réparer lui-même en cas de panne (*self-healing*). *A semantic overlay for self-* peer-to-peer publish/subscribe*[6] propose une définition de ces trois concepts, ainsi qu'une comparaison entre deux modes d'organisation possibles, l'un épidémique, et l'autre dans lequel un nœud particulier dirige les opérations. Les résultats obtenus montrent que, quoique l'approche épidémique conduise chaque nœud à recevoir davantage de messages de coordination, ce mode s'avère globalement distribuer plus aisément la charge.

La répartition du système présente donc deux aspects essentiels devant être étudiés conjointement : la répartition du fonctionnement, et celle de l'organisation. Concernant le premier aspect, nous distinguons trois niveaux. Le premier est celui des systèmes centralisés, dans lesquels une seule machine est mobilisée pour gérer l'ensemble du fonctionnement. Le second niveau correspond aux systèmes répartis sur un parc de machine dédiés à ce but : le nombre de ressources disponibles est donc plus important, mais toutes ces ressources restent gérées par la même entité. Le troisième niveau correspond au cas où les ressources des utilisateurs·trices du système sont mobilisées afin de permettre le fonctionnement de celui-ci.

Concernant la distribution de l'organisation, la classification proposée par Stéphane Bortzmeyer[12] présente quatre classes distinctes, basées sur la façon dont peut être modifié l'état du système. Dans les systèmes de classe 1, une seule entité est en charge de prendre les décisions, le reste du réseau n'ayant qu'à s'y plier. Cela concerne donc typiquement tous les systèmes dont le fonctionnement est géré par une entité unique, mais également les systèmes dotés d'un *tracker* pour gérer la mise en relation des participants

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

(Napster[13] est ainsi mentionné comme exemple de cette catégorie). Les systèmes de classe 2 sont ceux pour lesquels l'organisation est hiérarchique : une autorité principale décide du fonctionnement global du système, mais d'autres entités, qui lui sont subordonnées, peuvent prendre des décisions vis-à-vis de la partie du système qu'elles contrôlent. C'est, typiquement, le mode de fonctionnement des noms de domaines (« DNS »)[3]. Dans les systèmes de classe 3, l'organisation est le fruit de décisions mobilisant un nombre important de participants, sans qu'aucun ne joue de rôle prépondérant. Les systèmes de classe 4, enfin, sont ceux pour lesquels aucune autorité ne contraint l'organisation du système : chaque participant peut contacter directement les autres sans coordination globale.

<table>
<tr><th>Ressources support</th><th>Autorité d'organisation</th></tr>
<tr><td>1 site, 1 contrôleur</td><td rowspan="2">Classe 1 (unique)</td></tr>
<tr><td>n sites, 1 contrôleur</td></tr>
<tr><td rowspan="3">n sites, m contrôleurs</td><td>Classe 2 (hiérarchique)</td></tr>
<tr><td>Classe 3 (majoritaire)</td></tr>
<tr><td>Classe 4 (individuelle)</td></tr>
</table>

Tableau 2.1: Classification de la répartition des systèmes

Comme le montre le tableau 2.1, ces deux classifications sont complémentaires, et permettent ainsi d'étudier en détail le niveau de répartition d'une proposition donnée. Comme le souligne Stéphane Bortzmeyer[12], cette classification est destinée à décrire le fonctionnement des systèmes de manière objective, et n'inclut pas de jugement de valeur (la classe 4 n'est pas « meilleure » que la classe 3 dans l'absolu). Un fonctionnement centralisé pourra ainsi être préféré à un fonctionnement réparti lorsque ceci sera plus adapté au contexte. Toutefois, les réserves exprimées dans le chapitre précédent concernant les systèmes de classe 1 (nécessité d'une connaissance exhaustive, et généralement exclusive, des requêtes exprimées, ce qui entraîne notamment quelques vulnérabilités) nous font considérer qu'une approche permettant un certain degré de liberté vis-à-vis de sa répartition sera généralement préférable.

Au sein des différentes familles d'approches constituées selon le critère précédent, le niveau de répartition atteint par les différentes approches permet d'affiner leur classement. Ce niveau de répartition guide donc l'organisation interne des cinq sections présentant notre classification.

2.1.2 Éléments d'évaluation

Si les critères envisagés précédemment permettent de classifier efficacement l'approche, ils ne suffisent naturellement pas à en évaluer tous les aspects. Il convient également de prendre en compte un certain nombre d'autres éléments qui ne sont pas constitutifs de liens entre les différentes approches, mais reflètent les préoccupations envisagées dans leur mise en place. Deux points de vue sont ici prépondérants : d'une part, celui des utilisateurs·trices du système, relatif aux conditions dans lesquels il leur sera possible d'obtenir les résultats attendus ; d'autre part, celui de l'entité en charge de mettre le système en place et d'en maintenir le fonctionnement, relatif aux difficultés d'organisation. Sur de nombreux aspects, les deux points de vue sont fortement liés, et généralement en

[3] Il convient de noter que les participants considérés, pour le DNS, sont les différents serveurs de nom. Les requêtes exprimées par les et auprès des résolveurs ne modifient en effet pas l'état du système.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

opposition : répondre aux attentes particulières des utilisateurs·trices demande de prendre en compte des contraintes supplémentaires, qui peuvent complexifier l'organisation.

Un premier aspect concerne les **coûts de fonctionnement** du système. Quel que soit le mode d'organisation, il est nécessaire de limiter ceux-ci, afin de garantir le passage à l'échelle : dans les systèmes dont le fonctionnement est pris en charge par une unique entité, qu'elle dispose ou non de plusieurs ressources pour ce faire, il est nécessaire que l'ensemble de ces ressources suffise à servir un nombre d'utilisateurs·trices pouvant s'avérer potentiellement très important. Dans le cas où ces utilisateurs·trices contribuent au bon fonctionnement du système, leurs ressources doivent être utilisées de manière raisonnée. En effet, celles-ci sont généralement plus limitées que celles d'un parc de machines dédié. Même si un·e utilisateur·trice donné·e acceptait de voir ses capacités débordées, cela induirait des difficultés de fonctionnement faisant que cette situation est à éviter.

Dans le cadre d'un système de traitement de requêtes sur des flux, deux aspects particuliers du fonctionnement sont à étudier : d'une part, les coûts de calculs, relatifs au traitement de ces requêtes, et dépendant de la complexité de ces requêtes autant que de la volumétrie des flux utilisés, d'autre part, les coûts de diffusion, relatifs à la volumétrie des flux à partager entre les différents participants et au nombre de destinataires auxquels envoyer ces flux. Tous deux doivent être diminués autant que possible. Cette diminution peut être envisagée de deux manières, équivalentes du point de vue global, mais pouvant présenter une grande différence pour les participants concernés : soit répartir une charge faible sur l'ensemble du système, soit demander à un nombre réduit de participants de supporter une charge plus importante (sans, bien sûr, dépasser leurs limites de capacités), pour autoriser les autres participants à n'être pas sollicités du tout.

En contrepartie de ces coûts de fonctionnement doivent être étudiés les **coûts d'organisation** du système. En effet, cette organisation peut être coûteuse à calculer : les multiples éléments à considérer peuvent induire des difficultés à déterminer la structure la plus intéressante pour le système. De plus, la mise en place de cette structure, une fois celle-ci calculée, peut également être coûteuse, du fait de la nécessité de mettre en relation les différentes ressources par le réseau. Ces difficultés font qu'une réduction optimale des coûts de fonctionnement peut parfois être laissée de côté lorsqu'un fonctionnement de coût raisonnablement plus élevé est plus simple à déterminer et à mettre en place.

La **latence**, c'est-à-dire le délai requis entre l'émission des données d'origine par les sources primaires et l'obtention des résultats désirés, est également un aspect complémentaire aux coûts de fonctionnement. En effet, surcharger les capacités des différents participants entraîne[53] un ralentissement du système. Une répartition sollicitant certains participants au maximum de leurs capacités peut cependant permettre de rapprocher autant que possible d'autres participants des sources, et ainsi diminuer la latence globale du système. Les utilisateurs·trices étant généralement intéressé·e·s par le fait d'avoir une latence aussi faible que possible, un équilibre peut ici être trouvé, le cas échéant, concernant leur participation active au système : utiliser les capacités les plus élevées pour diminuer la latence globale peut inciter ces participants à offrir davantage de ressources au système.

L'intérêt d'une organisation donnée se mesure également à la façon dont certaines situations caractéristiques sont prises en compte, et donc à la façon dont le système *s'adapte* à ces situations. L'**adaptabilité** d'un système vis-à-vis d'un aspect donné représente son aptitude à évoluer en fonction de cet aspect pour obtenir un état valide qui convient à l'ensemble des participants. L'adaptabilité n'est donc pas une notion absolue et indépendante, mais qui se décline en fonction d'un critère donné. Notre analyse des différentes situations et propositions nous a permis de mettre en lumière quatre critères d'adaptabilité.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Le premier d'entre eux est la *connexité* des requêtes, c'est-à-dire le fait que les requêtes exprimées soient plus ou moins en lien avec les autres. Dans un système intégrant les calculs, une connexité importante entre les requêtes exprimées apporte souvent de bonnes opportunités, dans la mesure où c'est elle qui permet le partage des calculs communs. En contrepartie, plus cette connexité est forte, plus le nombre de relations possibles à évaluer est élevé, complexifiant d'autant la tâche d'organisation du système.

La *popularité* des requêtes représente pour sa part le fait que certaines requêtes seront exprimées par un nombre plus ou moins grand de participants différents, soit sous une forme identique, soit, lorsque le langage le permet, par l'usage de plusieurs expressions synonymes. La popularité peut être perçue comme un cas particulier de la connexité. Il s'agit toutefois d'un cas très particulier, entraînant des problématiques spécifiques.

Pour ces deux cas, l'évolution attendue d'un système adaptable correspond au fait de limiter autant que possible les traitements redondants (cette redondance portant sur l'ensemble de la requête dans le cas de la popularité). L'adaptabilité d'un système à la connexité et à la popularité s'évalue par la facilité avec laquelle ces traitements redondants sont identifiés et évités lors du calcul de l'organisation.

Au cours de la vie du système, la *volumétrie des flux* issus des sources de données peut varier de manière importante, ce qui peut entraîner des changements dans les coûts de traitement effectif des requêtes et de diffusion des résultats. Une approche adaptable à cet aspect est capable de réévaluer les relations entre les participants au sein du système, afin, *a minima*, de garantir qu'une augmentation de volumétrie ne causera aucun dépassement de capacités, et si possible d'améliorer le confort global (particulièrement vis-à-vis de la latence) lors d'une diminution. Une telle adaptabilité présente cependant un risque très important vis-à-vis de la stabilité du système, et donc des coûts d'organisation.

Enfin, les *capacités des participants* peuvent également varier, soit par l'arrivée ou le départ de participants, qui entraîne donc également une variation des besoins correspondants, soit par des changements de capacités au niveau de participants déjà intégrés. Une approche adaptable à ce niveau est une approche qui s'assure que les ressources sont dûment réparties pour éviter la surcharge si les capacités diminuent davantage que les besoins, et si possible qu'une augmentation des capacités supérieure à l'augmentation des besoins entraîne une meilleure répartition des charges et un gain au niveau de la latence globale du système.

D'une manière générale, l'adaptabilité d'un système vis-à-vis de l'un de ces aspects se mesure donc à la capacité du système à évoluer, au coût correspondant à ces évolutions, et au niveau de satisfaction final des participants. Il convient toutefois de prendre en compte le fait que ces différents aspects ne sont pas isolés les uns des autres. Utiliser la connexité des requêtes pour guider les relations entre participants, par exemple, contraint généralement la façon dont les variations de capacités pourront être utilisées. Une évaluation complète du système doit donc prendre en compte les différentes configurations possibles.

Enfin, un aspect qui nous semble particulièrement important est la **prise en compte des intérêts**. Il ne s'agit ici pas seulement de la façon de gérer la popularité, ou plus généralement la connexité, donc le fait que certains participants ont des intérêts communs ; mais également d'étudier quelle priorité est donnée au fait de respecter ces intérêts. Dès lors que les utilisateurs·trices sont invité·e·s à contribuer au bon fonctionnement du système, on peut en effet soit leur demander de ne travailler qu'au calcul de leurs propres requêtes, soit s'autoriser à leur confier des tâches arbitraires, sans rapport avec ce qu'ils ont exprimé.

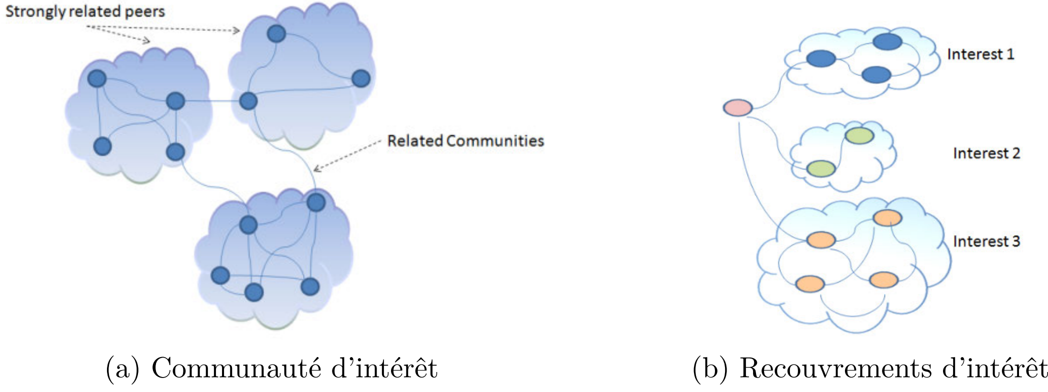
Cette dernière situation peut présenter, d'un certain point de vue, l'avantage de fa-

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

voriser le respect de l'intimité numérique : une analyse extérieure du système ne permettra alors pas nécessairement de déterminer quelles requêtes ont été exprimées par ces participants, ce qui ne pourra pas être empêché dans l'autre cas. Cependant, dans tous les systèmes de classe 1, cet intérêt est fortement limité par le fait qu'il ne protège pas de l'autorité en charge de l'organisation, qui est généralement perçu comme la menace principale envers cette intimité.

Même dans les cas où les intérêts sont respectés, le concept de *communautés*, intuitivement présenté dans le chapitre précédent et qui témoigne d'une forte prise en compte de cet aspect, semble pour l'instant très peu étudié dans le domaine des systèmes de requêtage sur des flux auquel nous nous intéressons (même si certaines propositions présentent des structures qui s'en rapprochent quelque peu). D'autres domaines comportant des échanges entre participants, en revanche, contiennent des concepts analogues.

En particulier, les systèmes de recommandations en pair-à-pair basés sur le bavardage (ou *gossip*), y ont aisément recours. À titre d'exemple de ce domaine, Prego[67] envisage les relations entre intérêts comme un graphe dont les composantes fortement connexes (liées à la notion de popularité dans notre contexte) forment des communautés. Comme le montre la figure 2.1 (a), ces communautés peuvent être reliées entre elles par la présence de relations d'intérêts moins marquées (correspondant à notre notion de connexité) avec les participants d'autres communautés. L'image (b) illustre le fait qu'un même participant exprimant différents intérêts peut se voir rejoindre différentes communautés, chacune correspondant à l'un des intérêts qu'il exprime. Transposé dans un système de requêtage, cela correspondrait à un participant exprimant plusieurs requêtes, et rejoignant une communauté différente pour chacune d'elles.



(a) Communauté d'intérêt

(b) Recouvrements d'intérêt

Figure 2.1: Communautés dans Prego (repris de [67])

Notons au passage que d'autres sortes de parallèles peuvent être fait entre le domaine des traitements de requêtes et celui de la recommandation par bavardages. Ainsi, Rappel[76] est un système Pub/Sub particulier, dans lequel la mise en relation des participants se fait par une recherche de similarités, suivant les principes de construction de voisinage. Un aspect intéressant est que cette approche se focalise en priorité sur le respect des intérêts des utilisateurs, les auteurs montrant que la perte d'informations due au manque de pertinence du voisinage utilisé n'ont lieu que durant une phase d'initialisation extrêmement réduite dans le temps.

Un tel mode d'organisation présente cependant, à nous yeux, l'inconvénient de nécessiter une remise en cause perpétuelle des relations de voisinage. Dans notre contexte, les souscriptions des utilisateurs·trices présentent des similarités plus simples à mettre en lu-

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

mière que les profils complexes habituellement étudiées dans ce domaine. C'est pourquoi nous préférons envisager l'analyse des relations mathématiques entre les requêtes, stables dans le temps, pour constituer notre modélisation.

2.2 Modélisation

Quoiqu'elles présentent des aspects particulièrement différents, les propositions visant à répondre au problème d'échange continu d'informations reposent pour la plupart sur les mêmes briques de base, que sont les flux de données, les requêtes exprimées sur ces flux et les participants exprimant ces requêtes et générant ces flux. Les définitions précises proposées pour ces briques peuvent varier d'une approche à l'autre, mais une modélisation suffisamment générique permet de les mettre en perspective de manière plus efficace et de comparer leurs différentes stratégies à travers un prisme commun. Nous proposons donc un modèle permettant de décrire les différentes propositions existantes, et de les classer en différentes catégories, dans un formalisme que nous utilisons ensuite pour spécifier le mode de construction que nous proposons.

Dans cette section sont d'abord décrits les différents composants de base utilisés pour notre modélisation, puis la façon dont ces composants sont utilisés pour étudier et représenter le système, tant pour ses aspects physiques (relations entre participants) que logiques (relations entre requêtes). Enfin, nous nous intéressons à quelques éléments structurels que cette modélisation permet de mettre en lumière, et qui sont retrouvés dans plusieurs des approches présentées par la suite.

L'ensemble des notations introduites dans cette section, qui sont utilisées tout au long de ce document, est résumé dans le tableau 2.2 présenté en fin de section.

2.2.1 Composants de base d'un système générique

La notion primordiale, pour définir un système répondant au problème que nous considérons, est celle de flux. *An Algebric Window Model for Data Stream Management*[77] en propose la définition suivante :

Définition 1 (Flux de données)**.**
Un flux est un ensemble possiblement infini de tuples correspondant à un schéma commun, contenant au minimum deux attributs spéciaux : un marqueur de temps et un identificateur physique.

Cette définition, suffisamment générique pour s'appliquer à de nombreux cas[4], met en avant deux éléments essentiels : d'une part, la présence d'indicateurs de temps, permettant de gérer les productions successives de nouvelles informations ; d'autre part, l'existence d'une structuration commune aux informations publiées, condition nécessaire à la mise en place de traitements automatisés du flux. Un identifiant unique pour chacun des éléments produits permet par ailleurs de gérer plus aisément les éventuelles duplications dues à la présence de ces tuples dans plusieurs flux intermédiaires.

Notant F l'ensemble des flux présents, nous considérons désormais la façon dont ces flux intermédiaires seront produits à partir des flux initiaux. En d'autres termes, nous nous intéressons à la façon dont les utilisateurs vont *transformer* ces flux :

[4] Elle pourrait s'avérer problématique pour, par exemple, un flux vidéo sans interruption, dans lequel il peut sembler difficile de découper l'information en une séquence de tuples. Il s'agit toutefois d'une problématique d'implémentation, les principes généraux exposés ici restant les mêmes dans ce cas.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Définition 2 (Transformation et requête)**.**

La transformation d'un ou de plusieurs flux consiste en la création, à partir de ces flux d'entrées, d'un flux de sortie, dont les éléments, s'ils peuvent présenter un schéma différent de ceux des flux initiaux, sont constitués à partir de ceux-ci. Mathématiquement parlant, une transformation est donc une fonction prenant en paramètre un ensemble de flux, et renvoyant un flux unique :

$$t : \mathscr{P}(F) \rightarrow F$$

Nous notons T l'ensemble des transformations du système.

Une requête $r \in R$ est l'expression d'une transformation particulière, définie à l'aide d'un langage algébrique sur un ensemble de flux $S \subseteq F$ préalablement connu.

Nous nous autorisons à confondre la requête algébrique avec la transformation correspondante, et donc à considérer que $R \subseteq T$. En effet, le but est ici d'avoir un terme générique désignant le fait de récupérer un nombre variable de flux d'entrée et d'émettre un unique flux de sortie, afin de pouvoir décrire le système de manière cohérente, tant au niveau des relations théoriques que des opérations effectuées en pratique.

Nous focalisant ici sur les échanges entre flux et non sur la manière dont les transformations sont réalisées en pratique, nous n'avons pas besoin du détail du fonctionnement interne, et considérons des transformations essentiellement leurs entrées et leurs sorties.

Étant donné une transformation $t \in T$, nous notons t_{in} l'ensemble des flux d'entrée qui sont utilisés, et $t(t_{in}) = t_{out}$ le flux de sortie. Chaque transformation, appliquée sur ses entrées régulières, produisant un unique flux de sortie, nous nous autorisons à confondre par abus t et t_{out} chaque fois que cela n'entraîne aucune ambiguïté. Par ailleurs, nous décrivons la production des flux initiaux du système comme le fruit de transformations particulières $s \in S \subseteq T$ telles que $s_{in} = \emptyset$.

Notons que la présence d'identifiants et indicateurs de temps dans les tuples permet de considérer que deux flux sont égaux dès lors qu'ils contiennent les mêmes éléments, sans considérer l'ordre dans lequel ils sont reçus. Par ailleurs, l'analyse du système nécessite de définir une notion d'équivalence entre transformations :

Définition 3 (Équivalence entre transformations)**.**

Deux transformations sont considérées comme équivalentes si, appliquées dans un environnement présentant les mêmes flux initiaux, les flux qu'elles produisent à partir de leurs entrées respectives contiennent les mêmes éléments :

$$\forall t, t' \in T, t \equiv t' \Leftrightarrow \forall S, t_{out} = t'_{out}$$

Ainsi, une transformation source, produisant l'un des flux initiaux du système, est strictement équivalente à toute requête identité exprimée sur ce flux initial :

$$s : \emptyset \mapsto s_{out}, r : \{s_{out}\} \mapsto s_{out}, r \equiv s$$

Ces définitions posées, nous pouvons désormais caractériser les participants au système, en fonction des flux qu'ils émettent et des transformations qu'ils effectuent :

Définition 4 (Participant)**.**

Un participant au système est une entité susceptible d'acquérir et d'émettre plusieurs flux d'information dans le système. À chaque participant p correspond un triplet $\langle p_s, p_r, p_t \rangle$ pour lequel :

- *p_s décrit l'ensemble des flux originaux qui seront produits par ce participant,*

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

- p_r *est l'ensemble des requêtes qu'exprime ce participant, avec,* $\forall r \in p_r, r_{in} \subseteq S$*,*
- p_t *est l'ensemble des transformations qui sont confiées à ce participant.*

Nous considérons, pour harmoniser les définitions, que tout participant du système est capable d'émettre le flux de sortie de chacune de ses transformations, même dans le cas où le système est conçu de telle sorte que cette possibilité ne sera jamais sollicitée pour certains d'entre eux. Notant P l'ensemble des participants du système, nous pouvons préciser que tout flux et toute transformation du système sont associés à un participant : $S = \bigcup p_s, p \in P$, $R = \bigcup p_r, p \in P$ et $T = \bigcup p_t, p \in P$. Par ailleurs, soulignons que l'ensemble S des flux sources utilisables par le langage de requêtes utilisé correspond à l'ensemble des flux sources du système : une requête ne peut donc être exprimée que si les participants produisant les flux concernés sont connus dans le système.

Les sources de données classiques, produisant un ensemble de flux de base sans émettre de requête, sont donc des participants tels que $p_s \neq \emptyset$, $p_r = \emptyset$. À l'inverse, à chaque utilisateur·trice classique émettant une ou plusieurs requête(s) correspond un participant tel que $p_s = \emptyset$, $p_r \neq \emptyset$. Néanmoins, cette modélisation permet d'autoriser un même participant à avoir à la fois les rôles de source et de requêteur (ce qui correspond, par exemple, un·e blogueur·se effectuant une revue de presse et publiant des articles originaux, ceux-ci n'étant pas le fruit de transformations automatiques).

Assez naturellement, $\forall p \in P, \forall s \in p_s, \exists t \in p_t, t = \emptyset \mapsto s$, dans la mesure où la production des flux initiaux est une activité indépendante du système, qui n'est pas modifiée par lui. Nous noterons donc, par abus, que $p_s \subseteq p_t$. En revanche, seuls les participants procédant eux-mêmes au calcul de leurs propres requêtes inchangées seront tels que $p_r \subseteq p_t$. L'ensemble $p_r \cup p_s$ décrit donc ici les *intentions* du participant, tandis que p_t décrit les opérations réellement effectuées en pratique.

2.2.2 Représentation logique : graphes et hypergraphes

Se basant sur ces différents éléments, nous pouvons représenter un système par un graphe orienté :

Définition 5 (Graphe des participants)**.**
Un système d'échange de flux correspond à un graphe $\langle P, \rightarrow \rangle$ *tel que :*

- P *est un ensemble de participants tels que définis précédemment,*
- $p \rightarrow p'$ *ssi le participant* p' *est abonné à l'un des flux émis par le participant* p *:*

$$p \rightarrow p' \Leftrightarrow \exists t \in p_t, \exists t' \in p'_t, t_{out} \in t'_{in}$$

Ainsi, la figure 2.2(a) présente un exemple de système composé de six participants et trois requêtes : `p0` gère une source de données, puis `p1` et `p2` effectuent directement leurs requêtes sur cette source. `p3` et `p5` récupèrent les flux de sortie de ces deux participants, l'un pour effectuer une transformation les combinant, l'autre parce qu'il a lui aussi exprimé ces deux requêtes. Enfin, `p4` récupère le flux de sortie de `p3`, correspondant aux résultats de sa requête. La symbolique ici utilisée (une couleur par classe d'équivalence de transformation, et une forme différente pour chaque participant) est commune à toutes les représentations de ce type au sein de ce document.

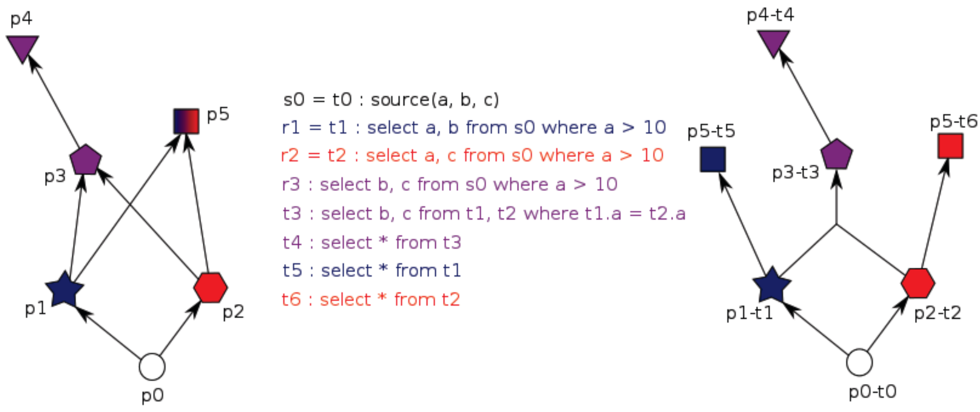
Cette structure *physique* apporte des informations sur la réalité matérielle du système. Comme dans les SGBDr, cependant, elle peut être complétée par une structure *logique*, s'axant sur les relations entre transformations effectuées dans le système plutôt que sur

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

les participants. Pour mettre en place une telle représentation logique, il est nécessaire de pouvoir distinguer les différentes transformations de chaque participant. Pour cela, nous proposons la notion d'*unités de calcul* :

Définition 6 (Unité de calcul)**.**
Une unité de calcul est un composant logiciel en charge de gérer les ressources allouées par un participant à l'une des transformations qu'il effectue. En d'autres termes, une unité de calcul est un triplet $\langle p, r, t\rangle$*, avec* $p \in P$ *un participant du système,* $t \in p_t$ *l'une de ses transformations, et* $r \in R$ *identifie la requête (exprimée ou non par* p*) pour laquelle cette transformation est effectuée.*

On note U_p l'ensemble des unités d'un participant p donné ($U_p = \{\langle p, r, t\rangle, t \in p_t\}$), formant une partition de l'ensemble U des unités du système ($U = \bigcup U_p, p \in P$). Par ailleurs, notant u une unité de calcul, on notera respectivement u_p et u_t le participant et la transformation concernée. Si cette transformation est associée à une requête particulière, celle-ci sera notée u_r, et l'on aura donc $u = \langle u_p, u_r, u_t\rangle \in U_p$. Nous nous autorisons, par abus, à confondre p et U_p (donc à considérer le participant comme l'ensemble de ses unités) chaque fois que cela n'entraînera aucune ambiguïté.



(a) Graphe des participants (b) Hypergraphe des unités

Figure 2.2: Exemple de correspondance entre graphes physique et logique
Chaque forme représente un participant, chaque couleur une requête.

Le recours à ces unités de calcul nous permet de clarifier le graphe physique défini plus haut, en dissociant les nœuds associés à chaque participant de manière à obtenir un nœud par unité de calcul, et donc par transformation. Dans ce cas, les arcs de notre graphe deviennent explicitement liés aux entrées et aux sorties de chaque transformation effectuée. Toutefois, la notion mathématique d'hypergraphe, permettant de considérer des liens d'un ensemble de sommets vers un sommet unique[5], permet une meilleure expression des relations :

Définition 7 (Hypergraphe des unités)**.**
Un système d'échange de flux est représenté par un hypergraphe $\langle U, \rightarrow\rangle$ *tel que :*

- U *est l'ensemble des unités de calculs mises en place par les participants,*

[5] Ou l'inverse, bien que cela ne nous concerne pas ici. Le concept d'*hypergraphe* est une généralisation du concept de *graphe*, par le fait qu'il admet des *hyperarcs* reliant plus de deux sommets à la fois. Dans nos travaux, nous considérons uniquement le cas des hypergraphes orientés dans lesquels chaque hyperarc relie un ou plusieurs sommet(s) à un seul.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

- $u_1, u_2, \cdots, u_n \to u$ *ssi l'unité* u *acquiert les entrées relatives à sa transformation* u_t *auprès des unités* $u_1, u_2, \cdots, u_n$ *:*

$$\langle p_1, r_1, t_1\rangle, \langle p_2, r_2, t_2\rangle, \cdots, \langle p_n, r_n, t_n\rangle \to \langle p, r, t\rangle \Leftrightarrow t_{in} = \{t_{1out}, t_{2out}, \cdots, t_{nout}\}$$

En d'autres termes, chaque hyperarc de ce graphe représente la transformation particulière affectée à l'unité de calcul qui en est le nœud sortant. Un exemple d'hypergraphe représentant un système logique est proposé en figure 2.2(b) : pour la plupart des participants, n'ayant qu'un seul flux d'entrée et qu'une seule transformation à gérer, la correspondance avec le système physique présenté en (a) est donc transparente. La transformation (unique) associée au participant `p3` nécessitant de combiner les flux envoyés par les participants `p1` et `p2`, l'hyperarc menant à l'unité correspondante possède deux bases distinctes. Le participant `p5`, en revanche, gère deux transformations distinctes : il est donc représenté par deux unités pouvant être séparées l'une de l'autre. Cet hypergraphe logique permet donc de différencier clairement deux cas de figure qui, dans le graphe physique, semblent très proches l'un de l'autre.

Notez qu'il est bien sûr possible que plusieurs unités à la base d'un hyperarc correspondent au même participant physique ; de même qu'il est possible qu'une liaison logique n'ait pas de correspondance physique, si l'une des unités d'un participant donné doit récupérer un ou plusieurs flux produits par une autre unité du même participant. Par ailleurs, et par construction, le degré entrant de chaque unité d'un tel hypergraphe est nécessairement de zéro si la transformation associée est une source, et de un sinon.

De cette représentation logique découle la possibilité de vérifier la *validité* du système. Intuitivement, le système est *valide* si et seulement si chaque participant ayant exprimé au moins une requête obtient effectivement les résultats qui y sont liés. Plus formellement, cela repose sur trois aspects :

Définition 8 (Système valide)**.**
Un système d'échange de données est valide si et seulement si :

- *Chaque participant se voit au minimum confié une transformation équivalente à chacune de ses requêtes :* $\forall p \in P, \forall r \in p_r, \exists t \in p_t, t \equiv r$
- *Pour chacune de ces transformations, l'unité correspondante acquiert bien les flux requis :* $\forall u \in U, \forall t \in u_{tin}, \exists u' \in U, u'_{tout} = t, u' \to u$
- *Aucun cycle ne vient perturber l'acquisition des informations :*
$$\forall u \in U, \nexists u' \in U, u' \to u, u \rightsquigarrow u'$$

En notant $u' \to u$ le fait que l'unité u' soit l'une des bases de l'hyperarc entrant de u (mais pas nécessairement la seule), et $u \not\rightsquigarrow u'$ le fait qu'il n'existe pas de chemin, de quelque longueur que ce soit, de u vers u' par $\to$. Notez cependant que cet aspect porte sur les unités de calcul : il n'est *a priori* pas gênant qu'un cycle existe entre les participants, dès lors que celui-ci ne concerne que des unités sans rapport entre elles.

Concernant le premier aspect, soulignons qu'une transformation affectée à un participant ne nécessite pas nécessairement de calcul : dans le cas où les résultats d'une requête r donnée sont calculés par un autre participant, par exemple, le participant p ayant exprimé r ($r \in p_r$) peut se voir chargé d'une simple identité : $\exists t \in p_t, t : \{r\} \mapsto r$. Un tel cas est illustré en figure 2.2 : l'unité associée au participant `p4`, de même que chacune des deux unités du participant `p5`, sont connectées à une unité travaillant sur une requête équivalente à la leur. La transformation qui leur est confiée est donc de simplement récupérer le flux parent sans y apporter de modification.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Notation	Description / explication
F	Ensemble des flux du système
$S \subseteq F$	Ensemble des flux initiaux produits par les sources
T	Ensemble des transformations $t : \mathscr{P}(F) \to F$
$R \subseteq T$	Ensemble des requêtes exprimées par les utilisateurs·trices
P	Ensemble des participants du système
U	Ensemble des unités de calcul correspondant à ces participants
$t_{in} \subseteq F$	Ensemble des flux récupérés par la transformation t
$t_{out} \in F$	Flux de sortie produit par la transformation t
$p_s \subseteq S$	Ensemble des flux sources produits par le participant p
$p_r \subseteq R$	Ensemble des requêtes exprimées par le participant p
$p_t \subseteq T$	Ensemble des transformations effectuées en pratique par le participant p
$U_p \subseteq U$	Ensemble des unités du participant p
$\langle p, r, t \rangle \in U$	Unité du participant p chargée d'effectuer t et associée à une requête r
$u_p \in P$	Participant gérant l'unité u
$u_t \in T$	Transformation effectuée par l'unité u
$u_r \in R$	Requête pour laquelle la transformation u_t est effectuée
$t_1 \equiv t_2$	Équivalence : t_1 et $t_2 \in T$ produisent des flux de sortie égaux
$p' \to p$	Participants reliés dans le graphe physique
$\bigcup\{u_i\} \to u$	Unités reliées dans l'hypergraphe logique
$u' \to u$	Existence d'un hyperarc $\bigcup\{u_i\} \to u$ tel que $u' \in \bigcup\{u_i\}$
$u' \rightsquigarrow u$	Existence d'un chemin reliant u' à u dans l'hypergraphe logique
$P_S \subseteq P$	Participants gérant des sources primaires de données
$P_R \subseteq P$	Participants exprimant des requêtes dans le système (usagers)
$P_A \subseteq P$	Participants actifs, contribuant au fonctionnement du système
$P_{A_C} \subseteq P_A$	Participants actifs contribuant notamment aux calculs
$P_{A_D} \subseteq P_A$	Participants actifs contribuant notamment à la diffusion
$P_I \subset P$	Participants ne contribuant à aucun aspect du fonctionnement

Tableau 2.2: Notations utilisées pour notre modélisation

2.2.3 Structures reconnaissables

Permettant de se focaliser sur les transformations effectuées et leur localisation, notre modélisation permet de caractériser certaines structures communes que l'on peut retrouver dans plusieurs modes d'organisation. Pour cela, il convient de nous doter de plusieurs sous-ensembles de P déterminant différents comportements des participants.

D'une part, les rôles classiques de source et de requêteur peuvent être identifiés aisément : $P_S = \{p \in P, p_s \neq \emptyset\}$ désigne les participants gérant au moins une source de données ; et $P_R = \{p \in P, p_r \neq \emptyset\}$ désigne les usagers du système, à savoir les participants exprimant au moins une requête. Naturellement, il peut exister dans le système des participants n'étant ni sources ni requêteurs, en particulier dans les systèmes répartis sur un parc de machines tiers.

D'autre part, nous distinguons également les participants contribuant à chacun des deux aspects de l'activité du système : $P_{A_C} = \{p \in P, \exists t \in p_t \setminus p_s, \{t_{out}\} \neq t_{in}\}$ pour les calculs et $P_{A_D} = \{p \in P, \exists p' \in P, p \to p'\}$ pour la diffusion. Rien n'empêche naturellement un même participant d'appartenir simultanément à ces deux ensembles ; mais l'organisation peut être conçue de telle sorte que certains participants ne contribueront

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

jamais. L'ensemble P peut donc être partitionné en deux : $P_A = P_{A_C} \cup P_{A_D}$, l'ensemble des participants *actifs*, et son complémentaire, P_I, celui des participants *inactifs*.

Ainsi, les systèmes conçus pour traiter l'ensemble des requêtes, ne demandant pas aux usagers de contribuer au système, sont organisés de telle sorte que $P_R = P_I$. Dans ce contexte, la différence entre les systèmes centralisés et répartis est marqué par le fait que les premiers soit sont gérés directement par les sources ($P_A = P_S$), soit sont constitué autour d'un participant unique collectant les données de chacune des sources et étant le seul à communiquer avec elles :

$$P_A = P_S \cup \{g\}, P_{A_C} = \{g\}, \forall p \in P_S : p \to g \wedge \nexists p' \in P, p' \neq g, p \to p'$$

Dans les systèmes techniquement répartis, en revanche, cet ensemble de participants actifs $P_A \setminus P_S$ est constitué d'un nombre plus important de participants ; dont toutefois aucun n'est un requêteur ($P_A \cap P_R = \emptyset$). Toutefois, dans un cas comme dans l'autre, il est nécessaire, pour garantir la validité du système, que l'ensemble des requêtes soient traitées par les membres de P_A. Cela signifie que le participant centralisateur, le cas échéant, doit gérer une unité par requête.

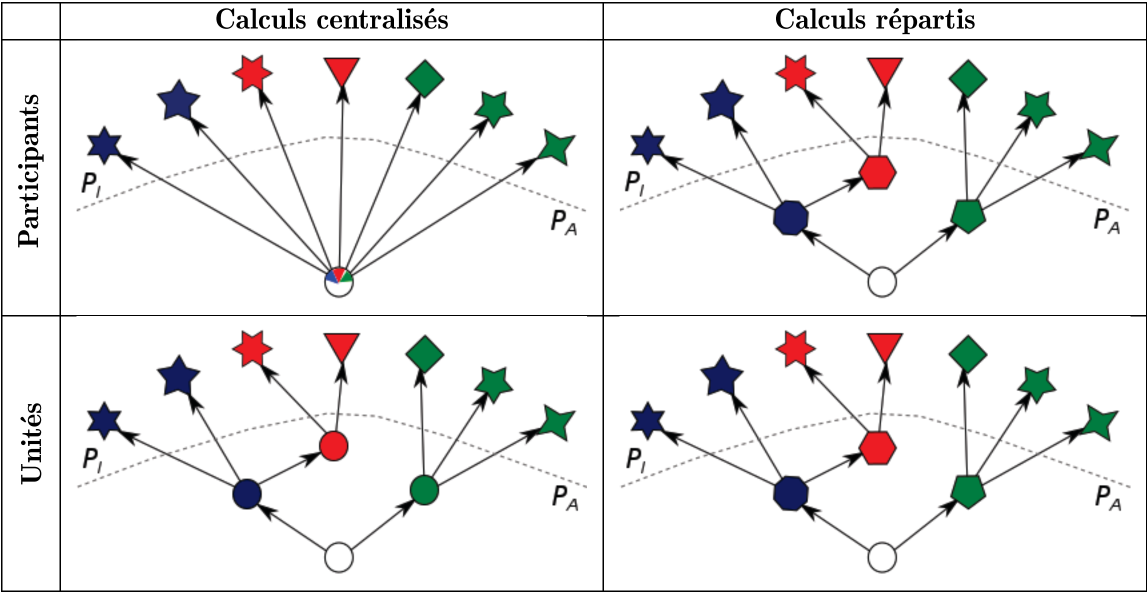


Figure 2.3: Centralisation logique, physiquement répartie ou non

La figure 2.3 illustre que les représentations logiques de ces deux cas sont très proches, explicitant que leurs modes de fonctionnement sont en fait très similaires. Dans les deux cas, chaque source est à la base d'un arbre dont les participants requêteurs sont des feuilles n'effectuant aucune transformation par eux-mêmes, et une « couche de calcul » intermédiaire qui, qu'elle soit distribuée ou centralisée, reste une boîte noire à laquelle les utilisateurs n'ont pas accès.

En revanche, dans les systèmes de pure diffusion dans lequel les flux initiaux des sources sont propagés à l'ensemble des participants, chacun ayant la responsabilité de traiter ses requêtes par ses propres moyens si besoin, la répartition est faite de telle sorte que les seuls participants réellement inactifs sont ceux dont la seule requête est une identité. La charge de calcul dans son ensemble (exception faite de la production des flux initiaux, toujours considérée séparément) est alors portée par les requêteurs ($P_{A_C} \subseteq P_R$).

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Le système le plus simple qui soit envisageable, dans lequel chaque participant traite directement ses propres requêtes sur les flux sources sans relais de diffusions, est dans ce cas ($\forall p \in P = P_R \cup P_S, p_t = p_r \cup p_s$). C'est cette configuration particulière que nous désignons par le terme d'*Unicast*. Toutefois, ce système extrêmement simple, trivialement valide, n'est qu'une des multiples formes que peuvent prendre les systèmes de pure diffusion, auxquels nous allons maintenant nous intéresser.

2.3 Systèmes guidés par la diffusion

Ces systèmes sont ceux pour lesquels les calculs sont laissés à la périphérie du réseau, que ce soit du côté des requêteurs (les flux bruts produits par les sources sont propagés à l'ensemble du système, chacun pouvant ensuite effectuer en local les traitements désirés sur ce flux), soit du côté des sources (les flux échangés sont directement les flux de résultat, et le système est conçu autour de la manière de les propager aux utilisateurs les ayant demandés). Dans les deux cas, donc, les participants peuvent être reliés entre eux sans prendre en compte les relations de réutilisabilité entre les différents résultats, ce qui garantit une forte indépendance vis-à-vis des langages de requêtes utilisés. Toutefois, même si la diffusion des données peut être plus ou moins répartie, la répartition des calculs est soit rendue délicate, soit déportée à l'extérieur du système.

2.3.1 Systèmes non-basés sur le langage

Nous étudions ici tout d'abord les situations qui peuvent être déployées sur un nombre restreint de serveurs, sans recourir à la participation des utilisateurs·trices ; avant de nous intéresser aux cas pour lesquels leurs capacités de diffusion sont mises à profit pour le partage des flux bruts produits par les sources. Une modélisation commune à ces deux modes d'organisation est présentée ensuite, reprenant le formalisme présenté dans la section précédente.

Systèmes non-portés par les utilisateurs·trices

Dans la plupart des cas, la mise en place d'un système distribué sur un parc de machine dédiée se fait en demandant à ces machines de réceptionner les requêtes des usagers et de procéder à des traitements, ce qui semble à première vue les exclure des systèmes de pure diffusion qui nous préoccupent ici.

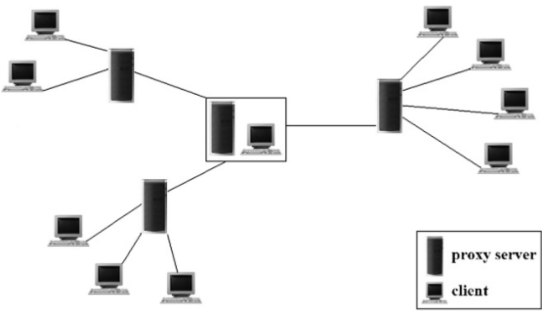


Figure 2.4: Schéma de fonctionnement d'un CDN (repris de [88])

Ainsi, les *Content Delivery Network*, ou « *CDN* », sont un exemple connu de distribution de charge de la source, auxquels sont consacrés plusieurs travaux[84, 88]. Le principe

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

(illustré en figure 2.4) est ici de mettre en place plusieurs *miroirs* chargés de répliquer une source de données : chaque miroir dispose de l'ensemble des données de la source correspondante et est susceptible aussi bien de diffuser les informations brutes que de procéder à des traitements particuliers – ces miroirs pouvant si besoin recourir à diverses techniques d'optimisation locale. Leur rôle est de gérer un nombre de participants trop important pour une seule source, la répartition étant généralement basée sur des critères géographiques. Les CDN sont généralement adaptés aux données statiques et aux échanges ponctuels, mais les mises à jour successives peuvent tout de même apparenter ces usages à ceux de flux de débit modéré.

Confier aux différents miroirs la responsabilité de traiter les requêtes n'est cependant pas constitutive de ce mode d'organisation, et l'on peut tout à fait concevoir une organisation de type CDN pour permettre l'échange de flux inchangés, laissant la responsabilité du traitement aux participants. De plus, même dans le cas où les miroirs assument la charge de calcul, un CDN doit être organisé de telle sorte que l'ensemble des miroirs récupèrent le flux brut sur lequel ils effectueront ces traitements. On peut donc, comme l'illustre la figure 2.5, envisager un système restreint, limité à l'ensemble de ces miroirs, et qui fonctionnera comme un système de pure diffusion.

Cette figure montre en effet la représentation logique d'un CDN traitant les requêtes, mettant en évidence sa division en plusieurs sous-systèmes. Chaque miroir est en situation de source secondaire d'une organisation centralisée, les participants étant répartis entre des différents sous-systèmes. Néanmoins, un système de pure diffusion relie la source primaire à ces différents miroirs, pour leur permettre d'obtenir le flux à traiter.

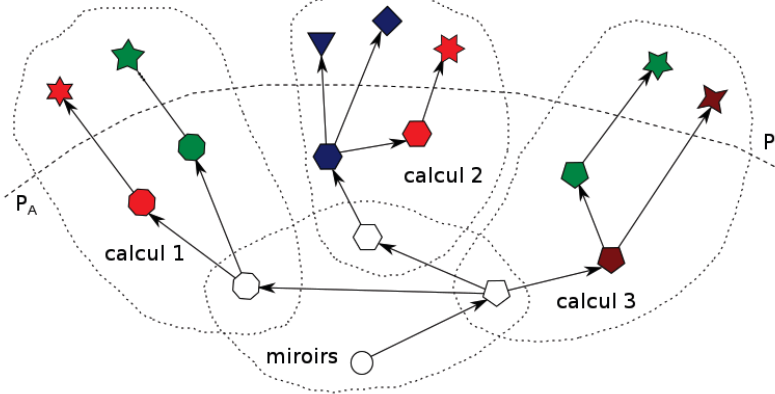


Figure 2.5: Un CDN inclut un système de diffusion même s'il traite les requêtes

Notons que, tant qu'il n'y a aucune modification des flux, il est également possible de ne faire reposer le fonctionnement de l'échange sur aucun participant en particulier, mais plutôt directement sur le réseau : c'est le cas du concept de Multicast[31], dans lequel la répartition des informations vers les différents destinataires est directement prise en charge sans qu'il n'y ait de réplication physique. La mise en place d'un tel mode de fonctionnement peut toutefois s'avérer problématique : s'il est utilisé par certains opérateurs pour la diffusion télévisuelle, ou peut être mise en place dans des réseaux d'échelle locale (*LAN*), le Multicast au sens réseau du terme n'est en revanche pas disponible dans le contexte d'Internet.

Mise à contribution des utilisateurs·trices

Afin de pallier ce problème de mise en place du Multicast au sens réseau du terme, un certain nombre de propositions[50] se sont construites autour du concept de « Multicast

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

applicatif », dans lequel les participants prennent en charge le partage du flux considéré. L'un des intérêts de ces approches est qu'elles permettent de décharger complètement une source de données, ne lui demandant que d'émettre une seule fois chaque élément du flux, comme ce serait le cas en Multicast réseau. Bien sûr, dans le cas d'une source dotée de capacités suffisantes, il est également possible de l'intégrer aux autres participants et de lui demander une participation plus active.

Dans le domaine général du Multicast applicatif, SplitStream[14] est l'une des approches principales. Le principe y est, comme le nom l'indique, de séparer le flux initial, permettant aux différentes informations d'être échangées par différentes routes. Comme illustré sur le schéma présenté en figure 2.6, plusieurs arbres de diffusions distincts sont mis en place, permettant à tous les participants de recevoir les informations sans toujours tenir le même rôle (ici, les participants 2, 3 et 4 sont des feuilles pour l'arbre tracé en pointillé, mais pas de celui en trait plein ; et réciproquement pour les participants 5, 6 et 8). Les données sont envoyées selon l'un ou l'autre de ces arbres, pour équilibrer charges et latence entre les participants.

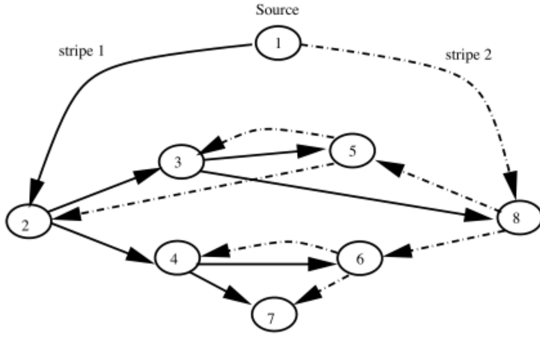


Figure 2.6: Schéma de fonctionnement de SplitStream (repris de [14])

Plus généralement, ces approches, où l'ensemble des participants contribue à la diffusion de l'ensemble des données, peuvent être rapprochées du cas des CDN collaboratifs, comme par exemple Globule[78] : le concept est ici de mettre en place un CDN qui soit entièrement porté par ses utilisateurs·trices, ce qui requiert donc que chacun·e puisse, comme un miroir dans un CDN classique, récupérer le flux complet, même si des traitements peuvent ensuite être réalisés.

Toutefois, gérer le flux de données complet peut présenter une charge assez importante pour les participants, et une telle structure basée essentiellement sur l'échange du flux complet ne leur permet pas de collaborer à ce niveau : même si de nombreux usagers ont des requêtes similaires, chacun d'entre eux doit transférer le flux entier et donc traiter ces requêtes séparément.

Modélisation commune

Nous ne considérons ici que les participants contribuant au système (et nous limitons donc à P_A plutôt qu'à P). En effet, dans le cas des CDN pour lesquels les miroirs traitent les requêtes, nous retombons dans le cas du système centralisé classique décrit plus haut dans lequel $P_R = P_I$, et l'aspect qui nous préoccupe est la mise en place de la diffusion entre les différents miroirs. Dans tous les autres cas ici décrits, les participants sont en charge de procéder à leurs propres calculs et donc actifs même s'ils ne diffusent pas eux-mêmes ($P_R = P_{A_C} \subseteq P_A$).

Cette restriction prise en compte, la caractéristique commune à tous ces systèmes de pure diffusion est que chaque participant concerné gère au moins une unité *miroir* m,

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

dont le but est de manipuler le flux entier. Soit $f_s \in S$ le flux source initial (que nous considérerons unique pour simplifier[6]) :

$$\forall p \in P_A, \exists m \in p, m_r : \{f_s\} \rightarrow f_s, m_{tout} = f_s$$

Sur les systèmes simples, basés sur un unique arbre de diffusion, nous pouvons par ailleurs spécifier que cette unité chargée de gérer le flux ne traitera qu'une identité. En notant $P_M = \{p \in P_A, p_s = \emptyset\}$ l'ensemble des participants tenant le rôle de miroir au sens strict (correspondant donc à P_A privé de la source initiale du flux, pour laquelle $m_{tin} = \emptyset$) :

$$m_{tin} = \{f_s\}$$

Sur des systèmes plus complexes comme celui proposé par SplitStream, en revanche, son rôle sera de reconstituer ce flux source à partir des fragments récupérés dans chacun des arbres de diffusion :

$$\forall p \in P_M, \exists u_1, u_2, \cdots, u_n \in p, u_{it} : \{f_i\} \rightarrow f_i, m_{tin} = \bigcup \{f_i\} = f_s, \bigcup \{u_i\} \rightarrow m$$

Le nombre n d'unités concernées étant ici égal au nombre d'arbres de diffusions distincts constitués pour l'échange de données. Chaque participant disposant d'une ou plusieurs requêtes devant alors traiter celle-ci de manière indépendante, à partir du flux qu'il aura récupéré. Puisque nous avons ici $P_R = P_{A_C}$, cela donne donc :

$$\forall p \in P_R, \forall r \in p_r, \exists t \in p_t, t : \{f_s\} \rightarrow r_{out}, \{m\} \rightarrow \langle p, r, t \rangle$$

Notons que cette dernière expression néglige, pour simplifier les notations, le cas des requêtes combinant plusieurs flux de sources distincts, en ne considérant qu'une seule unité m par participant, mais cette situation est bien sûr possible dans le cas général.

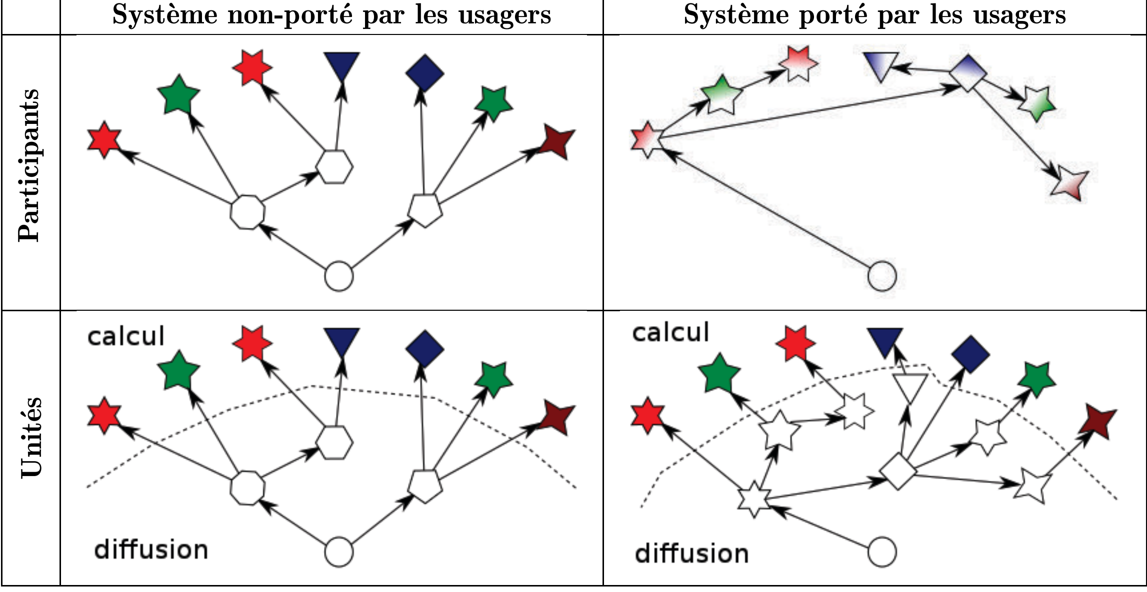


Figure 2.7: Comportement des systèmes à miroirs

[6] Le flux de chaque source étant diffusé inchangé, et les requêtes n'étant traitées que de manière interne à chaque participant, un système de ce type comprenant plusieurs sources aura la forme de plusieurs sous-systèmes juxtaposés, sans relations entre eux.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

La figure 2.7 présente deux exemples de tels systèmes, le second étant porté par ses utilisateurs·trices et le premier ayant recours à des ressources tierces. Dans le système non-porté par ses utilisateurs·trices, les participants supplémentaires, n'exprimant aucune requête par eux-mêmes, ont seulement pour rôle de diffuser le flux brut aux requêteurs, qui ne diffusent rien. Dans l'autre cas, chaque participant doit fournir au système, en plus des unités en charge de traiter ses requêtes, une unité chargée de diffuser le flux brut. L'ensemble des unités de ces deux systèmes, peu importe leur répartition physique, est donc strictement partitionné : $U = U_{A_C} \cup U_{A_D}$, $U_{A_C} \cap U_{A_D} = \emptyset$.

Cette famille d'approches s'avère donc problématique vis-à-vis des aspects de calcul lorsque les capacités de traitement des participants sont limitées. Malgré cela, les possibilités d'optimisation qu'elle apporte en termes de latence et de diffusion en font un point de comparaison important. Le tableau 2.3 récapitule ses caractéristiques principales.

Expressivité des requêtes	aucune limitation sur le langage
Distribution	de $n/1$ (classe 1) à classe 3 (n/m)
Coûts de fonctionnement	aucune optimisation des calculs
Priorités	réduction de latence, respect des capacités de diffusion
Adaptabilité	aux capacités uniquement
Respect des intérêts	non pertinent (mais bon respect de l'intimité numérique)

Tableau 2.3: Principales caractéristiques des systèmes à miroir

2.3.2 Systèmes basés sur l'équivalence des requêtes

Les utilisateurs·trices du système ne disposant pas nécessairement des capacités requises pour gérer le flux complet, il est possible de séparer celui-ci, et donc de demander à la plupart des participants de n'en gérer qu'une partie. Dans certains cas, il est possible de procéder aux traitements en amont, afin d'organiser le système autour de la seule diffusion des résultats attendus. Cela conduit alors à un partitionnement thématique du réseau, sans échange entre les différentes parties concernées.

Une organisation de type CDN peut en effet s'appliquer, de manière assez simple, à un tel partitionnement du système. Il « suffit », pour cela, de demander aux différents miroirs de se spécialiser chacun sur une partie des intérêts concernés, et d'utiliser ce critère d'intérêt pour répartir les simples utilisateurs·trices sur ces miroirs, en lieu et place des critères habituels, tels que le critère géographique.

FlowerCDN[32, 38]) est une proposition allant dans ce sens, fonctionnant sur le modèle d'un CDN collaboratif, donc porté par ses utilisateurs·trices : à partir d'un site web, par exemple, des participants volontaires vont maintenir des copies des pages qui les intéressent, tenant compte des mises à jour successives. Ces copies peuvent alors être consultées plutôt celles du site d'origine, permettant de décharger ce dernier. Se développent ainsi des « pétales » contenant un nombre variable de machines et correspondant aux intérêts exprimés par les utilisateurs pour le contenu du site.

Dans le domaine Pub/Sub, les approches de type *topic-based* reposent elles aussi naturellement sur ce genre de principes. À titre d'exemple, Ripple[94], illustré en figure 2.8, repose sur la mise en place de plusieurs plans d'organisations distincts, un pour chacun des sujets considérés, dans lesquels les flux sont diffusés sur un modèle de « vagues » (« *ripple waves* » en anglais). Les utilisateurs·trices ayant la possibilité de s'abonner à plusieurs sujets, ces plans doivent cependant être coordonnés de telle sorte que les utilisateurs concernés soient toujours ordonnés de la même façon, afin d'éviter que les transferts

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

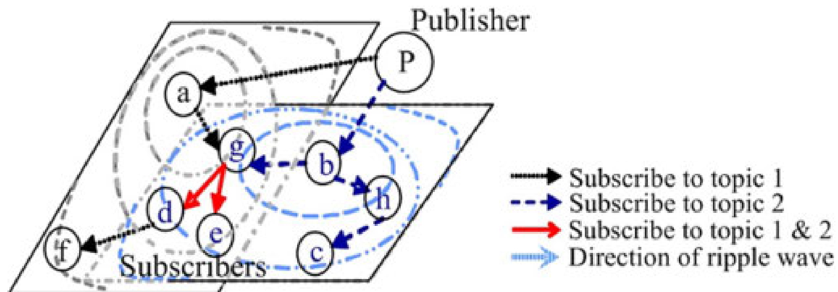


Figure 2.8: Schéma de fonctionnement de Ripple (repris de [94])

d'informations se fassent dans deux sens opposés.

Cette situation dans laquelle un certain nombre d'utilisateurs·trices sont abonné·e·s à plusieurs sujets est fréquente[22] dans les systèmes *topic-based*. Cela est peut-être dû à un manque d'expressivité des requêtes : ces systèmes considèrent généralement les différents sujets comme indépendants, l'utilisateur·trice devant exprimer une souscription différente par sujet. Cela entraîne des problématiques spécifiques[22] qui pourraient être prises en compte différemment sur des langages plus complexes.

Certaines approches *content-based* peuvent d'ailleurs également être considérées comme faisant partie de cette famille. Ainsi, RDF-based P2P[20] repose sur le fonctionnement de base d'un réseau d'échange pair-à-pair ponctuel, auquel des mécanismes propres à ces systèmes Pub/Sub *content-based* sont ajoutés pour permettre, notamment, de récupérer les nouveaux documents arrivant dans le réseau correspondant aux critères désignés (ce qui amène donc une notion proche de celle de flux). L'un des apports majeurs de cette approche est d'aborder la problématique de l'apparition de nouvelles sources de données correspondant aux requêtes existantes, quand les réseaux classiques sont pour leur part construits de manière exclusive autour de la ou des source(s) initiale(s).

L'aspect commun à ces différents systèmes est qu'ils sont partitionnés entre plusieurs sous-systèmes d'échange, chacun dédié à une classe d'équivalence particulière de requêtes. Au sein de ces différents sous-systèmes, les participants s'échangent un unique flux qui, s'il ne correspond pas aux flux initiaux du système complet, est produit par l'unité en position de source secondaire, comme le montre la figure 2.9 :

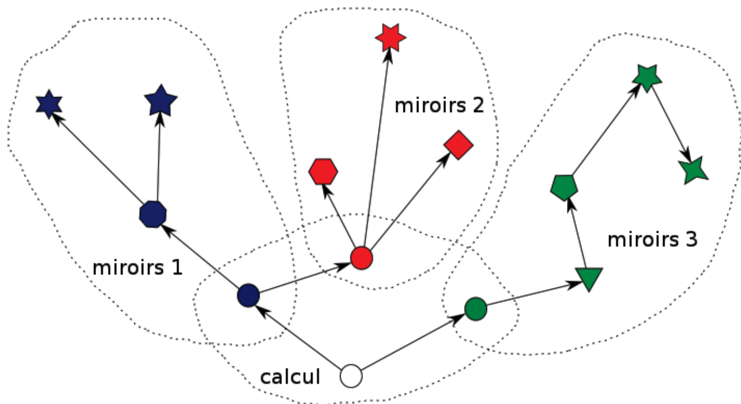


Figure 2.9: Un système à équivalence présente une organisation à miroirs par requête

En d'autres termes, ces systèmes peuvent donc être considérés comme une juxtaposi-

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

tion de plusieurs sous-systèmes à miroirs, constitués autour d'un sous-système de calcul permettant d'obtenir les résultats. Chaque participant gère alors un miroir dédié au résultat de chacune de ses requêtes :

$$\forall p \in P, \forall r \in p_r, \exists m \in p, m_r = r, m_{tout} = r_{out}$$

La mise en place de systèmes de ce type requiert donc la possibilité de pouvoir identifier, parmi l'ensemble des requêtes exprimées, différentes *classes d'équivalences*, permettant de regrouper et de mettre en relation les participants en fonction des résultats attendus. Les différents sous-systèmes constitués correspondent donc, dans une certaine mesure, à une première approche de la notion de communauté.

Toutefois, une fonction de vérification d'équivalence ne donne aucune information sur la façon dont les requêtes seront effectivement calculées, ce qui reste ici la responsabilité d'un ensemble restreint de participants dont les requêteurs sont exclus (contrairement aux approches précédentes, $P_{A_C} \cap P_R = \emptyset$).

Le tableau 2.4 récapitule les caractéristiques des approches de cette famille.

Expressivité des requêtes	langage généralement limité (*topic-based*)
Distribution	calcul : 1/1 à n/1, diffusion : n/m classe 1 à classe 3
Coûts de fonctionnement	aucune optimisation des calculs
Priorités	réduction de latence, respect des capacités de diffusion
Adaptabilité	aux capacités et à la popularité
Respect des intérêts	constitutif de l'organisation

Tableau 2.4: Principales caractéristiques des systèmes à équivalence

2.4 Systèmes guidés par les calculs

L'autre grande famille de systèmes est celle pour laquelle les aspects relatifs au calcul sont pris en compte directement dans l'organisation générale du système. Les échanges sont alors constitués de telle sorte que chaque participant reçoive les données dont il a besoin pour réaliser les calculs qui lui sont affectés, fréquemment plus complexes qu'une simple identité, sans pour autant que le flux d'entrée ne corresponde au flux brut de la source. En d'autres termes, ces approches sont guidées par les calculs, car les relations possibles entre participants sont celles qui permettent aux calculs d'être réalisés au cours de la diffusion. Cela n'empêche bien sûr pas qu'une fois cet ensemble des possibles établi, les aspects liés à la diffusion soient les critères principaux d'organisation du système.

Nous envisageons ici trois catégories distinctes de systèmes. En premier lieu, nous traitons les approches dont le mode d'organisation est spécifique à certains aspects particuliers de la situation considérée, tels que les propriétés avancées du langage. Ensuite, nous étudions plus en détail deux manières de gagner en généricité vis-à-vis de ce langage, l'une portant sur l'identification des séquences d'opérateurs utilisés, et l'autre sur le concept de réécritures de requêtes.

2.4.1 Systèmes à organisation spécifique à un langage particulier

Cette catégorie regroupe une majorité des propositions présentes dans la littérature. On trouve en effet des approches de ce type dans tous les domaines de publication concernés (CQ, DSMS, DSPS, Pub/Sub). Nous proposons ici de considérer ces approches en fonction

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

de leur niveau de répartition : d'abord les systèmes pour lesquels les calculs sont centralisés au sein d'une même machine, puis ceux pour lesquels un parc de machines dédié est mobilisé, et enfin ceux où tous les usagers peuvent potentiellement participer à ces calculs. Nous présentons ensuite une analyse commune à l'ensemble de ces propositions.

Systèmes centralisés et organisation locale des calculs

Un exemple de référence dans ce domaine est STREAM[9], développé par l'université de Stanford, dans lequel les requêtes continues (exprimées en CQL) sont « compilées » en un plan d'exécution, dont la structure peut être déterminée automatiquement ou spécifiée par un document XML. Le gestionnaire des tâches associe ensuite les différents plans d'exécutions entre eux, dans l'optique de mutualiser autant que possible les calculs et la mémoire – en particulier, la mémoire utilisée lors des opérations de bufferisation.

QSystem[52], pour sa part, s'attaque à des requêtes de type *top-K* (récupération des K meilleurs résultats parmi l'ensemble des données disponibles). Cette approche est une illustration de l'intérêt que peut parfois présenter un tiers centralisateur : la source de données considérée est ici une base de données initialement distribuée ; mais les flux constitués par les modifications successives de cette base sont réunis en un même lieu afin de permettre de factoriser les plans d'exécutions des différentes requêtes en vue, encore une fois, de diminuer les coûts.

Ces systèmes permettent une gestion efficace de la charge, mais, naturellement, avec les limites habituellement inhérentes aux systèmes centralisés, en particulier le fait qu'un nombre toujours croissant d'usagers à servir finit toujours par déborder les ressources d'une machine unique. Une première manière de dépasser cette limite peut être de mettre en place plusieurs machines indépendantes, fonctionnant chacune comme un tiers centralisateur unique, mais sur lesquelles les différentes requêtes à traiter sont réparties, à la manière de ce que l'on peut obtenir avec un CDN[84, 88].

À ce titre, l'exemple d'Aurora et de Medusa, conjointement présentés dans *Scalable Distributed Stream Processing*[19], est assez illustratif : le même article porte d'abord sur Aurora, un système de gestion de flux centralisé, puis présente Aurora*, un réseau de plusieurs Aurora distincts dans lesquels certaines tâches peuvent être partagées, avant de proposer Medusa, une infrastructure distribuée reposant sur une analogie au monde économique (mécanisme de « marché » de services, basé sur une monnaie virtuelle) pour gérer les charges. L'ensemble permet une certaine continuité entre les deux domaines (centralisés et distribués), mais nécessite un choix de mise en place exclusif : il est impossible de passer dynamiquement d'un modèle à l'autre en fonction des besoins.

Répartition du calcul sur des nœuds système

Un certain nombre d'autres approches reposent, comme Medusa, sur le fait de répartir les tâches sur un grand nombre de machines, plutôt que de laisser chacune fonctionner de manière indépendante. Ainsi, SPADE[43], l'outil de traitement de flux du projet System S, repose sur une compilation des requêtes permettant de partager différentes parties de plan d'exécution. SODA[92], répartisseur de tâches du même projet, se charge de placer données et traitements conformément à un modèle de coût, cherchant notamment à minimiser une moyenne pondérée d'utilisation des ressources et à maximiser une mesure théorique d'importance pour le système des travaux effectués.

Maîtriser l'infrastructure sur laquelle le système est distribué permet d'en simplifier la gestion. À ce titre, les objectifs et hypothèses formulées dans *Simple Scalable Streaming System*[70] (ou « *S4* »), sont assez représentatives : cette proposition, basée sur Hadoop et

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

destinée au domaine de la fouille de données (*data mining*), suppose que tous les nœuds du réseau partagent les mêmes fonctionnalités et responsabilités, et qu'aucun ne sera ajouté ou retiré à un *cluster* au cours de son fonctionnement.

Les systèmes Pub/Sub, pour leur part, reposaient historiquement sur des tiers chargés de diviser les flux d'informations et d'effectuer des calculs qui pouvaient tout de même être plus diversifiés. SemCast[75], par exemple, repose sur une architecture de *brokers* (« courtiers »), des nœuds système chargés de récupérer les éléments des flux considérés, de les évaluer, puis, le cas échéant, de les rediffuser, comme présenté en figure 2.10.

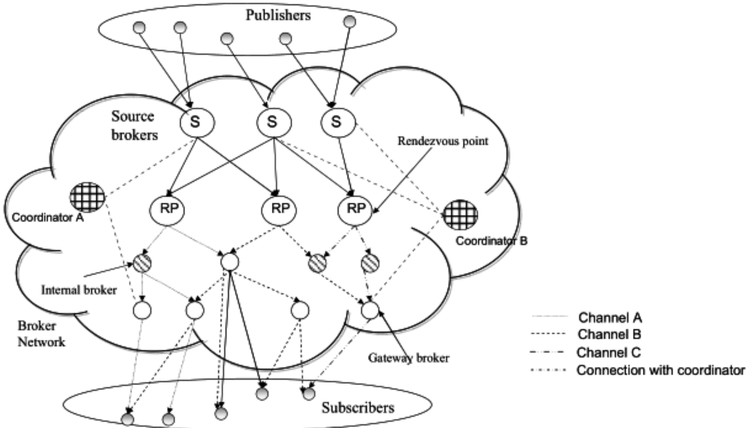


Figure 2.10: Schéma de fonctionnement de SemCast (repris de [75])

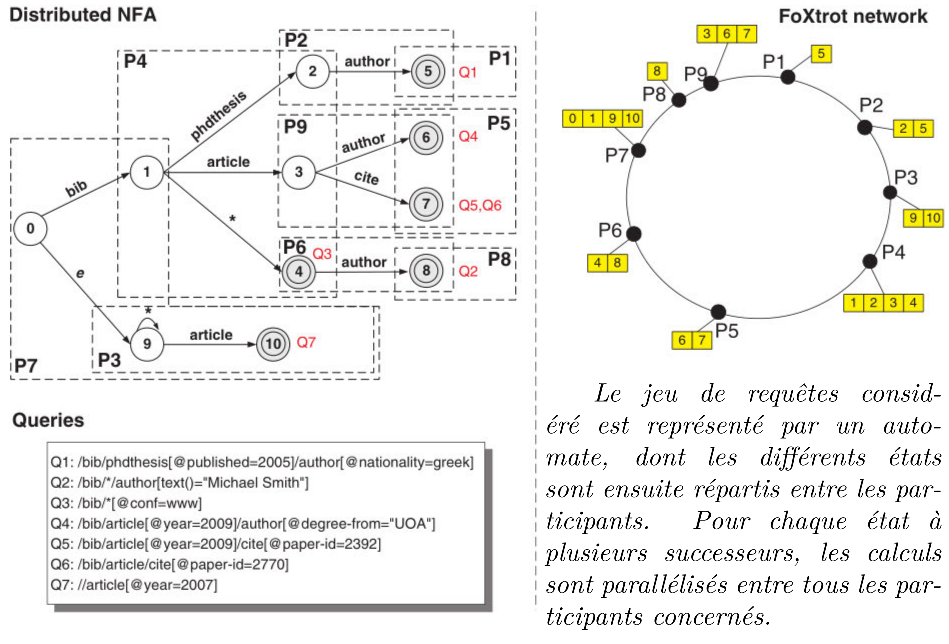
Quoique les deux domaines présentent ici une divergence d'approche notable – les nœuds intermédiaires, dans le monde Pub/Sub, sont plus souvent spécialisés pour certaines tâches en particulier ; tandis que dans les DSPS les considèrent plus souvent comme des ressources indifférenciées –, les systèmes concernés présentent dans les deux cas des caractéristiques de fonctionnement assez similaires. En particulier, comme dans le cas des systèmes entièrement centralisés, ces différents modes d'organisation requièrent de collecter l'ensemble des requêtes émises par les usagers pour organiser un traitement sur une infrastructure entièrement séparée des nœuds requêteurs (ce qui correspond, comme déjà montré en figure 2.3, à une même structure logique). De ce fait, leurs différences ne sont pas sensiblement plus marquées qu'entre deux systèmes centralisés. Toutefois, cette divergence d'approche entre les deux domaines est sans doute pour beaucoup dans le fait que le monde Pub/Sub se soit tourné vers l'intégration des participants requêteurs dans la prise en charge des calculs beaucoup plus massivement que ne l'a fait celui des DSPS.

Mise à contribution des participants exprimant les requêtes

En effet, là où leurs homologues *topic-based*, du fait de leur organisation bien plus centrée sur cet aspect, ont intégré les usagers dans la diffusion uniquement, les Pub/Sub *content-based* récentes se sont affranchis des *brokers* et autres médiateurs utilisés jusque là pour tirer parti des ressources de calcul de ces usagers. Pour ce faire, l'usage des DHT, courant pour des échanges ponctuels, semble avoir également pris une certaine importance. Ainsi, *Distributed Large-Scale Information Filtering*[90] propose une extension à Chord baptisée *DHTrie*, permettant d'ajouter à cette DHT des fonctionnalités propres aux problématiques du Pub/Sub, notamment plusieurs méthodes permettant de retrouver les destinataires d'une publication, qui ne sont pas nécessairement connus à l'avance.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Dans Meghdoot[46], système Pub/Sub plus ancien, les souscriptions utilisateurs sont « installées » dans un espace cartésien à nombre variable de dimensions, en fonction de l'importance qu'ont pour elles les différents attributs considérés (le nombre de dimensions correspondant au nombre d'attributs). Chaque pair du réseau est responsable d'une partie de l'espace total, les informations étant maintenues, là encore, au sein d'une DHT.



Le jeu de requêtes considéré est représenté par un automate, dont les différents états sont ensuite répartis entre les participants. Pour chaque état à plusieurs successeurs, les calculs sont parallélisés entre tous les participants concernés.

Figure 2.11: Schéma de fonctionnement de FoXtrot (repris de [64])

FoXtrot[64], pour sa part, s'attaque au traitement de requêtes XPath en mettant en place, à partir de l'analyse des requêtes considérées, un automate fini non-déterministe (*NFA*), dont les participants sont en charge de maintenir des fragments de taille configurable. Ces NFA correspondent à des structures arborescentes dans lesquels chaque état peut avoir plusieurs successeurs, entraînant une parallélisation appréciable des calculs, selon le principe illustré en figure 2.11.

Quelques propositions de DSPS portés par leurs utilisateurs·trices ont toutefois également vu le jour. SAND[5], par exemple, organise le système en un arbre de traitement, divisé en plusieurs zones. Ces sous-arbres sont en charge de différentes parties de l'exécution commune, chacune étant affectée à un coordinateur chargé de contrôler le bon déroulement des opérations, un coordinateur racine gérant la répartition des zones.

SBON[79], implémenté par une extension d'un DSPS plus classique baptisé Borealis[2], travaille pour sa part essentiellement sur la charge réseau et sur la latence, donnant notamment une importance assez grande à la proximité géographique des différents nœuds.

Modélisation et analyse

Nous voyons donc que la distribution de ces différentes approches se fait dans une certaine continuité : la façon de mobiliser les participants requêteurs, le cas échéant, reste assez proche de celle dont les nœuds système sont sollicités dans les approches distribuées plus classiques. La différence notable est cependant que, dans les approches basées sur l'usage des DHT, l'organisation ne dépend pas d'un tiers unique, passant, selon la terminologie Bortzmeyer[12], d'une classe 1 à une classe 3.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Le prix en est que ces organisations basées sur les DHT concernent des requêtes de moindre expressivité. Elles peuvent être, certes, plus complexes que dans le cas d'un Pub/Sub *topic-based*; mais la prise en compte de requêtes plus avancées nécessite encore la présence (constitutive de la classe 1) d'une entité tierce pour calculer un plan d'exécution commun à l'ensemble des requêtes du système, qui sera ensuite distribué sur les différents nœuds concernés, requêteurs ou non.

Une autre limitation est le fait que ces approches soient soumises à des choix de mise en place exclusifs. Les approches les plus centralisées présentent un certain degré de liberté, dans la mesure où, une fois le calcul effectué de cette manière, des organisations de pure diffusion (basés sur l'équivalence des requêtes) peuvent théoriquement être mises en place pour acheminer les résultats. Toutefois, pour l'aspect calcul comme pour l'aspect diffusion, il s'agit d'un choix à effectuer lors de la mise en place du système et qui ne pourra que difficilement être remis en cause par la suite, aucune approche ne proposant actuellement de passer d'un mode à l'autre dynamiquement. De telles opérations sont pourtant parfois nécessaires. Ainsi, dans le domaine, relativement proche, de la diffusion musicale (*streaming on-demand*), Spotify, initialement connu pour son réseau massivement pair-à-pair[55] a décidé il y a quelques années[1] de revenir à un mode de fonctionnement plus centralisé (de type CDN), correspondant davantage à leurs nécessités du moment.

Par ailleurs, le fait de considérer l'ensemble des participants requêteurs, lorsqu'ils sont sollicités, comme étant des ressources système, entraîne le fait qu'il soit possible et même fréquent (par exemple pour des raisons géographiques dans le cas de SBON) de leur demander des traitements arbitraires, sans rapport aucun avec les requêtes qu'ils ont exprimées ($\exists u \in U : u_r \notin u_{p_r}, \nexists u' \in u_p, u \not\rightarrow u'$) sans pour autant que cette situation ne dépende d'un comportement volontairement altruiste du participant concerné.

La caractéristique principale, à nos yeux, des approches présentées dans cette section reste toutefois le fait que leur organisation nécessite la mise en place de transformations arbitraires, qui ne correspondent directement à aucune des requêtes exprimées par les participants ($\exists u \in U : \nexists r \in R, u_t \equiv r$), mais dont les résultats servent de données intermédiaires pour plusieurs des requêtes à traiter. Ceci entraîne qu'une connaissance avancée du langage utilisé est nécessaire non seulement pour organiser le système, mais également pour l'analyser une fois qu'il est en place, notamment pour en vérifier la validité.

Le tableau 2.5 récapitule les caractéristiques des approches de cette famille.

Expressivité des requêtes	dépendant du langage (constitutif de l'approche)
Distribution	1/1 à n/m, de classe 1 sauf DHT (classe 3)
Coûts de fonctionnement	réduction efficiente des calculs
Priorités	optimisation des calculs
Adaptabilité	bonne pour la connexité, très faible pour les capacités
Respect des intérêts	généralement non-pris en compte

Tableau 2.5: Principales caractéristiques des systèmes spécifiques à un langage

2.4.2 Systèmes basés sur l'extraction des opérateurs

D'autres approches reposent sur des mécanismes que l'on va pouvoir retrouver à l'identique dans plusieurs langages, apportant ainsi un certain degré de généricité. Cette section en présente un premier exemple, basé sur l'extraction de la séquence d'opérateurs qui sera nécessaire à traiter chaque requête. Les deux approches ici présentées concernent les requêtes continues sur des bases de données. Le rôle de telles requêtes est de prendre

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

en compte les modifications apportées sur la base au fur et à mesure : s'il ne s'agit pas exactement de traitement de flux, le fonctionnement reste toutefois globalement similaire.

Dans ce domaine, NiagaraCQ[18] est une approche assez importante, dans laquelle les requêtes sont groupées entre elles par leurs « signatures », dans le but de minimiser les entrée-sortie, qui, dépendant d'opérations physiques, contraignantes. La notion de « signature » ici utilisée correspond aux différentes combinaisons d'opérations à effectuer dans un fichier XML contenant les données actuellement connues (qui sera réévalué à chaque modification). Ainsi, deux sélections portant sur le même attribut correspondront à la même signature, même si la valeur à sélectionner diffère. Une unique lecture du fichier de données permet alors de déclencher plusieurs actions successives (nouveaux traitements ou création des différents flux de résultats), comme le présente la figure 2.12.

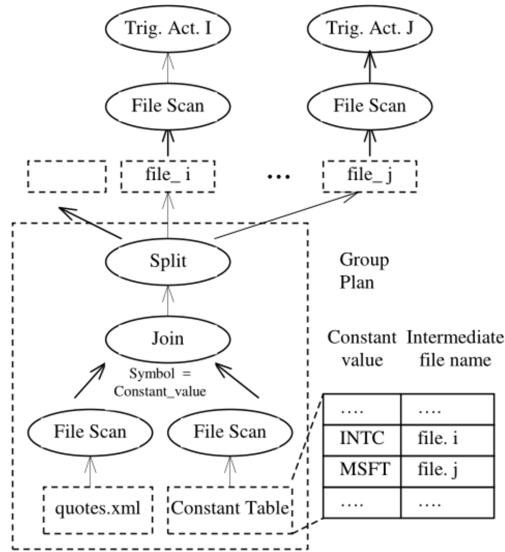


Figure 2.12: Schéma de fonctionnement de NiagaraCQ (repris de [18])

TelegraphCQ[17], légèrement plus récent, propose une approche globalement très similaire. La proposition évoque un concept d'« empreinte » (*footprint*) plutôt que de « signature », mais son rôle est là encore de regrouper les requêtes en fonction de la structure correspondante. Cette approche, initialement envisagée dans le cadre de systèmes de gestion de base de données classiques plutôt que celui de fichiers XML, a pour objectif de mutualiser autant que possible les accès disques.

En effet, toute requête, quel que soit le langage dans lequel elle est exprimée, finira tôt ou tard par être transcrite en une séquence d'opérateurs physiques à mobiliser. Disposer d'une fonction qui, étant donné un langage particulier, fournit la liste des opérateurs qui devront être appelés permet de se focaliser sur eux plutôt que sur les résultats qui seront générés. C'est ce qui permet de regrouper toutes les lectures d'un attribut donné en un seul passage, même quand elles attendent différentes valeurs pour cet attribut. De ce point de vue, il s'agit d'un cas-limite de notre modélisation, des « transformations » produisant plusieurs flux de sortie pouvant être plus représentatives.

Ces approches présentent cependant l'inconvénient d'être plus difficiles à répartir, tirant précisément leur efficacité du fait que les calculs sont effectués au même endroit. Il serait possible de distribuer partiellement ce mode de fonctionnement en confiant à plusieurs participants la charge d'effectuer chacun un opérateur pour l'ensemble des requêtes concernées, mais cette « distribution » demeurerait limitée à un nombre réduit de

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

nœuds, conservant les problématiques inhérentes au monde centralisé (nécessité de collecter l'ensemble des requêtes et probable goulot d'étranglement si le nombre de requêtes à traiter devient critique). De plus, la participation ne peut ici qu'être décorrélée des requêtes exprimées, demandant une fois encore aux usagers, s'ils sont mobilisés, d'effectuer des traitements arbitraires.

Le tableau 2.6 récapitule les caractéristiques des approches de cette famille.

Expressivité des requêtes	dépendant du langage (approche générique)
Distribution	difficile à mettre en place hors 1/1
Coûts de fonctionnement	réduction efficiente des calculs
Priorités	mutualisation des opérateurs
Adaptabilité	principalement à la connexité
Respect des intérêts	non pertinent (centralisé)

Tableau 2.6: Principales caractéristiques des systèmes à opérateurs

2.4.3 Systèmes basés sur les réécritures

Une autre approche envisageable, pour que le système repose sur une API commune à plusieurs langages plutôt que sur les spécificités d'un langage donné, est de baser le fonctionnement sur le concept de *réécritures de requêtes*, qui peut, pour sa part, permettre de laisser les participants ne travailler que sur les requêtes qu'ils expriment.

Notion de réécritures de requêtes

Le concept de réécritures de requêtes provient des SGBDr classiques, dans lequel il est une pratique usuelle concernant le traitement des requêtes[45]. Une première forme de réécritures consiste en le fait de reformuler une requête pour supprimer les opérateurs redondants ou simplifier les opérations à effectuer. Ainsi, une requête SQL définie par :

```
SELECT DISTINCT id FROM table
WHERE NOT champ < valmin AND NOT champ > valmax;
```

présente deux points pouvant être réécrits. D'une part, savoir que la colonne « id » correspond à une clef primaire, et ne peut donc contenir de doublons, permet de conclure que le calcul de l'opérateur `DISTINCT` est inutile. D'autre part, évaluer une condition puis prendre la négation du résultat peut être simplifié en calculant directement la condition opposée. Cette requête peut donc être réécrite plus efficacement en :

```
SELECT id FROM table
WHERE champ >= valmin AND champ <= valmax;
```

Ces deux expressions sont strictement *équivalentes*, c'est-à-dire qu'elles produiront toujours les mêmes résultats, quelles que soient les données présentes dans la table concernée. L'intérêt d'utiliser la seconde requête plutôt que la première est ici d'éviter les calculs superflus, le plan d'exécution étant calculé en fonction de l'expression utilisée. Plus généralement, la recherche d'une réécriture pour une requête donnée permet donc, en fonction de la structure logique de la base, à déterminer l'expression équivalente qui amènera les meilleures performances.

Cette recherche de réécritures peut, si la base en comporte, s'appuyer sur des *vues matérialisées*. Ces vues sont des ensembles de données intermédiaires, définies à l'aide d'une requête utilisant un langage identique à celui des requêtes utilisateurs. En d'autres

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

termes, ces vues correspondent à des résultats de requêtes qui sont maintenus dans le temps par le système, susceptibles d'être utilisés en lieu et place des tables réelles, permettant potentiellement de simplifier la recherche de résultats. Plus formellement[47],

Définition 9 (Réécriture équivalente)**.**
Soit r une requête et $V = \{v_1, v_2, \cdots, v_n\}$ un ensemble de vues (chacune d'entre elles définie par une requête). La requête r' est une réécriture équivalente de r utilisant V si :

- *r' acquiert ses données auprès de V uniquement,*
- *r' est équivalent à r.*

Ainsi, l'on peut par exemple disposer d'une vue `v1` définie comme suit :

```
SELECT * FROM table
WHERE champ <= valmax;
```

Les résultats de la requête précédemment mentionnée sont strictement inclus dans les données de cette vue : d'une part, l'opérateur `*` permet de sélectionner l'ensemble des colonnes de la table, dont l'identifiant fait bien évidemment partie ; d'autre part, les deux valeurs maximales correspondant, les données qui seront exclues de la vue le seront également de la requête. De plus, cette vue permet de simplifier les calculs en diminuant le nombre de données à considérer, mais également en dispensant de tester l'une des conditions de la requête, dont on sait qu'elle sera toujours vérifiée par les données fournies par la vue. Utilisant cette vue `v1`, la requête peut donc être réécrite en :

```
SELECT * FROM v1
WHERE champ >= valmin;
```

Cette réécriture est toujours équivalente à la requête de départ. Elle présente cependant, par rapport à la précédente, une différence notable, qui est que les données utilisées ne sont plus récupérées directement dans la table : on constitue cette fois l'ensemble des résultats attendus à partir d'un ensemble de résultats intermédiaires, dont le maintien est géré par une autre partie du système, le coût étant divisé entre les deux plans d'exécutions. Bien sûr, l'exemple présenté est ici un exemple trivial, n'utilisant qu'une vue unique avec peu de conditions, quand des exemples réels peuvent nécessiter des jointures et autres opérations plus complexes mobilisant plusieurs vues.

La réécriture de requêtes en utilisant des vues matérialisées est un problème étudié[47] dans le domaine des bases de données. Si ses effets en termes de charge de calcul et de nombre de données à manipuler sont assez intéressants, il convient cependant de noter que sa complexité varie en fonction du langage, et peut être assez élevée. Ainsi, le problème est connu pour être NP-difficile concernant les langages SQL[59] et DataLog[3].

Il existe des algorithmes (Bucket[58] ou Minicon[80], par exemple) ayant pour but de déterminer des réécritures à partir d'un ensemble de vues pour des requêtes conjonctives. Si cela ne permet pas nécessairement de prendre en compte toutes les possibilités offertes par les langages complexes, cela offre néanmoins une très solide base de travail permettant d'utiliser la réécriture de requêtes de manière efficace dans de nombreux domaines.

Lorsqu'il est possible de s'appuyer sur des vues matérialisées, la recherche de réécriture peut être décomposée en deux aspects : d'une part, déterminer quelles combinaisons de vues permettent d'obtenir les résultats demandés ; d'autre part, comment obtenir ces résultats de la manière la plus efficace pour ces différentes combinaisons.

Il peut être possible de réduire la taille du premier de ces deux sous-problèmes par heuristique, en considérant que les vues n'ayant rien en commun avec la requête considérée

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

n'ont aucune chance d'être utiles dans quelque combinaison que ce soit. Ainsi, la vue `v2` définie ci-après n'a aucune chance d'avoir le moindre intérêt pour le calcul de la requête précédemment citée :

```
SELECT champ, autrechamp FROM table
WHERE champ > valmin;
```

En effet, même si elle porte sur la même table, la requête correspondant à `v2` ne sélectionne que des colonnes qui ne sont pas demandées dans les résultats de notre requête (même si l'une d'elles est utilisée dans ses conditions). Les données fournies par cette vue n'auront donc rien en commun, et la vue peut être laissée de côté. En revanche, on peut considérer une vue `v3` définie comme suit :

```
SELECT id, autrechamp FROM v1, v2
WHERE v1.champ = v2.champ;
```

Cette dernière vue fournit des résultats communs avec ceux de notre requête, étant donné qu'elle porte sur la même colonne `id` et que ses conditions ne sont pas entièrement incompatibles avec celles qui sont demandées. Il est donc possible de considérer `v3` comme *pertinente* (« *relevant* » en anglais) pour réécrire la requête ; et elle devra être proposée à l'algorithme de réécritures.

Cet exemple permet de mettre en lumière deux aspects importants de la pertinence. D'une part, une vue pertinente peut n'avoir qu'une partie de ses résultats commune avec ceux de la requête. Ici, `v3` fournit une grande partie des données attendues, mais qui n'en est pas un sur-ensemble, dans la mesure où il manque toutes les lignes pour lesquelles `champ = valmin`. Il ne s'agit pas non plus d'un sous-ensemble, dans la mesure la colonne `autrechamp` n'est pas utile et où ses données seraient donc retirées.

Il est possible de ne considérer que les vues pertinentes fournissant un strict sur-ensemble des résultats attendus. Dans ce cas, il ne sera jamais nécessaire de combiner plusieurs vues : chaque requête soit pourra être réécrite à partir d'une vue unique, soit ne pourra pas être réécrite. Cela correspond au concept d'*inclusion de requêtes* (« *containment* » en anglais), qui peut donc être considéré comme une forme limitée de réécritures.

D'autre part, la relation de pertinence n'est pas nécessairement transitive (ici, `v3` est pertinente pour notre requête ; `v2` est pertinente pour `v3` (puisqu'elle y est utilisée comme source) ; mais `v2` n'est pas pertinente pour notre requête). En revanche, toute vue pertinente reposera nécessairement sur au moins une vue elle-même pertinente (ici, `v1`), les données communes (ici, la colonne `id`) devant bien être récupérées quelque part.

Les systèmes de réécritures ont principalement[61] été considérés dans les environnements centralisés, en particulier en ce qui concerne la sélection des vues à utiliser. Rien n'empêche toutefois de les envisager dans un contexte plus distribué. Dans ce cas, la *minimalité* des combinaisons de vues utilisées[21] devient un aspect particulièrement important à prendre en compte. En effet, une réécriture est dite *minimale* lorsqu'aucune des vues utilisées n'est redondante par rapport à une autre, c'est-à-dire lorsque retirer une vue de la réécriture entraîne nécessairement des pertes de données. Bien évidemment, l'usage de combinaisons non-minimales de vues entraînerait, dans le cadre d'échanges réseau, une source de transferts inutiles qui pourraient être économisés.

Toutefois, même dans un système constitué d'un grand nombre de participants reliés par le réseau, le concept de réécritures peut bien sûr être utilisé de manière centralisée avec une certaine efficacité, dans le cas où un participant donné se voit demandé de travailler sur plusieurs requêtes (qu'il les ait lui-même exprimées ou que celles-ci lui aient été confiées par d'autres).

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Échanges de flux et réécritures locales

Optimization of Continuous Queries with Shared Expensive Filters[69] (ci-après désigné comme « *shared filters* ») présente par exemple le cas de requêtes continues constituées de filtres possiblement coûteux. Par nature, les requêtes de filtres ne modifient pas les éléments du flux considéré, et leurs flux de résultats sont une sous-partie du flux initial utilisé. Cette approche propose d'ordonner l'ensemble des requêtes, puis de réaliser des traitements successifs, de telle sorte que le flux de sortie d'une requête donnée puisse être utilisé comme flux d'entrée pour toute requête dont les résultats seraient strictement inclus dans ceux de la requête précédente.

Il s'agit donc d'une approche basée sur l'inclusion de requêtes (ou *containment*), permettant à la fois de diminuer le nombre d'éléments à évaluer (ce nombre diminue d'autant que la requête considérée arrive tardivement dans le classement), et d'économiser le calcul des filtres qui ont déjà été appliqués aux éléments considérés.

Il n'est cependant pas forcément nécessaire de grouper toutes les requêtes du système pour procéder à de telles réécritures. Ainsi, RoSeS[89] s'intéresse pour sa part au problème de requêtes complexes exprimées par un même participant. L'idée développée dans la proposition est de combiner les plans d'exécution des différentes requêtes à traiter, ce qui présente l'avantage, pour le participant l'utilisant, de diminuer sa charge de calcul. Cela peut par ailleurs avoir une certaine influence sur le reste du réseau dans le cas où ces combinaisons permettent d'obtenir les résultats attendus en exprimant dans le système une requête différente de celles formulées par l'utilisateur·trice.

Initialement envisagée pour le suivi de flux RSS, cette approche, illustrée en figure 2.13, peut potentiellement s'appliquer dans n'importe lequel des systèmes décrits précédemment, y compris les plus centralisés d'entre eux. Quoiqu'elle puisse permettre de diminuer les charges de calcul et de réseau de plusieurs participants (celui effectuant ces opérations, et ceux chargés de lui envoyer les données), elle reste une optimisation locale, interne, et non une organisation globale du système.

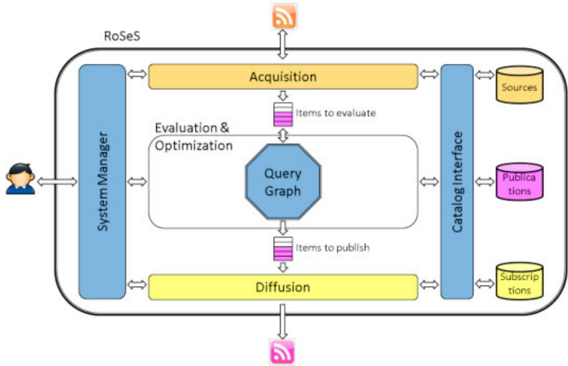


Figure 2.13: Schéma de fonctionnement de RoSeS (repris de [89])

Il est bien sûr possible de mettre en place une approche utilisant un mécanisme de réécritures sur l'ensemble du système ; quoique ce cas de figure semble moins présent dans la littérature que ce que nous envisagions initialement. D'après nos connaissances actuelles, deux propositions en particulier tendent à se rapprocher de nos attentes à ce sujet : *Semantic Peer-to-Peer Overlays for Publish/Subscribe Networks*[16] et Delta[53].

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Semantic Peer-to-Peer Overlays for Publish/Subscribe Networks

Cette approche[16], ci-après désignée comme « SemPO », est un système distribué basé sur l'inclusion de requêtes. Le principe général est ici de placer les utilisateurs dans un arbre de diffusion qui soit organisé logiquement, afin de diminuer autant que possible le nombre de données non-pertinentes à transférer d'un participant à l'autre.

Leur analyse repose sur les notions de *faux négatifs* (tuples non-reçus qui auraient été pertinents) et de *faux positifs* (tuples reçus mais non-pertinents). Évidemment, les faux négatifs sont à éviter absolument, puisqu'ils correspondraient au fait, pour un participant donné, de ne pas recevoir les résultats attendus. L'usage des relations d'inclusion permet de s'assurer que ces faux-négatifs sont évités, et donc tous les résultats attendus dûment reçus, dans la mesure où un participant n'est placé à la suite d'un autre que si l'ensemble des résultats qu'il attend est un sous-ensemble de celui des résultats qu'attend son parent. Les faux positifs, pour leur part, ne peuvent être entièrement évités dans cette approche, un participant renvoyant toujours ses propres résultats, lesquels correspondent à un sur-ensemble strict de celui de certains enfants. Une telle organisation conduit toutefois à en limiter grandement le nombre.

En pratique, cependant, si cette organisation réduit efficacement la charge réseau, elle n'impacte la charge de calcul de chaque participant que par la diminution du nombre de tuples en entrée : chaque participant est considéré comme n'ayant pas d'informations spécifiques sur ceux qui le précèdent dans cet arbre, ce qui conduit à réévaluer la requête entière, non-réécrite, sur l'ensemble des tuples entrant, et donc à réévaluer des conditions déjà analysées et qui n'ont donc aucune chance d'être utiles.

L'organisation de l'arbre de diffusion logique de SemPO repose sur un fonctionnement incrémental : chaque nouveau participant est inséré en parcourant l'arbre déjà existant à la recherche de la position qui lui fournira le flux entrant le plus proche possible de son flux sortant. En particulier, si un autre participant du système effectue déjà une requête strictement équivalente à la sienne, un nouveau venu deviendra un successeur direct de celui-ci. Cette propriété d'équivalence logique est également utilisée en cas de départ ou de disparition : si le participant qui quitte le système avait un ou plusieurs successeurs traitant la même requête (aux équivalences près), l'un de ceux-ci est alors chargé de le remplacer, ce qui permet de minimiser le nombre de modifications de l'arbre de diffusion.

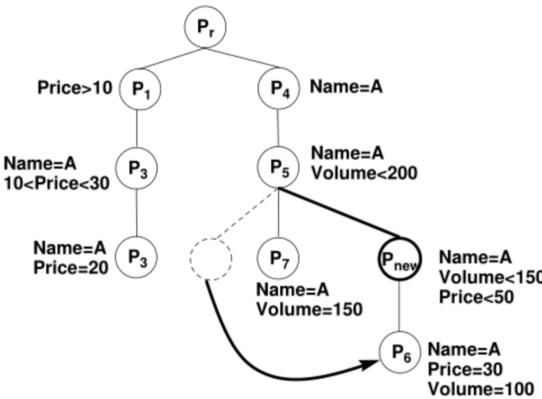


Figure 2.14: Opportunité de réorganisation dans SemPO (repris de [16])

Une question importante posée par cette étude est celle de la pertinence de réorganiser entièrement le système après l'insertion de nouvelles unités (qui, comme présenté sur la

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

figure 2.14, peuvent créer des opportunités de placements plus efficaces pour celles préalablement insérées). Cette réorganisation peut en effet permettre de diminuer les charges fonctionnelles, mais présente un coût d'organisation important, puisqu'elle nécessite de réévaluer les placements d'un grand nombre de participants. Les résultats présentés tiennent compte de trois cas (pas de réorganisation, réorganisation occasionnelle et réorganisation systématique), et tendent à montrer que, sur les jeux sur lesquels les auteurs ont travaillé, la réorganisation est d'impact suffisamment faible pour qu'il soit envisageable de s'en dispenser. Cette considération dépend cependant énormément des jeux utilisés (en particulier, de l'ordre d'arrivée des requêtes), et les conclusions pourraient être différentes dans d'autres cas.

L'inconvénient majeur de cette approche, à notre sens, est qu'elle ne tient pas compte des limitations de ressources de diffusion des différents participants. Comme mentionné ci-dessus, par exemple, un nouveau venu traitant une requête équivalente à celle d'un précédent nœud du système sera placé immédiatement après ce dernier, sans considération du nombre d'enfants que ce nœud parent possède déjà. Le problème est analogue pour les autres cas, où le parent offrant le moins de faux positifs est sélectionné sans considération de charge. En fait, comme nous l'apprend *Minimum Maximum Degree Publish-Subscribe Overlay Network Design*[73], cette considération de degré sortant, pourtant essentielle, a été pendant assez longtemps absente de la littérature.

Nous considérons toutefois que le mode d'organisation de SemPO en fait un point de comparaison important, sous réserve de lui apporter deux modifications : d'une part, utiliser de véritables réécritures (même si limitées à l'inclusion de requêtes) pour éviter les traitements rendus superflus par le fait qu'ils ont déjà été effectués par les participants précédents (ce qui n'est qu'une optimisation locale sans influence sur l'organisation du système) ; d'autre part, prévoir un mécanisme supplémentaire permettant de limiter autant que possible les débordements de capacités. Le tableau 2.7 récapitule les caractéristiques de la proposition d'origine.

Expressivité des requêtes	dépendant du langage (approche générique)
Distribution	n/m, classe 1 d'origine mais classe 3 ou 4 envisageable
Coûts de fonctionnement	réduction efficiente des échanges inutiles
Priorités	réduction des « faux positifs »
Adaptabilité	connexité et popularité (capacités non-prises en compte)
Respect des intérêts	constitutif de l'approche

Tableau 2.7: Principales caractéristiques de SemPO

Delta

Cette approche[53], plus récente et qui repose explicitement sur l'utilisation de réécritures utilisant des vues, porte en revanche une attention toute particulière à cette considération de degré sortant, et plus généralement de limitation de capacités.

Il s'agit ici d'une organisation globale du système, visant à décharger la source de données sans outrepasser les limites de diffusion des autres participants. Cette organisation repose sur cinq étapes distinctes :

1. Toutes les vues présentes dans le système, correspondant aux différentes requêtes émises par les participants, sont d'abord comparées deux à deux, dans le but de former un *embedding graph* (que l'on pourrait traduire par « graphe de pertinences »),

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

présentant les relations de réutilisabilités possibles entre ces vues : un arc relie une vue à une autre s'il est théoriquement envisageable d'utiliser la vue de base dans une réécriture de la requête correspondant à la vue d'arrivée, avant de savoir si une telle réécriture sera possible en pratique.

2. Ce graphe des pertinences pouvant présenter des cycles, qui risqueraient de perdurer dans les étapes suivantes, il est ensuite fait appel à un algorithme[37] issu de la théorie des graphes, permettant de rendre le graphe acyclique.

3. Après quoi, le système de réécritures concerné (la proposition originale reposant sur le langage XQuery, le système de réécriture est celui présenté dans *Efficient XQuery Rewriting using Multiple Views*[62]) calculera, pour chaque requête, autant de réécritures que possible à partir du graphe des pertinences. Le résultat de cette opération est un autre graphe, le *graphe des réécritures* (de type graphe et/ou, comme présenté en figure 2.15, ce qui s'apparente à nos hypergraphes).

   Rechercher l'exhaustivité des réécritures possibles risquant d'être assez coûteux, l'approche de Delta est de n'en rechercher qu'un nombre limité. Une estimation d'intérêt des différentes vues permet d'augmenter les chances que les réécritures les plus intéressantes soient trouvées en premier, ramenant cette recherche à un *Top-K* (ce qui pourrait s'avérer difficile à mettre en place sur d'autres langages).

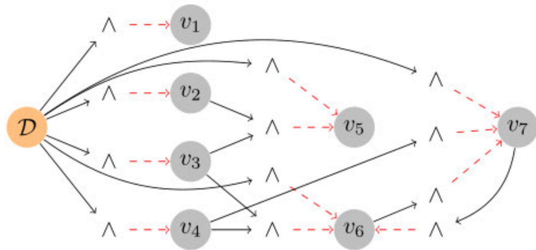


Figure 2.15: Exemple de graphe de réécritures dans Delta (repris de [53])

4. Un solveur linéaire (Gurobi[74] dans la proposition originale) est ensuite appelé sur ce graphe des réécritures, afin de sélectionner pour chaque vue, la plus intéressante parmi les différentes possibilités.

   Trois types de contraintes sont transmises à ce solveur : en premier lieu, bien sûr, le fait qu'à chaque vue doit être affectée exactement une requête (réécrite ou non), qui détermine le degré entrant. Ensuite, la limite de diffusion du participant qui indique la limite maximale de degré sortant. Enfin, le coût estimé de chaque réécriture, qu'il faut chercher à minimiser.

   Les auteurs indiquent avoir envisagé de faire entrer la latence dans ces contraintes également ; mais avoir renoncé à cette idée car la latence globale du système n'évolue pas de façon linéaire et ne peut donc pas être prise en compte par le solveur.

5. L'ultime étape a donc pour objectif de réduire cette dernière : l'algorithme baptisé *LOGA* compare le graphe de solution fourni par le solveur linéaire au graphe des réécritures présenté précédemment, et peut alors modifier les placements des participants, quitte à réaugmenter les coûts de calcul si besoin, pour diminuer autant que possible la profondeur moyenne du graphe.

   Cela se fait, bien sûr, en fonction des réécritures calculées précédemment, et dans la limite des capacités de diffusion non-encore utilisées.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Une fois cette organisation calculée, une négociation peut avoir lieu entre deux participants devant échanger un flux d'informations, le nœud émetteur pouvant envoyer ses résultats tel qu'il les obtient (auquel cas le nœud récepteur doit effectuer l'entièreté des calculs), aussi bien que calculer la requête et envoyer directement le flux attendu. N'importe quelle situation intermédiaire, où le calcul serait partagé entre ces deux participants, peut d'ailleurs également être envisagée.

Quoique l'une des étapes importantes de cette proposition soit le recours à un solveur linéaire, destiné à fournir un résultat optimal en fonction de la base de connaissances de départ, il convient de noter que l'organisation calculée en pratique présente de forts risques de s'éloigner de l'optimalité réelle. En effet, même sans tenir compte du fait que le LOGA remet en cause le résultat obtenu, l'algorithme de suppression des cycles utilisé[37] est un algorithme générique de traitement des graphes, ne tenant donc pas compte des spécificités du réseau considéré, comme la limite de degré sortant : même en considérant que le système de réécritures trouvera effectivement les plus intéressantes possibilités en fonction des vues sur lesquelles il travaille, il n'aura potentiellement à sa disposition qu'une partie des relations de pertinence disponibles, lui faisant manquer des opportunités. Et même parmi les relations résultantes, la limitation à k réécritures au maximum fait que la base de connaissances fournie au solveur sera incomplète.

Par ailleurs, l'organisation ne tient pas compte de l'état précédent du système, ce qui nous semble problématique, car il peut entraîner un nombre important de débranchements d'une réorganisation à l'autre. L'algorithme de suppression des cycles utilisé n'est par ailleurs pas déterministe, ce qui augmente d'autant les risques que les réécritures trouvées, et donc sélectionnées, diffèrent d'une étape à l'autre, sans que cela ne se traduise nécessairement par une amélioration des charges. Notons d'ailleurs que de telles réorganisations ne sont pas systématiques, le processus d'organisation complète étant trop lourd pour être appliqué à chaque arrivé ou départ d'un participant.

Dans l'ensemble, Delta présente de fortes chances d'être une organisation efficace pour les systèmes dans lesquels la popularité des requêtes est assez faible (les jeux de tests proposés vont jusqu'à trois expressions de chaque requête dans le système, ce qui reste très peu par rapport au contexte que nous envisageons), mais risque de s'avérer beaucoup moins intéressante pour les cas dans lesquels de nombreux participants vont exprimer des requêtes équivalentes (ce qui signifie que le graphe des pertinences présentera beaucoup de cycles, qui seront autant d'opportunités perdues après leur suppression).

De plus, s'il était possible d'envisager une mise en œuvre de SemPO dont l'organisation soit répartie entre les participants, le mode de fonctionnement de Delta, et en particulier la présence d'un ILP, suppose la présence d'un intervenant unique connaissant l'ensemble des requêtes du système qui sera bien plus délicat à contourner. Le tableau 2.8 récapitule les caractéristiques de cette proposition.

Expressivité des requêtes	satisfaisante dans le langage proposé
Distribution	n/m, de classe 1
Coûts de fonctionnement	réduits sous contrainte de capacités
Priorités	réduction des coûts (ILP) puis de la latence (LOGA)
Adaptabilité	optimisation globale non-systématique
Respect des intérêts	constitutif de l'approche

Tableau 2.8: Principales caractéristiques de Delta

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Modélisation commune

Contrairement aux systèmes dont l'organisation est spécifique au langage, qui pouvaient autoriser des transformations intermédiaires arbitraires, les systèmes présentés dans cette section, basés sur l'utilisation d'un système de réécritures, sont donc caractérisés par le fait que les seules transformations exécutées dans le système, en dehors de la production des flux initiaux, correspondent aux requêtes des utilisateurs·trices, réécrites ou non :

$$\forall u \in U, \exists p \in P, \exists r \in p_r \cup p_s, u_r = r, u_{tout} = r_{out}$$

Le cas des négociations dans Delta pourrait faire exception à ce principe, si deux participants se mettaient d'accord pour traiter chacun une partie de la requête concernée ; mais il s'agirait toutefois d'une optimisation quasi-locale indépendante du système général, et donc sans importance dans le graphe logique décrivant ce dernier.

En conséquence de cette caractéristique sur les unités, tous les hyperarcs du système logique, qui relient ces unités entre elles, correspondent eux aussi à une requête particulière, réécrite si les unités qui lui servent de base ne sont pas des sources primaires de données (rappelons que, nous focalisant sur les aspects réseau, nous considérons comme identiques deux transformation ayant même entrée et même sortie, et ne tenons donc pas compte, pour ce graphe logique, des réécritures n'influant pas sur les flux d'entrée). L'arc peut donc être étiqueté par cette réécriture.

Chaque réécriture d'une requête donnée mobilisant une ou plusieurs vue(s) pouvant elle(s)-même(s) être réécrite(s), plusieurs étapes peuvent être nécessaires pour retrouver la correspondance entre la requête d'origine et sa version réécrite. Nous nous reposons, pour une telle analyse, sur la notion de *schéma de réécriture*[33] :

Définition 10 (Schéma de réécriture)**.**
Le schéma de réécriture d'une requête $q_r \in R$ est un hypergraphe fortement connexe et dénué de cycles $\langle Q, \rightarrow \rangle$, avec $Q \subseteq R$ un ensemble de requêtes, tel que :

- $\forall \{q, q_1, q_2, \cdots, q_n\} \in Q^{n+1}$, $\{q_1, q_2, \cdots, q_n\} \rightarrow q$ *signifie qu'il existe une réécriture de q utilisant $q_1, q_2, \cdots, q_n$,*
- *le degré entrant de chaque nœud est au plus de un,*
- *q_r est la seule requête ayant un degré sortant de zéro.*

La *concrétisation* d'un schéma de réécriture consiste en le fait de remplacer chaque vue par sa définition, partant de la requête q_r à laquelle est dédiée ce schéma, jusqu'à ce que la définition ne contienne plus que des sources primaires de données. Il en découle la possibilité de vérifier la validité du système :

Propriété 1 (Validité d'un système à réécritures)**.**
Un système à réécritures est valide ssi le chemin reliant les sources primaires de données à chacune des unités requêtrices forme, pour la requête correspondante, un schéma de réécriture se concrétisant en un nombre fini d'étapes pour obtenir une requête équivalente à la requête d'origine.

En effet, la concrétisation du schéma de réécriture permet de s'assurer que les entrées fournies à chaque requête étudiée permettent bien d'obtenir le flux de résultat attendu ; et le fait que cette concrétisation se termine en un nombre fini d'étapes assure l'absence de cycles dans le système. La figure 2.16 présente un exemple de correspondance entre un

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

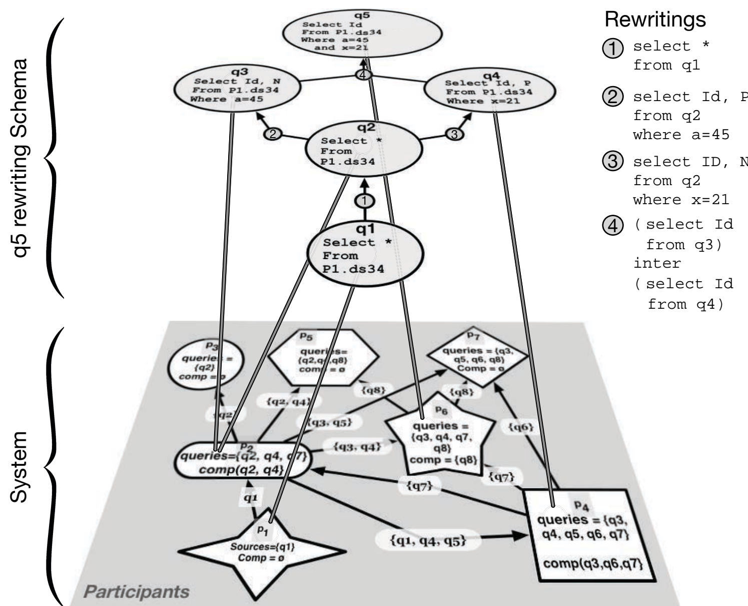


Figure 2.16: Schéma de réécriture et système valide (issu de [33])

système à réécritures (présenté sous la forme d'un graphe de participants) et le schéma de réécriture de l'une de ses requêtes.

Nous pouvons en outre caractériser les systèmes basés sur l'inclusion de requêtes comme une catégorie particulière de systèmes à réécritures, pour laquelle il est impossible d'utiliser plus d'une vue à la fois pour réécrire chaque requête, et dans laquelle tout flux de sortie d'une unité (exception faite, évidemment, des sources primaires de données) est constitué d'un sous-ensemble des éléments utilisés par le flux parent :

$$\forall u \in U, \nexists u', u'' \in U, u' \rightarrow u, u'' \rightarrow u, u' \neq u''$$

$$\forall u \in U, u_{tin} = \emptyset \vee \exists! f \in F, u_{tin} = \{f\}, u_{tout} \subseteq f$$

Synthèse

Comme on l'a vu, l'état de l'art est partitionné sur plusieurs aspects. En particulier, les domaines auxquels se rattachent les différentes propositions sont variés (CQ, DSMS, Pub/Sub, DSPS...). Nous considérons toutefois que cet aspect n'est pas représentatif des divergences et proximités réelles entre les approches : le tableau 2.10 dresse un récapitulatif de la nouvelle classification que nous proposons, s'appuyant pour sa part sur la dépendance vis-à-vis du langage de requêtes.

Un nombre important d'approches orientées requêtes sont conçues spécifiquement pour un langage donné – qui sont à classer en deux familles principales : d'un côté les langages issus du monde des bases de données (SQL et ses dérivés (CQL...), Datalog) ; de l'autre les langages liés au monde du Web (interrogation de documents XML (XQuery, XPath...) ou basés sur RDF) –, quoiqu'il soit parfois possible de reprendre la logique de fonctionnement pour l'appliquer à d'autres formes de requêtes. Le manque d'adaptabilité est également notable en ce qui concerne la localisation des calculs, souvent un élément de choix d'architecture qui ne sera plus remettable en cause par la suite. Le tableau 2.9

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

récapitule le niveau de distribution atteint par certaines approches, décomposé selon les aspects de calcul et de diffusion.

Distribution de la diffusion

Distribution des calculs	**1 site / 1 contrôleur**	***n* sites / 1 contrôleur**	***n* sites / *m* contrôleurs**
1 site / 1 contrôleur	NiagaraCQ[18] TelegraphCQ[17] STREAM[9] shared filters[69]		Ripple[94]
***n* sites / 1 contrôleur**		S4[70] Medusa[19] CDN[88, 84] SemCast[75]	
***n* sites / *m* contrôleurs**	Unicast	CDN[88, 84]	SplitStream[14] RoSeS[89] SemPO[16] Delta[53]

Tableau 2.9: Différents modes de distribution

Dans ce tableau, les CDN sont présentés dans deux cases différentes : ils peuvent en effet être conçus pour traiter ou non les requêtes, ce qui mobilise ou non les usagers du système pour cette tâche. Quoiqu'elles aient été placées dans le contexte dans lequel elles ont été conçues, d'autres approches pourraient présenter un degré de liberté similaire : ainsi, les systèmes de traitement centralisés pourraient se voir adjoindre un mécanisme de diffusion réparti, et ainsi occuper d'autres cases de la première ligne. Un système basé sur l'équivalence, comme Ripple[94], pourrait fonctionner de manière analogue si les calculs étaient répartis sur des nœuds système. Enfin, RoSeS[89] peut être appliqué à l'identique quelle que soit l'organisation extérieure, tant que les participants traitent leurs requêtes, et pourrait donc occuper d'autres cases de la dernière ligne.

Même dans la catégorie des systèmes entièrement pair-à-pair, la prise en compte des participants laisse généralement de côté plusieurs aspects importants, comme leurs intérêts, condition pouvant pourtant s'avérer nécessaire au fait qu'ils consentent à porter le système, ou certaines limitations techniques (cas, notamment, de l'absence de limite pour le degré sortant dans SemPO[16]). La popularité des requêtes, que l'on peut pourtant envisager comme un axe majeur dans la problématique de diminution des coûts liés à la redondance des calculs, n'est abordée que de manière très secondaire.

Pour autant, quoique ces différents points conduisent, nous semble-t-il, à la nécessité de proposer une nouvelle approche, susceptible de mieux prendre en compte des aspects qui seront primordiaux dans les cas d'utilisation auxquels nous nous intéressons, il convient de noter que de nombreuses propositions présentent des points forts qui, même pris en dehors de leur domaine de prédilection, peuvent s'avérer particulièrement intéressants.

Ainsi, les possibilités d'optimisations locales des calculs offertes par la centralisation peuvent s'avérer un avantage primordial, dès lors que cette centralisation ne devient ni un goulot d'étranglement, ni un abandon du contrôle du système. À l'opposée, la répartition des calculs et la possibilité d'échanger des résultat permettent une gestion des ressources entre les différents participants qui assure un système très équitable, dès lors que l'on prend attention à ne pas contraindre les participants à travailler hors de leurs intérêts.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Domaine de publication d'origine

Systèmes guidés par les calculs / par la diffusion

	Bases de données **(CQ, DSMS, DSPS...)**	**Réseau** ***(Middleware / P2P)***
Systèmes **indépendants** **du langage** **Section 2.3.1**		CDN[88, 84] Globule[78] SplitStream[14]
Systèmes **basés sur** **l'équivalence** **Section 2.3.2**	FlowerCDN[32, 38]	Ripple[94] RDF-based P2P[20]
Systèmes **à organisation** **spécifique** **Section 2.4.1**	STREAM[9] System S[43, 92] SAND[5] SBON[79]	SemCast[75] FoXtrot[64] Meghdoot[46] DHTrie[90]
Systèmes **basés sur** **les opérateurs** **Section 2.4.2**	NiagaraCQ[18] TelegraphCQ[17]	
Systèmes **basés sur** **les réécritures** **Section 2.4.3**	shared filters[69] RoSeS[89] Delta[53]	SemPO[16]

Tableau 2.10: Domaine d'origine de quelques propositions

Les réseaux de distribution mis en place dans les systèmes à miroirs assurent eux aussi une répartition efficace des charges, tout en garantissant une latence raisonnable. Ceux-ci nécessitent toutefois d'être complétés par une prise en compte des problématiques de coût de traitement. Les systèmes à réécritures, pour leur part, assurent une prise en charge efficace de ces coûts de calculs, mais pêchent généralement par la difficulté de mise en place résultant de la complexité des relations entre l'ensemble des participants − difficulté qui dépend tant du nombre de nœuds à considérer que du niveau d'expressivité attendu dans la recherche de réécritures.

La modélisation que nous proposons permet, d'une part, d'exprimer chacune de ces approches dans un langage commun, facilitant leur comparaison ; d'autre part, de catégoriser aisément ces approches en mettant en lumière les points communs dans leurs constructions. Appliquer ce formalisme dès la conception d'une proposition rend alors possible d'identifier les aspects pour lesquels des écueils connus sont à éviter, ou au contraire, pour lesquels il existe des solutions tout à fait satisfaisantes.

Le défi de la conception du système QTor était donc celui de fournir une approche nouvelle qui soit suffisamment ouverte pour pouvoir reprendre et exploiter ces points forts sur les aspects pour lesquels ils sont indispensables, tout en évitant les écueils qui y sont jusque là liés. Plus spécifiquement, nous espérions un système comparable au Multicast applicatif pour ce qui est de la réduction de latence, mais qui soit susceptible de diminuer les coûts autant que peuvent le faire les systèmes à réécriture tels que SemPO[16] ou Delta[53], pour des coûts d'organisation limités. Les principes d'organisation que nous avons mis en place dans ce but sont détaillés dans le prochain chapitre.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Chapitre 3

Système à torrent de requêtes

L'étude de l'état de l'art, présentée au chapitre précédent, nous a permis de conclure à la nécessité de proposer une approche nouvelle, mieux à même de correspondre aux différents contextes pouvant être rencontrés. Cette approche, que nous avons baptisée *QTor* (pour « *Query Torrent* » en anglais, c'est-à-dire « torrent de requêtes ») repose sur l'usage de *communautés d'intérêt* regroupant les participants émettant des requêtes équivalentes, et d'un système de réécritures permettant de déterminer la façon dont ces communautés sont mises en relation.

Au cours de ce troisième chapitre, nous présentons de façon détaillée le mode de fonctionnement de cette approche, dont les intuitions ont été données en introduction. Pour ce faire, nous commençons par spécifier précisément la façon dont nous envisageons le problème. Puis, nous présentons les lignes générales de notre approche, et les outils dont nous nous dotons pour pouvoir y répondre. Ensuite, nous détaillons notre manière de mettre en œuvre cette approche de manière incrémentale. Enfin, nous analysons les propriétés qui découlent de ces mécanismes et dont la mise en place d'un système à torrent de requêtes permet de bénéficier.

3.1 Formalisation du problème

Le problème général que notre proposition vise à résoudre est celui de la mise en place, étant donné un ensemble connu de sources émettant des flux et sur lesquels un grand nombre d'utilisateurs·trices vont émettre des requêtes, d'un système permettant auxdits utilisateurs·trices de recevoir les résultats de ces requêtes. Les différents critères d'analyse présentés en section 2.1 (pages 19 et suivantes) forment autant d'aspects à prendre en compte pour envisager ce problème, et la façon de les prendre en compte influe grandement sur la solution pouvant être proposée.

L'objet de cette section est donc de spécifier la façon dont nous envisageons ce problème, qui justifie les choix effectués dans notre approche. Pour cela, il est nécessaire de commencer par définir ce que sont les *fonctions d'organisation* du système, puis de déterminer quelles priorités donner aux différentes contraintes s'appliquant sur ces fonctions d'organisation, définissant par là nos objectifs.

3.1.1 Fonctions d'organisation

Au cours du chapitre précédent, nous avons décrit ce qui correspondait aux *résultats* d'une approche, à savoir une forme finale à laquelle l'organisation proposée permet d'aboutir à un instant t de la vie d'un système. Notre objectif est ici de spécifier la façon

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

dont ce résultat est obtenu. En d'autres termes, quoique nous conservions l'ensemble des notations résumées dans le tableau 2.2 (page 34), nous sortons ici des conceptions mathématiques immuables : l'ensemble P des participants du système évolue au cours de la vie de celui-ci, et il devient nécessaire de distinguer différentes étapes $P_0, P_1, \cdots, P_n, P_{n+1}, \cdots$.

De même pour le graphe $\langle P, \rightarrow \rangle$ décrivant les relations entre participants (définition 5 page 31), qui est pour sa part susceptible d'évoluer sans qu'il n'y ait nécessairement de variations sur P, si les échanges de flux entre les différents participants sont modifiés. Il est donc nécessaire de disposer d'une *fonction d'organisation* capable de déterminer le prochain état du système :

Définition 11 (Fonction d'organisation)**.**
Une fonction d'organisation Ω est une fonction qui, étant donné un état $\langle P_n, \overset{m}{\rightarrow} \rangle$ du système et un événement e donné, détermine l'état suivant $\Omega(\langle P_n, \overset{m}{\rightarrow} \rangle, e) = \langle P_{n+1}, \overset{m+1}{\rightarrow} \rangle$ qui devra être mis en place et déclenche si besoin les modifications concernées.

Les *événements déclencheurs* pris en compte par cette fonction peuvent être de différents types. Les plus fréquemment considérés sont sans doute l'arrivée ou le départ d'un participant, et l'expression ou le retrait d'une requête par un participant déjà intégré et qui le reste par ailleurs ; mais il est également possible d'envisager des causes extérieures susceptibles de remettre en question l'organisation du système, comme une variation importante de la volumétrie des flux sources entraînant des changements notables dans le coût de traitement des requêtes.

Reposer uniquement sur une telle fonction d'organisation Ω induit une organisation incrémentale, dans laquelle chaque état du système est obtenu relativement à l'état précédent. Ainsi, on considère généralement que le premier élément déclencheur rencontré dans l'état du système consiste en l'arrivée du participant exprimant la première source de données (attendu que les requêtes sont exprimées en fonction d'une ou plusieurs source(s) déjà connue(s)[1]), passant ainsi d'un ensemble vide de participants à un premier état du système. Celui-ci se construit ensuite au fur et à mesure.

De tels fonctionnements incrémentaux peuvent éventuellement conduire à des situations où l'historique du système empêche d'atteindre des configurations optimales. En effet, certains choix effectués lors d'itérations précédentes peuvent s'avérer ne pas être judicieux vis-à-vis de modifications ultérieures, et pourtant difficiles à remettre en cause. Néanmoins, il peut tout de même être possible, dans les situations de ce type, de rechercher des configurations relevant de la notion micro-économique d'optimum de Pareto, telle que définie dans *Microeconomic Theory*[63] :

Définition 12 (Optimalité de Pareto)**.**
Soit x une ressource et u une fonction de satisfaction. Une allocation de ressources $(x_1, \ldots, x_n)$ est Pareto-optimale (ou Pareto-efficiente) s'il n'est possible de mettre en place aucune autre allocation $(x'_1, \ldots, x'_n)$ telle que $u_i(x'_i) \geq u_i(x_i)$ pour tout $i = 1, \ldots, n$ et $u_i(x'_i) > u_i(x_i)$ pour un i donné.

En d'autres termes, un optimum de Pareto est une situation dans laquelle il n'est pas possible d'améliorer la situation globale sans dégrader celle d'au moins un participant.

Toutefois, il peut parfois sembler préférable de remettre en cause l'ensemble de l'organisation afin de rechercher une configuration optimale au sens strict. Pour cela, certaines

[1] Le problème de la *découverte dynamique* de sources permettant d'exprimer des requêtes sans connaître à l'avance toutes les sources de données qui permettront d'y répondre, abordé notamment dans *RDF-based P2P*[20] offre néanmoins des perspectives intéressantes, qui seront l'objet de futurs travaux.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

approches reposent donc, en complément de la fonction d'organisation, sur l'utilisation d'une *fonction d'optimisation* :

Définition 13 (Fonction d'optimisation)**.**
Une fonction d'optimisation $\mathcal{W}$ *est une fonction qui, étant donné un état* $\langle P_n, \overset{m}{\rightarrow}\rangle$ *du système, détermine l'état suivant* $\mathcal{W}(\langle P_n, \overset{m}{\rightarrow}\rangle) = \langle P_n, \overset{m+1}{\rightarrow}\rangle$ *qui devra être mis en place et déclenche si besoin les modifications concernées.*

Cette fonction est donc assez similaire à la fonction d'organisation classique, à ceci près que son rôle n'est pas de réagir à un événement donné, mais de remettre en cause l'ensemble des relations existant entre participants afin d'atteindre autant que possible une configuration optimale, au prix de modifications pouvant être d'ampleurs assez importantes.

Du fait du coût important que peuvent représenter la recherche d'une configuration optimale et la modification du système depuis son état non-optimal précédent, une telle fonction d'optimisation n'est pas nécessairement conçue pour être systématiquement appelée. On pourra ainsi considérer que la plupart des événements conduisant à modifier l'état du système déclenchent le seul appel à la fonction Ω, qui reste toujours nécessaire, mais que la fonction $\mathcal{W}$ sera pour sa part appelée à intervalles réguliers (par exemple, toutes les dix itérations de Ω). Alternativement, certaines mesures de l'état du système peuvent être mises en place (par exemple, une évaluation de la latence globale), conduisant à déclencher $\mathcal{W}$ chaque fois qu'un certain seuil est dépassé.

3.1.2 Contraintes et objectifs

Notre objectif principal est de proposer une organisation aussi souple que possible, susceptible d'être utilisée dans la plupart des situations pouvant être rencontrées. Pour ce faire, nous avons tout d'abord choisi de ne pas dépendre d'un langage de requêtes en particulier, mais de mettre en place des mécanismes d'organisation capables de tirer profit de propriétés génériques pouvant être retrouvées dans la plupart des langages.

Nous avons par ailleurs choisi de considérer que tout participant dans le système doit nécessairement avoir le rôle de source et/ou celui de requêteur : $\nexists p \in P, p_s \cup p_r = \emptyset$. Cette contrainte permet en effet de prendre efficacement en compte les situations dans lesquelles le système est porté par ses utilisateurs·trices (auxquel·le·s correspondent des participants requêteurs) ; mais n'exclue pour autant pas la mise en place de ressources tierces pour aider au fonctionnement du système. La seule réserve est que ces dernières doivent être explicitement dédiées à certaines requêtes en particulier, en fonction des besoins rencontrés dans le système.

La question de l'autorité de cette organisation, sur laquelle porte la classification Bortzmeyer[12] (présentée en page 25), devient une question importante dès lors que l'on considère les utilisateurs·trices comme susceptibles de contribuer au fonctionnement du système. Notre objectif est ici de permettre à terme une organisation de classe 3 ou 4, mais, ces organisations étant généralement plus délicates à mettre en place, nous nous sommes autorisés dans un premier temps à reposer sur un *tracker* centralisé (caractéristique d'une classe 1). Ce document présente nos différents algorithmes dans ce contexte, tout en fournissant, pour chaque aspect de mise en œuvre, des informations sur la façon dont l'organisation pourrait, en l'état actuel de nos travaux, être répartie.

Quoiqu'elle nécessite une connaissance exhaustive du système, potentiellement difficile à mettre en place dans le système réparti visé à terme, la présence d'une fonction d'optimisation $\mathcal{W}$ ne nous semble pas *a priori* à exclure. Toutefois, une mise en place

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

périodique d'une telle fonction nous semble problématique. En effet, plus cette période est élevée, plus le système s'éloignera de la configuration optimale entre deux occurrences. À l'inverse, une période courte permet de maintenir un système bien plus fréquemment proche de l'optimal, mais pour un coût d'organisation qui peut s'avérer bien plus élevé que nécessaire. Nous avons donc choisi de travailler sur la mise en place d'une fonction d'organisation Ω seule, avec l'objectif de pouvoir déterminer, lors de l'appel à cette fonction, si une optimisation globale du système pourrait ou non s'avérer utile.

Le problème que notre proposition vise à résoudre est donc la mise en place d'une fonction d'organisation Ω répondant à trois sortes d'événements :

- L'expression d'une nouvelle requête par un participant (qui intègre alors le système s'il n'était pas déjà présent),
- Le retrait d'une requête par un participant (qui quitte le système lors du retrait de sa dernière requête),
- Le changement de capacités d'un participant.

Cette fonction d'organisation doit systématiquement fournir un système *valide*, au sens de la définition 8 (page 33) et respectueux des limites de capacités des participants. Toutefois, chaque participant peut potentiellement être sollicité au maximum des capacités qu'il déclare, principalement pour ce qui concerne la diffusion, dans le but de réduire la latence globale. Par ailleurs, un participant ne peut être amené à travailler sur une requête différente de celle qu'il a exprimée que sous deux conditions :

- Soit ce travail est strictement requis pour obtenir les résultats de l'une des requêtes qu'il traite (la transformation correspondante comptant ce flux parmi ses entrées),
- Soit l'avis du participant a été sollicité et celui-ci a donné son accord explicite (par exemple par la définition d'une politique d'acceptation).

Un déploiement en situation réelle devrait également tenir compte de la capacité des participants à communiquer entre eux, du fait des limitations imposées notamment par les NAT[41]. Considérant toutefois l'existence de diverses propositions[93, 82, 91] visant à contourner ce problème, nous considérons qu'il s'agit d'une problématique d'implémentation qui n'a pas à être prise en compte à ce niveau.

En revanche, le coût de la fonction d'organisation elle-même nous semble un élément primordial à prendre en compte : nous nous fixons comme objectif principal de pouvoir déterminer, à chaque itération de la fonction Ω, le prochain état du système pour un coût raisonnable, et de ne remettre en cause les liens existants entre participants que dans le but de diminuer les coûts de fonctionnement globaux.

3.2 Approche QTor

Ces contraintes étant posées, nous pouvons désormais présenter notre approche proprement dite. Ceci peut se faire en deux parties : la présente section aborde les aspects généraux, c'est-à-dire les caractéristiques par lesquelles nous proposons de définir un système à torrent de requêtes. La section suivante porte pour sa part sur la façon dont nous proposons de mettre en place un tel système.

Indépendamment de la façon dont ce système est mis en œuvre, trois caractéristiques principales nous semblent constitutives de notre approche. La première d'entre elles est le fait de regrouper les participants au sein de *communautés d'intérêt*. La seconde porte

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

sur la façon de mettre en relation ces communautés. La troisième, enfin, consiste en la manière de choisir quelles relations ces communautés auront entre elles.

3.2.1 Système communautaire basé sur les requêtes

Nous savons que, dans de nombreux cas[68], les participants expriment des requêtes équivalentes les unes aux autres. Or, deux requêtes mathématiquement équivalentes (confer définition 3 page 30) présentent les mêmes caractéristiques vis-à-vis notamment de la connexité : quoiqu'elles puissent être exprimées différemment si le langage utilisé autorise plusieurs expressions synonymes, elles s'intéressent aux mêmes sources primaires et visent à recevoir les mêmes résultats. De ce fait, toute requête connexe avec l'une d'elles le sera également avec l'autre, et réciproquement.

Considérer chacune de ces requêtes indépendamment des autres revient donc à répéter plusieurs fois une analyse dont le résultat est déjà connu et immuable, ce qui présente un surcoût d'organisation pouvant être évité. Par ailleurs, cela peut augmenter la difficulté de l'organisation, dans la mesure où, quel que soit le mode d'organisation choisi, la présence de cycles dans l'hypergraphe des unités obtenu rend le système invalide. Étant donné que des participants exprimant des requêtes équivalentes présentent, pour toute analyse préalable, une composante fortement connexe, étudier chaque requête indépendamment entraîne donc un nombre important de cycles à éviter ensuite.

Regrouper ces participants selon les *classes d'équivalences* de leurs requêtes semble donc une manière efficace de simplifier l'organisation en évitant ces deux écueils. Considérant, toutefois, que la vérification théorique d'équivalence n'est pas systématiquement simple à mettre en place en pratique, nous préférons ne pas nous baser sur la notion mathématique théorique de classe d'équivalence, mais sur la notion plus concrète de *communauté* :

Définition 14 (Communauté)**.**
Une communauté est un triplet $\langle r, U_r, t \rangle$*, dans lequel :*

- r *est une expression de la requête à laquelle est dédiée la communauté,*
- U_r *est un ensemble des unités étiquetées comme correspondant à une requête* u_r *équivalente à* r*.*
- t *est la transformation affectée à la communauté pour récupérer les résultats de la requête* r*.*

Assez naturellement, nous notons que la transformation affectée à la communauté doit être équivalente à sa requête ($t \equiv r$). Toutefois, les unités de U_r ne sont pas nécessairement toutes destinées à gérer le flux de sortie commun r_{out}, mais peuvent être affectées à toute tâche interne à la communauté. Étant donné une communauté c, nous notons respectivement c_r et c_t la requête et la transformation concernées. Nous nous autorisons par ailleurs à confondre c et U_r, donc à utiliser directement chaque communauté comme l'ensemble des unités qui la composent, chaque fois que cela n'entraînera aucune ambiguïté.

Naturellement, ces communautés ne sont pas particulièrement utiles si l'analyse du système se limite au niveau des unités de calcul. L'intérêt principal des communautés est qu'elles permettent une *abstraction*, offrant un nouveau niveau d'analyse :

Définition 15 (Hypergraphe des communautés)**.**
Un système d'échange de flux abstrait aux communautés est représenté par un hypergraphe $\langle C, \rightarrow \rangle$ *tel que :*

- C *est l'ensemble des communautés, qui forme une partition de l'ensemble des unités,*

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

- $c_1, c_2, \cdots, c_n \rightarrow c$ *ssi la communauté c acquiert les entrées relatives à sa transformation* c_t *auprès des communautés* $c_1, c_2, \cdots, c_n$ *:*

$$\langle r_1, U_{r1}, t_1 \rangle, \langle r_2, U_{r2}, t_2 \rangle, \cdots, \langle r_n, U_{rn}, t_n \rangle \rightarrow \langle r, U_r, t \rangle \Leftrightarrow t_{in} = \{t_{1out}, t_{2out}, \cdots, t_{nout}\}$$

Cette définition est très proche de celle de l'hypergraphe des unités présenté dans la définition 7 (page 32). En effet, l'objectif de l'usage des communautés est de simplifier le problème d'organisation rencontré au niveau des unités. À titre illustratif, considérons l'ensemble de participants et de requêtes présenté en figure 3.1 : résoudre le problème de manière classique revient à déterminer comment, à partir de l'ensemble des possibles présenté sur la première partie de la figure, on peut arriver à un graphe d'unités valide et respectant les contraintes de capacités telle que celle de la seconde partie.

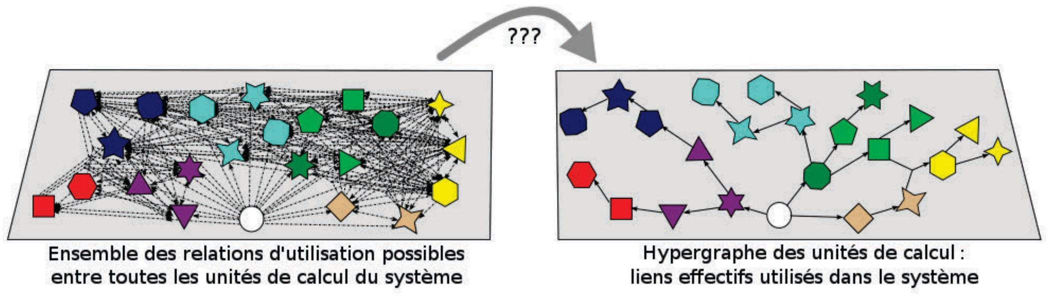


Figure 3.1: Exposé du problème dans le cas général

Aborder ce problème en passant par l'abstraction aux communautés, comme le présente la figure 3.2, permet de diviser ce problème en deux aspects : d'une part, l'exploitation des relations dues à la connexité se fait sur un ensemble des possibles de taille réduite, puisque ne considérant plus qu'un nœud par communauté et non plus un nœud par unité. La résolution de ce problème est illustrée par la transition (b), soit la *sélection des branchements*.

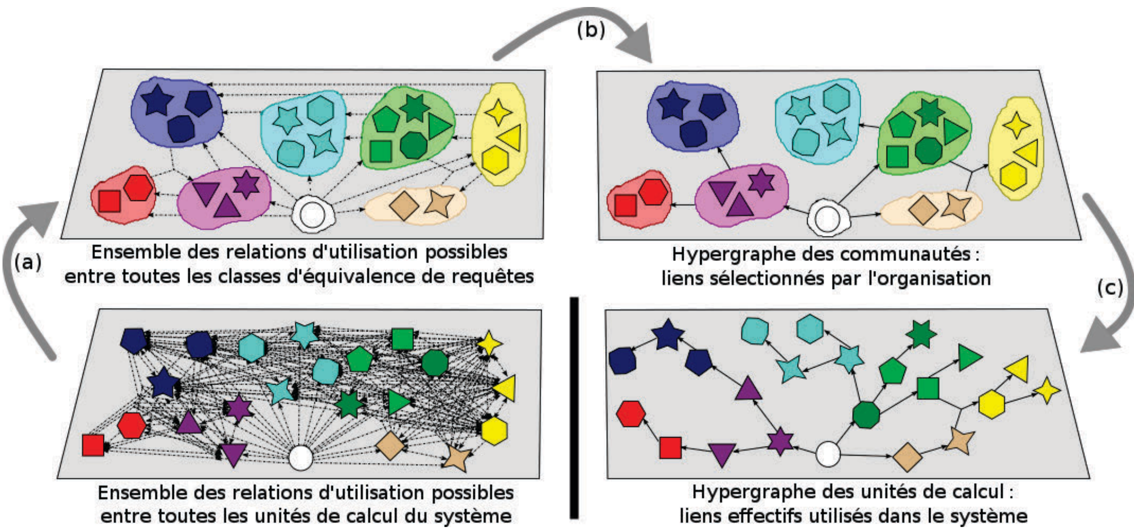


Figure 3.2: Différents niveaux logiques apportés par les communautés

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

D'autre part, la mise en place effective des relations choisies entre requêtes (représentée par la transition (c), soit la *concrétisation*), pour laquelle les contraintes de capacités des participants sont l'aspect principal, se fait en suivant les relations formées par ces communautés : l'organisation d'une communauté donnée peut donc être totalement indépendante des communautés qui ne sont pas reliées à elle. La taille du problème considéré correspond donc à la taille de la communauté étudiée et non plus à celle de l'ensemble du système.

La contrepartie à cette simplification notable est l'apparition d'un nouvel aspect au problème, celui de la *montée en abstraction*, représenté par la transition (a), qui correspond à la nécessité de regrouper entre elles les unités devant constituer une communauté.

Considérant tout système d'échange de flux ayant recours à cette abstraction comme un *système communautaire*, nous pouvons poser que la validité d'un tel système est une spécialisation de la définition 8 (page 33). Cette spécialisation repose sur trois aspects : la validité théorique des relations entre communautés, le fait que chaque communauté récupère effectivement les données requises auprès de la communauté parente, et le fait que tous les participants au sein d'une communauté reçoivent les résultats concernés.

Propriété 2 (Validité d'un système communautaire)**.**
Un système communautaire est valide si et seulement si :

- *Le système abstrait aux communautés* $\langle C, \rightarrow \rangle$ *est valide,*
- *Les connexions entre communautés dans ce système abstrait sont concrétisées au niveau des participants,*
- *Chaque communauté forme un sous-système* $\langle U_r, \rightarrow \rangle$ *lui-même valide.*

En effet, le fait qu'une communauté c donnée forme un sous-système valide signifie que, à partir des sources de données secondaires de ce sous-système (pouvant correspondre à une source primaire de données, si elle fait partie de la communauté, ou à un ensemble de participants collectant les différents flux d'entrée), toutes les unités de U_{c_r} reçoivent les informations en découlant. Si les relations entre communautés sont correctement concrétisées au niveau des participants, et que l'hypergraphe des communautés est lui-même valide, les flux d'entrée éventuellement utilisés par les sources secondaires correspondent bien au flux d'entrée de c_t, et la communauté forme donc un sous-système valide. L'ensemble des communautés formant une partition stricte de l'ensemble des unités, si chaque communauté est bien valide, le système entier l'est également.

De nombreuses possibilités peuvent être envisagées pour mettre en place un système communautaire, sans pour autant relever de l'approche QTor proprement dite. Tout mode d'organisation pouvant être mis en place au niveau de l'hypergraphe des unités peut en effet être appliqué à l'hypergraphe des communautés avec des avantages similaires. L'annexe un de ce document est destinée à présenter quelques aspects généraux du modèle communautaire que nous avons envisagé sans pour autant les reprendre dans notre proposition principale.

3.2.2 Relations entre communautés : réécritures et modèle de coût

Notre approche, visant à diminuer les coûts de fonctionnement, doit intégrer les calculs à l'organisation du système. Dans le but de rendre notre proposition la plus générique possible vis-à-vis du langage de requêtes utilisé, il nous est donc nécessaire de reposer sur une API particulière, que les langages utilisés devront mettre en œuvre.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

L'objet de cette section est de spécifier l'API sur laquelle notre proposition se base, permettant ainsi la mise en place d'un système à torrent de requête pour tout langage fournissant une API compatible. Afin de garantir une meilleure généricité, toutefois, nous avons choisi de ne pas considérer cette API comme monolithique, mais d'autoriser le langage à n'en implémenter qu'une partie, considérant qu'une fonction d'organisation correctement conçue peut tenir compte de la présence ou non des éléments optionnels.

L'élément incontournable que le langage doit nous fournir dans tous les cas pour que notre approche puisse fonctionner est une fonction de **vérification d'équivalence**, nécessaire pour délimiter les communautés. Autrement dit, une fonction qui, étant donné deux requêtes fournies, indique si ces requêtes sont, ou non, mathématiquement équivalentes, au sens de la définition 3 fournie page 30. Nous réutiliserons ici la notation générale de l'équivalence, $r_1 \equiv r_2$.

Selon les langages et les expressions utilisées, vérifier l'équivalence entre deux requêtes peut s'avérer un travail complexe. Aussi, nous n'attendons pas que cette fonction soit nécessairement exhaustive. La condition absolue est qu'elle *ne doit pas produire de faux positifs* (requêtes non-équivalentes mais détectées comme telles), car une erreur de ce type entraînerait l'invalidité du système. Les faux négatifs (requêtes équivalentes non-détectées comme telles) peuvent en revanche exister sans que cela ne soit particulièrement problématique ; mais, naturellement, moins nombreux ils seront, plus performante sera l'organisation du système.

Les relations mathématiques entre requêtes n'évoluant pas dans le temps, nous souhaitons éviter autant que possible de remettre en cause les analyses déjà effectuées précédemment. Aussi nous semble-t-il important que l'organisation du système puisse être paramétrée en fonction de la possibilité, ou non, que la fonction de vérification d'équivalence produise des faux négatifs : s'il est assuré que ceux-ci ne surviendront pas, certaines vérifications peuvent être évitées dans la fonction d'organisation.

Cette fonction de vérification d'équivalence permet de prendre en compte la popularité des requêtes. Pour aller plus loin et tenir également compte de la connexité, nous avons choisi de recourir à un système de **réécriture de requêtes**, selon les principes exposés en section 2.4.3 (pages 48 et suivantes).

Nous attendons donc que nous soit fournie avec le langage une *fonction de réécriture* w chargée de calculer et de renvoyer autant de réécritures que possible d'une requête donnée, à partir d'un ensemble donné de vues. Selon le contexte, des limitations (de nombre de réécritures maximales à calculer, ou de temps d'exécution) devront très probablement être fixées à cette fonction ; mais nous n'en faisons cependant pas un pré-requis à notre mode d'organisation, considérant simplement qu'une fonction de recherche de réécriture trop lente ou trop coûteuse aura des répercussions évidentes sur les performances de la fonction d'organisation qui l'appelle.

Toutefois, afin de garantir de meilleures performances, il nous semple une bonne chose de pouvoir reposer sur la notion de *pertinence* pour filtrer l'ensemble des vues disponibles et ne pas communiquer à la fonction de réécriture un ensemble de vues qui ne pourront pas être utilisées pour réécrire la requête considérée. Nous attendons donc que nous soit fournie, aux côtés de la précédente, une fonction de **vérification de pertinence**. Cette fonction, étant donné deux requêtes passées en paramètres, doit nous indiquer si la seconde peut être utilisée comme vue pour réécrire la première. Nous noterons la relation de pertinence entre deux requêtes par $r_1 \dashrightarrow r_2$.

Comme pour la vérification d'équivalence, il peut arriver que la vérification de pertinence soit difficile à réaliser en pratique. Les conséquences d'une fonction non-exacte sont cependant différentes dans ce cas. En effet, les faux positifs ne causent ici aucun problème

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

de validité, mais ne font que rendre la tâche plus difficile à la fonction de réécriture, qui devra évaluer des vues dont elle ne pourra pas faire usage. À l'inverse, les faux négatifs entraînent la mise à l'écart de vues qui auraient potentiellement pu être utiles, pouvant empêcher de trouver certaines réécritures. Cela n'est pas nécessairement problématique, selon le contexte, mais nous considérons qu'il est préférable de limiter cet effet autant que possible. Dans le cas où aucune information ne serait disponible concernant la pertinence ou l'absence de pertinence d'une requête, nous encourageons donc à ce que la fonction renvoie `vrai`.

Naturellement, ces fonctions peuvent sans difficulté s'appliquer à des situations où seule l'inclusion de requêtes (*containment*) est disponible, sans que cela ne soit aucunement problématique, attendu que nous considérons le concept d'inclusion comme une forme limitée de réécriture. Comme pour ce qui concerne la présence de faux négatifs dans la vérification d'équivalence, nous désirons que l'organisation du système puisse être paramétrée pour indiquer si le mécanisme de réécritures (s'il est présent) se limite ou non à ce cas, ce qui peut servir à simplifier certaines opérations.

La présence d'un mécanisme de réécritures nous fournissant, pour chaque requête concernée, plusieurs réécritures candidates, il est nécessaire de disposer d'un outil permettant de choisir, entre ces différentes possibilités, lesquelles seront les plus intéressantes. Pour cela, nous demandons donc également l'existence d'une *fonction de coût* k, chargée d'associer à chaque requête (réécrite ou non) une estimation du coût de traitement qu'elle représentera. Une telle fonction est usuellement définie d'après un **modèle de coût** qui évalue différents aspects, tels que la complexité des traitements à effectuer, ou la volumétrie des données à considérer (laquelle, dans une configuration distribuée, impacte fortement les coûts de transferts réseau).

Nous avons choisi de ne pas formuler d'hypothèses concernant la constitution de ce modèle de coût : la fonction d'organisation n'a en effet aucun besoin du détail de son fonctionnement, et les choix effectués à ce niveau (favoriser, par exemple, les réécritures utilisant le moins de transferts réseau) peuvent servir à paramétrer la forme du système qui résultera de l'organisation.

Le tableau 3.1 résume les différentes fonctionnalités que nous attendons vis-à-vis du langage et leurs notations d'usage.

Définition	Exemple d'usage	Description / explication
$\equiv : R \times R \mapsto \mathbb{B}$	$r_1 \equiv r_2$	Vérification d'équivalence de r_1 et r_2
$w : R \times \mathcal{P}(R) \mapsto \mathcal{P}(T)$	$w(r, \{v_1, \cdots, v_n\})$	Calcul des réécritures de r
$\dashrightarrow : R \times R \mapsto \mathbb{B}$	$v \dashrightarrow r$	Vérification de pertinence de v pour r
$k : T \mapsto \mathbb{R}^+$	$k(r)$	Estimation du coût de la requête r

Tableau 3.1: API d'évaluation des requêtes dont l'implémentation dépend du langage
Nous utilisons dans ce chapitre la notation francophone $\mathcal{P}(X)$ pour désigner l'ensemble des parties d'un ensemble X, c'est-à-dire l'ensemble constitué de tous les sous-ensembles possibles de X. Dans l'annexe quatre, dont les algorithmes sont en anglais, nous utilisons dans le même sens la notation anglophone 2^X.

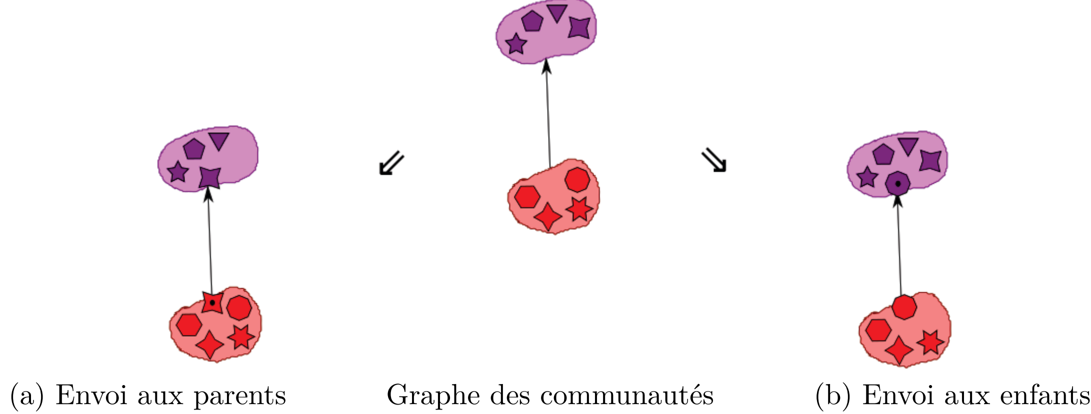
3.2.3 Échange de ressources entre communautés

Les communautés ayant, comme les unités, vocation à être mises en relation entre elles, la question se pose de la façon dont cette mise en relation peut être effectuée. En effet, toute communauté, n'étant constituée que de la somme de ses membres, est *a priori* également sujette aux limitations de capacités. Or, celles-ci doivent être utilisées à la

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

fois en interne, pour permettre aux membres de la communauté de récupérer les résultats requis, et en externe, pour transférer le flux généré par la requête à laquelle est dédiée la communauté aux autres communautés en ayant besoin. Il est donc nécessaire de mettre en place un mécanisme permettant de s'assurer que ces deux types de transferts pourront bien avoir lieu dans tous les cas.

Nous proposons pour cela de mettre en place un mécanisme d'échange de ressources entre communautés, tel que celui décrit en figure 3.3. Le principe est de faire en sorte que l'un des participants de l'une des deux communautés concernées intègre l'autre communauté, et prenne ainsi en charge le transfert de l'une à l'autre. Ce participant mobilise ainsi une partie de ses ressources pour contribuer au fonctionnement de la nouvelle communauté qu'il intègre.



(a) Envoi aux parents Graphe des communautés (b) Envoi aux enfants

Figure 3.3: Deux possibilités d'échange de ressources

Le cas présenté en (a) dans la figure est sans doute le plus simple à envisager : puisque l'une des communautés (désignée comme communauté « enfant ») doit récupérer le flux de l'autre communauté (désignée comme communauté « parente »), et donc consommer une partie de ses ressources, c'est elle qui envoie un participant, qui se charge de compenser cette consommation en fournissant à la communauté parente au moins autant de capacités de diffusions qu'il n'en a utilisé pour servir sa communauté d'origine.

Toutefois, si les capacités de la communauté parente le permet, et que l'un des participants concernés l'accepte, nous envisageons également la possibilité que cet échange de ressources puisse se faire dans l'autre sens : un membre de la communauté parente envoie une unité dans la communauté enfant pour lui diffuser le flux requis, après potentielle transformation, comme présenté en (b). L'échange dans ce sens peut avoir des effets intéressants, notamment sur la latence générale du système.

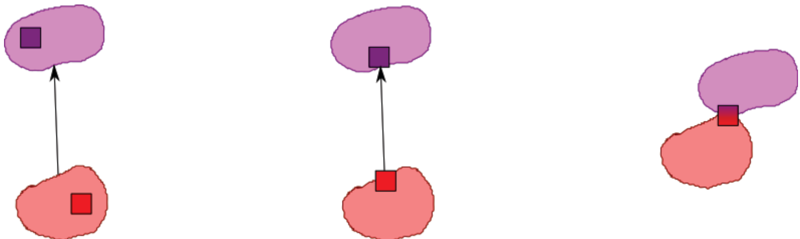
Le choix entre ces deux stratégies, pouvant dépendre d'aspects tels que la volumétrie des flux à échanger (un participant intégrant une communauté enfant et procédant lui-même à certains traitements associés à la transformation concernée peut ainsi avoir moins de donner à transmettre au sein de cette communauté), peut par exemple faire l'objet d'une négociation entre les deux communautés lors du branchement. Dans toutes les figures où cette information est utile, nous adoptons, pour clarifier la lecture, la convention de signaler les unités créées dans le cadre de l'échange de ressources par un point noir central.

L'un des aspects particulièrement intéressants de ce mécanisme d'échange est qu'il permet de s'affranchir de considérer les limitations de capacités lors de l'étude des relations entre communautés. En effet, dès lors que l'échange de ressources de la communauté enfant vers la communauté parente est possible, les capacités de celles-ci, quoique finies,

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

ne sont plus limitées. L'étape de sélection des branchements, constituant l'hypergraphe des communautés, peut donc ne prendre en compte que les aspects logiques (relatifs aux relations entre requêtes et aux coûts qui en découlent), laissant les aspects physiques (connexions effectives des participants tenant compte des limitations de ressources) à l'étape de concrétisation.

Ce mécanisme d'échange présente aussi l'avantage d'augmenter la stabilité de l'ensemble des communautés. En effet, les unités envoyées par les communautés parentes et enfants peuvent maintenir la communauté qu'elles intègrent en vie même en cas de départ des participants ayant créé cette communauté. De ce fait, seules les communautés n'étant plus utiles à aucun descendant sont supprimées de l'hypergraphe, ce qui simplifie considérablement l'organisation en évitant de laisser des communautés orphelines à replacer. Bien sûr, cela n'empêche pas, lorsqu'une communauté n'est plus maintenue que par les participants des communautés qui en dépendent, de relancer des analyses de réorganisations (en tenant compte du coût de maintien en vie de la communauté concernée), mais celles-ci n'ont pas à être effectuées dans l'urgence.



(a) Graphe des communautés (b) Graphe des unités (c) Graphe des participants

Figure 3.4: Échange entre communautés par un seul participant

Une autre conséquence intéressante, comme le montre la figure 3.4, est que les relations présentées dans l'hypergraphe des communautés n'ont, dès lors que ce mécanisme d'échange de ressources est généralisé, plus de correspondance physique : elles sont gérées en interne par le participant concerné, tandis que les seuls échanges réseau sont ceux ayant lieu au sein d'une communauté donnée.

Précisons que ce mécanisme d'échange de ressources n'est pas spécifique à l'utilisation de réécritures pour organiser l'hypergraphe des communautés. Un système communautaire reposant sur un mode d'organisation différent de celui que nous proposons pourrait également tirer profit de cet aspect. C'est la réunion de ces trois éléments qui forme le cœur de notre proposition.

Nous désignons donc comme *système à torrent de requêtes* tout système communautaire dont les relations entre communautés sont déterminées par un système de réécritures et mises en œuvre par un mécanisme d'échange de ressources tel que ceux décrits dans cette section. Il reste maintenant à spécifier comment un tel système peut être mis en place en pratique, définissant par là le fonctionnement de la fonction d'organisation Ω.

3.3 Mise en œuvre incrémentale

Conformément aux principes et objectifs décrits précédemment, nous avons choisi de mettre en œuvre notre approche de manière incrémentale, notre fonction d'organisation Ω réagissant aux différents événements en ajustant au mieux le système existant pour obtenir la configuration la plus performante pour un coût aussi réduit que possible.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Cette section détaille ces mécanismes et algorithmes généraux de notre approche en les séparant en deux aspects : en premier lieu, ceux qui relèvent de la *montée en abstraction*, c'est-à-dire de la manière de déterminer, pour chaque requête exprimée par un participant, dans quelle communauté elle doit s'insérer, ainsi que de la résolution du problème abstrait, c'est-à-dire le fait de déterminer les relations possibles entre ces communautés et de sélectionner celles qui seront utilisées. En second lieu, nous présentons les aspects relatifs à la *concrétisation*, c'est-à-dire à la mise en relation effective des communautés par la mise en relation de certains de leurs participants, ainsi que la mise en relation des participants au sein de chacune des communautés.

La figure 3.5 présente un schéma logique synthétisant les différentes étapes de l'organisation et la façon dont elles s'enchaînent.

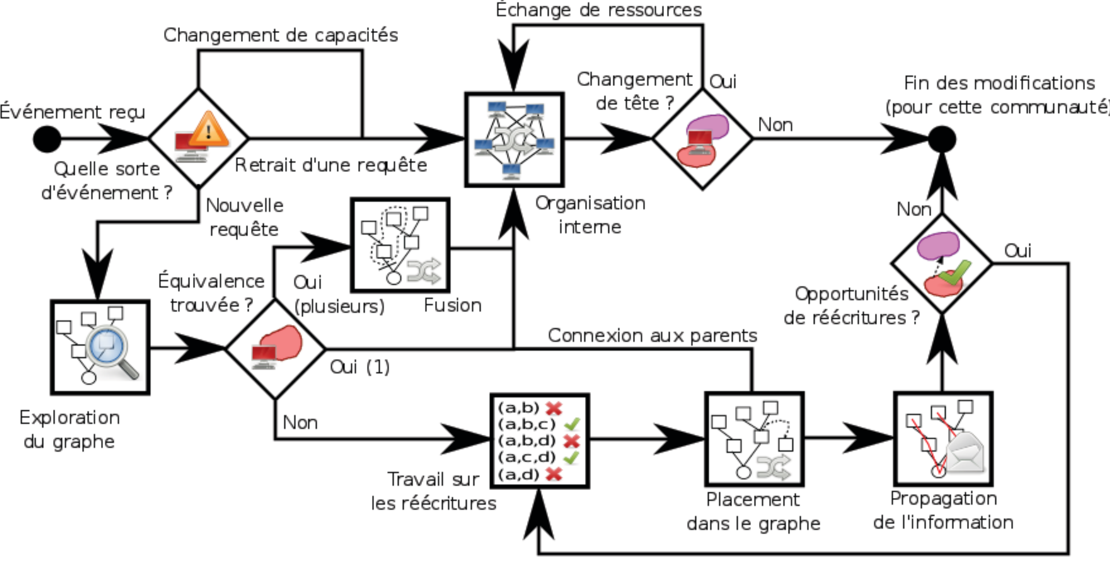


Figure 3.5: Enchaînement des différentes étapes d'organisation

3.3.1 Abstraction : gestion de l'hypergraphe des communautés

La première étape consiste donc en le fait de construire le graphe des communautés au fur et à mesure des événements rencontrés. Comme mentionné précédemment, l'événement déclencheur initial est l'arrivée de la première source de données. Le système est alors constitué d'une unique communauté contenant seulement le participant gérant cette source. Par la suite, de nouvelles requêtes sont exprimées, conduisant à l'apparition de nouvelles communautés.

Notre mécanisme d'échange de ressources nous permet de ne considérer, pour cette partie abstraite de l'organisation, que les ajouts de nouvelles requêtes. En effet, les retraits de requêtes commencent par provoquer des modifications au sein de la communauté concernée ; et ne déclenchent sa disparition que lorsque celle-ci n'a plus ni participants directement intéressés par sa requête, ni ressources envoyées par les communautés en dépendant. La modification du graphe concernée est alors la simple disparition d'une feuille, considérée comme triviale.

C'est donc sur l'arrivée d'une nouvelle requête que nous nous concentrons ici. La première étape pour l'insertion de cette requête est la recherche d'une communauté déjà insérée traitant d'une requête équivalente, ce qui se fait par une exploration du graphe des communautés. Dans le cas où cette exploration ne permet pas de trouver une communauté

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

existante, il est alors nécessaire d'en créer une nouvelle, ce qui peut donner lieu à des réorganisations dues à l'apparition de nouvelles opportunités de réécritures.

Maintien de l'ensemble des communautés

Une requête $r \in R$ est par définition exprimée en fonction d'une ou de plusieurs sources de données connues dans le système. Connaître une source de données suffisant à pouvoir identifier la communauté constituée autour d'elle, ce point de départ permet de parcourir l'ensemble du graphe dès communautés dès lors qu'il est possible de suivre, dans ce dernier, les liens de relations d'une communauté vers une autre. L'algorithme 1, fonctionnant sur le modèle classique du *parcours en largeur*, est conçu dans ce but.

Algorithme 1 Exploration du graphe

1: **Entrées :** une requête q dont les sources sont connues
2: **Sorties :** deux ensembles de communautés C_E (équivalentes) et C_P (pertinentes)
3: **Variables :** un ensemble de communautés C_S et une communauté c
4:
5: C_E et C_P sont initialement vides
6: C_S contient initialement l'ensemble des communautés sources utilisées par q
7: **Tant que** $C_S \neq \emptyset$, **faire :**
8:     **Retirer** une communauté c de C_S
9:     **Si** $c \notin C_P \cup C_E$ **et que** $c_r \dashrightarrow q$, **alors :**
10:         **Si** $c_r \equiv q$, **alors :**
11:             **Insérer** c dans C_E
12:         **Sinon :**
13:             **Insérer** c dans C_P
14:         **Fin si**
15:         **Insérer** tous les successeurs de c dans C_S
16:     **Fin si**
17: **Fin tant que**
18: **Renvoyer** les deux ensembles.

En sortie de cet algorithme, C_E contient l'ensemble des communautés équivalentes détectées. Il peut arriver que cet ensemble contienne plusieurs communautés distinctes, dans le cas où la fonction de vérification d'équivalence fournie est sujette à faux négatifs : cela signifie alors que les deux requêtes c_{1r} et c_{2r} sont équivalentes entre elles (par transitivité), mais n'avaient pas été détectée lors de l'insertion de la seconde (et ainsi de suite s'il y a plus de deux résultats). C'est alors le fait que la nouvelle requête soit exprimée différemment des deux autres qui permet de les identifier. Il convient, dans ce cas, de déclencher une procédure de fusion de communautés. Une telle procédure, dont un exemple est proposé en annexe quatre, consiste essentiellement en le fait de choisir celle des communautés ayant la transformation la plus économique et à lui intégrer l'ensemble des participants (y compris ceux chargés de transférer les données vers d'éventuelles communautés enfants) des autres communautés. De cette manière, les modifications de l'hypergraphe des communautés suivent de manière transparente l'organisation interne des communautés.

Naturellement, si la fonction de vérification d'équivalence ne présente aucun risque de faux négatifs, il est possible d'économiser quelques étapes de l'algorithme 1 en déclenchant une interruption dès qu'une communauté équivalente est trouvée, la recherche de nouvelles vues pertinentes devenant dès lors inutile. Dans tous les cas, dès lors qu'au moins une communauté est identifiée comme équivalente à la nouvelle requête, le participant doit

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

être inséré dans cette communauté. La suite des opérations dépend donc des algorithmes de concrétisation, et est donc traitée dans la prochaine section.

La figure 3.6 présente un exemple de tel parcours : à partir de la source de donnée concernée par la requête, trois communautés enfants peuvent directement être trouvées. Deux d'entre elles traitent de requêtes pertinentes, et l'exploration se poursuit, tandis qu'une autre ne l'est pas, et les communautés qui la suivent ne sont pas explorées. Les étapes suivantes se déroulent de même, jusqu'à finir par trouver la communauté traitant une requête équivalente, lorsqu'il y en a une.

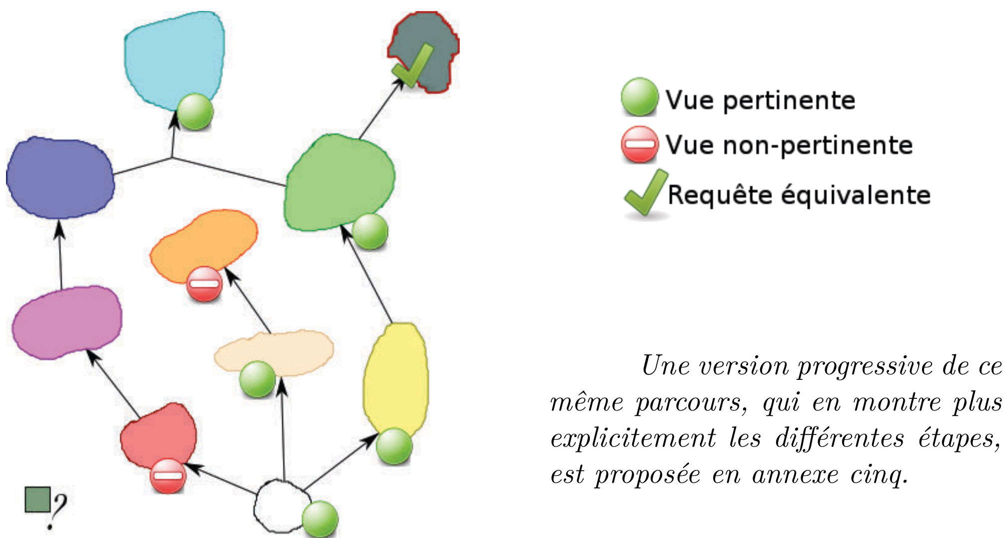


Figure 3.6: Exploration du graphe illustrée

Notons que cet algorithme n'explore pas les descendants des communautés jugées non-pertinentes (l'instruction d'insertion des successeurs dans les sommets à explorer, ligne 15, est à l'intérieur de la conditionnelle ouverte en ligne 9). Si la fonction de vérification de pertinence ne génère pas de faux négatifs, cette optimisation est sans risque, car elle repose sur une propriété de la pertinence déjà présentée en section 2.4.3 (pages 48 et suivantes) : toute vue (ici, communauté) pertinente aura nécessairement au moins un parent lui-même pertinent, comme le montre la figure. De ce fait, explorer les descendants des communautés non-pertinentes n'est pas utile, ceux devant être parcourus pouvant être retrouvés par un autre chemin. Il est cependant possible de désactiver cette optimisation si des faux négatifs sont à craindre, afin d'en limiter la portée. On voit en revanche que ses faux positifs (notamment dans le cas où, cette propriété n'étant pas gérée, toutes les vues sont considérées comme pertinentes) ne conduisent qu'à évaluer plus de communautés que cela n'aurait été strictement requis.

Lorsque, en sortie de l'algorithme 1, l'ensemble C_E est vide, une nouvelle communauté doit être créée, et les requêtes utilisées par les communautés de C_P forment alors l'ensemble à fournir à la fonction de réécritures. Celle-ci fournit alors l'ensemble $w(q, C_P)$ des différentes transformations possible. Comme pour le graphe des unités, les transformations affectées aux communautés correspondent aux hyperarcs du système abstrait, et chaque réécriture correspond donc à un placement possible dans le graphe.

Le choix parmi ces différentes réécritures, et donc entre les différents placements associés, se fait alors par l'utilisation de la fonction de coût, la réécriture r sélectionnée étant celle de coût minimal $(\forall r' \in w(q, C_P), k(r') > k(r))$. En cas d'égalité, il est nécessaire

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

de recourir à une heuristique pour trancher. Des statistiques concernant la volumétrie réelle des flux de sortie, le nombre de capacités disponibles ou la latence attendue peuvent ainsi être utilisées si elles sont disponibles. Ces informations ne sont cependant pas nécessairement fiables, dans la mesure où ces données peuvent être susceptibles de varier grandement au cours de la vie du système.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Modification des relations existant entre communautés

Toute nouvelle communauté dans le système forme une nouvelle vue, susceptible d'être utilisée pour réécrire d'autres requêtes. Aussi est-il nécessaire, une fois la nouvelle communauté connectée (et branchée, ce qui nécessite ici un passage par l'organisation concrète), de vérifier si de nouvelles possibilités de réécritures ne sont pas apparues pour les communautés déjà présentes dans le graphe. En effet, s'il est possible d'obtenir, pour une communauté donnée, une réécriture de coût moindre que celle actuellement sélectionnée, déplacer cette communauté vers une nouvelle position est souhaitable. Cependant, de tels déplacements sont susceptibles, s'ils sont réalisés sans précaution, de faire apparaître des cycles dans le graphe abstrait aux communautés (ce qui n'était pas possible lors de l'ajout de la nouvelle communauté, celle-ci ayant été seule concernée alors qu'elle n'était pas encore connectée).

Il convient donc, tout d'abord, de s'assurer qu'aucune communauté dont la nouvelle venue ne dépend ne va se déplacer. Pour ce faire, un mécanisme empêchant la réciprocité de la relation de pertinence[2] peut sembler à première vue intéressant, mais ses conséquences sur l'évolution à long terme du système pourraient s'avérer problématiques. En effet, cela requerrait soit de devoir réévaluer les pertinences à plusieurs reprises, et donc d'augmenter inutilement le coût de l'organisation, soit de manquer des opportunités de réécritures permettant de diminuer les coûts même après que les modifications successives du graphe aient rendu celles-ci sans danger.

De plus, le déplacement d'un ancêtre de la communauté nouvellement insérée n'est pas le seul cas pour lequel des cycles pourraient se produire : deux déplacements simultanés de communautés liées entre elles, mais sans lien préalable avec la nouvelle venue, pourraient éventuellement avoir eux aussi cet effet.

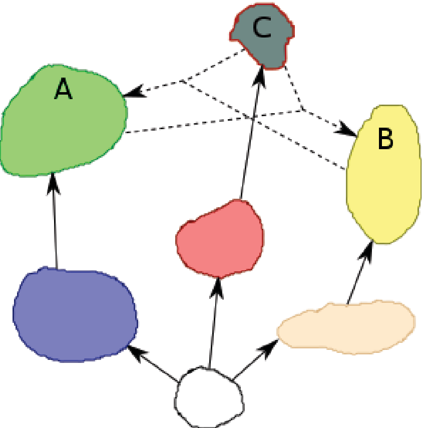


Figure 3.7: Cas d'une réorganisation problématique

La figure 3.7 illustre une telle situation : avant l'insertion de la communauté C, le système ne présentait aucune possibilité de réécriture susceptible de causer de cycles. Le placement choisi pour cette nouvelle communauté ne pose pas de problème en lui-même ; mais les communautés A et B, sans lien direct entre elles dans le graphe, peuvent désormais se réécrire en utilisant la nouvelle venue. La réécriture de A utilise B et C, et la réécriture

[2] Dit autrement, un mécanisme faisant que, si la vue B a déterminé, lors de sa recherche de placement, que la vue A, préalablement insérée, était pertinente, alors la vue A ne peut plus considérer la vue B comme pertinente.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

de B utilise C et A : un déplacement simultané de ces deux communautés les ferait donc toutes deux attendre les données envoyées par l'autre, ce qui est à éviter absolument.

Pour éviter ce cas de figure, nous avons choisi de procéder par modifications successives : un seul hyperarc du graphe des communautés est modifié à la fois. Le système peut ainsi s'assurer qu'il n'existe aucun chemin partant de la communauté déplacée (nœud sortant de l'hyperarc, toujours unique) vers l'une des communautés qui deviennent ses nouveaux parents (nœuds entrants de l'hyperarc). Les déplacements étant successifs, il n'y a à chaque fois qu'à tenir compte du nouvel état de l'arbre.

Un possible inconvénient de cette approche est que l'état du système dépend hautement de l'ordre dans lequel les opérations sont effectuées. Si des informations sont disponibles concernant l'importance du déplacement des différentes communautés, elles peuvent cependant être utilisées pour ordonner ces opérations ; dans le cas contraire, il n'est de toute façon pas possible de déterminer quel ordre sera le meilleur. Une question à se poser est la stabilité d'un tel système : si les déplacements sont effectués un par un, il pourrait être possible que l'un d'entre eux annule les effets d'un mouvement précédent, ce qui pourrait entraîner un mouvement d'oscillation qui nuirait beaucoup au système. Il est toutefois simple d'éviter ce risque en n'autorisant les déplacements que s'ils diminuent strictement le coût de calcul pour la communauté concernée.

Théorème 1 (Stabilisation)**.**
La phase de réorganisation d'un système à torrent de requêtes se termine en un nombre fini d'étapes, amenant le système à se stabiliser.

Preuve. Le coût associé à un placement dans le graphe est déterminé par le modèle de coût en fonction de la réécriture correspondante. Ce coût ne dépend que de cette réécriture, et est indépendant de la façon dont les vues concernées acquièrent leurs résultats. De ce fait, aucun déplacement tiers ne peut affecter le coût d'une communauté donnée, qui ne peut varier que si cette communauté-ci se déplace. Or, l'hypergraphe étant de taille finie, ces déplacements sont nécessairement en nombre limité. De ce fait, si une communauté se déplace chaque fois qu'une réécriture disponible est de coût strictement inférieur à celui de sa réécriture précédente et uniquement dans ce cas, il arrive nécessairement un moment où l'ensemble des réécritures sélectionnables pour chaque communauté devient vide. Cet état ne sera remis en cause que par une modification extérieure, telle que l'apparition d'une nouvelle communauté permettant de nouveaux déplacements.

Un point à préciser ici concerne la façon dont les nouvelles réécritures sont déterminées.

Cela ne pose en effet aucun souci dans les cas où le mécanisme de réécritures se limite à des relations d'inclusion de requêtes, dans la mesure où chaque réécriture n'utilise qu'une unique vue. Il suffit donc, lorsque la vue nouvellement insérée est jugée pertinente (donc si elle *contient* la requête à réécrire), de calculer la réécriture correspondante (qui, d'un point de vue réseau, sera nécessairement unique), ne dépendant d'aucune autre vue, et de comparer son coût à celui de la réécriture actuellement sélectionnée.

Dans les cas de systèmes de réécritures plus complexes, en revanche, où une réécriture donnée peut utiliser un nombre variable de vues, il est nécessaire de fournir à la fonction de réécritures, en plus de la vue nouvellement insérée, l'ensemble des vues pertinentes préalablement identifiées, celles-ci étant susceptibles d'être réutilisées. Il est donc nécessaire soit de mettre en place un historique de ces vues, qui devra être mis à jour (toute communauté qui disparaîtrait en cours de vie du système devrait ainsi en être retirée), soit de réaliser un nouveau parcours du graphe pour identifier de nouveau ces communautés. Cette seconde possibilité nous apparaît plus coûteuse (elle conduit à réévaluer périodique-

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

ment des relations de pertinence immuables), mais peut être préférée selon les cas (en fonction, notamment, de la facilité à maintenir l'historique pour l'autre possibilité).

Si le système de réécritures permet une telle chose, il peut être judicieux, plutôt que de calculer de nouveau toutes les réécritures possibles pour l'ensemble des vues (ce qui conduit, là encore, à reproduire des calculs déjà effectués), de ne chercher que les réécritures utilisant la nouvelle vue, et de vérifier si l'une d'elles est moins coûteuse que la réécriture actuellement sélectionnée.

Distribution de l'organisation abstraite

L'algorithme 1, formant la base de l'organisation abstraite, est conçu de telle sorte qu'il puisse être utilisé aussi bien par un *tracker* centralisé que, dans le cas d'une organisation distribuée, par un participant souhaitant s'insérer de manière autonome. En effet, la seule connaissance préalable qu'il requiert est celle des communautés d'origines, qui sont obtenues à partir de la requête exprimée par le participant. Si un *tracker* centralisé maintient vraisemblablement une représentation locale de ce graphe sur laquelle effectuer de telles recherches, rien n'empêche d'effectuer le même parcours en interrogeant directement chaque communauté par le réseau. Il est nécessaire, pour cela, que chaque communauté dispose d'au moins un participant identifié comme pouvant être contacté, qui soit capable d'indiquer la requête sur laquelle travaille la communauté, ainsi que les différentes communautés (et les pairs à contacter pour les interroger) qui dépendent directement des résultats produits par cette communauté. De ce fait, cette première étape du placement peut sans difficulté se faire sans l'aide d'une autorité unique, et ainsi quitter la classe 1 de la classification Bortzmeyer[12]. Le classement effectif dépendra ensuite de la façon dont s'effectue la concrétisation.

Le mécanisme de réorganisation du graphe des communautés peut lui aussi quitter aisément le cadre d'une autorité unique. En effet, chaque communauté peut maintenir l'historique des vues pertinentes considérées (ou procéder à de nouvelles explorations du graphe), et par là, décider de manière autonome quelle position dans le graphe lui convient le mieux (de la même manière qu'un participant n'ayant trouvé aucune communauté traitant d'une requête équivalente à la sienne peut, dans ce contexte, décider de manière autonome de l'endroit où connecter la nouvelle communauté qu'il crée). Nous considérons cependant préférable que cette décision, quoiqu'autonome, soit prise d'après les mêmes critères pour l'ensemble des communautés, c'est-à-dire relève d'un modèle de coût commun fourni avec le langage, comme c'est le cas dans une organisation centralisée, n'ayant pas encore étudié les problématiques qui résulteraient de l'usage de critères différents.

La décision de déplacer, ou non, une communauté résultant de l'apparition d'une nouvelle vue, il est nécessaire que les différentes communautés susceptibles de se déplacer soient informées de l'apparition d'une nouvelle vue. Pour cela, plusieurs mécanismes peuvent être envisagées, tels que la propagation d'un signal au sein du graphe (la nouvelle communauté prévient les sources qu'elle utilise de son arrivée, et chaque communauté transmet l'information à ses descendants directs) ou l'existence d'un mécanisme de type Pub/Sub, auquel chaque communauté serait abonnée, dans lequel seraient publiées toutes les modifications du graphe.

Une difficulté à ce niveau est cependant d'éviter les déplacements simultanés, afin de s'assurer de ne pas créer de cycles. Ceci peut être géré aisément lorsque l'organisation est le fait d'un *tracker* centralisé, qui examine les différentes possibilités de manière séquentielle ; mais présente davantage de difficultés pour des communautés autonomes pouvant évaluer les possibilités de déplacement au même moment. De ce fait, il est nécessaire d'introduire,

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

dans une organisation distribuée, un mécanisme d'exclusion mutuelle sur les déplacements. L'état actuel de nos travaux nous amène donc à considérer cet aspect comme relevant d'une hiérarchie des autorités, constitutive de la classe 2 : l'autorité principale reste en charge de spécifier l'ordre dans lequel les communautés sont autorisées à ce déplacer, même si chaque communauté peut décider de ses déplacements de manière autonome.

3.3.2 Concrétisation : relations entre les participants

Chaque communauté est donc constituée d'un groupement d'unités associées à une requête donnée (aux expressions équivalentes près), et dispose, pour obtenir ses résultats, d'une transformation (requête réécrite) fournie par l'organisation abstraite. Le rôle de l'organisation communautaire est alors de concrétiser cette organisation en mettant en relation les participants concernés.

Pour ce faire, trois tâches doivent être effectuées. Tout d'abord, il est nécessaire d'*acquérir* les flux de données produits par les communautés parentes. À partir de ces flux d'entrée, il devient alors possible de *calculer* les résultats de la requête. Ces résultats doivent alors être *diffusés* à l'ensemble des participants de la communauté, ainsi qu'aux autres communautés devant les acquérir.

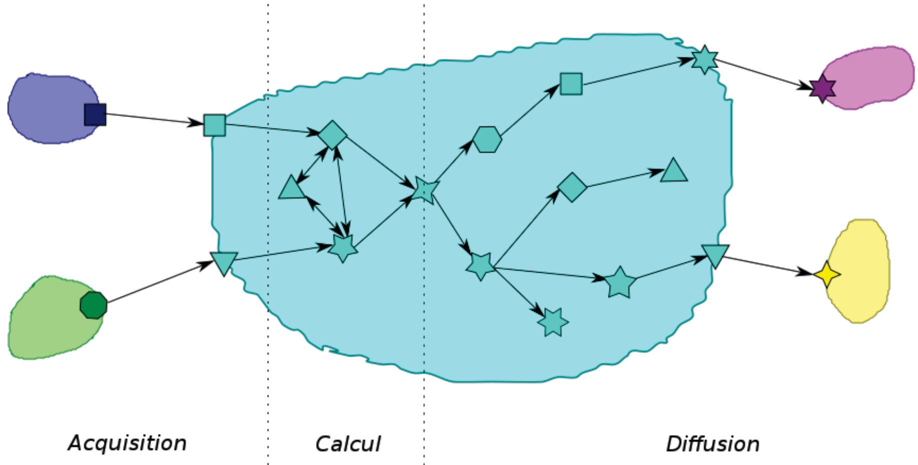


Figure 3.8: Possible répartition des tâches au sein d'une communauté

De très nombreuses organisations sont envisageables pour réaliser ces différentes tâches au sein d'une communauté. La figure 3.8, non décrite ici, présente ainsi un possible cas d'organisation complexe pouvant être envisagé dans le cas général d'un système communautaire (pas de mécanisme d'échange, diverses unités spécialisées). Nous préférons toutefois nous appuyer ici une version plus simple, dans laquelle, notamment, un seul participant gère simultanément les tâches d'acquisition et de calcul. L'organisation distribuée du calcul d'une requête par un ensemble de participants de taille variable est en effet un problème à part entière, auquel sont consacrés des travaux en cours[54] parallèles à ceux présentés dans ce document.

Nous présentons ici les mécanismes liés à la mise en place de l'acquisition des données, et donc à la mise en relation d'une communauté avec ses parents ; puis ceux permettant la diffusion des résultats obtenus au sein de la communauté. L'étude du cas général, plus complexe et contenant notamment une description de la situation présentée en figure 3.8, est proposée en annexe un.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Mise en relation des communautés

Conformément à notre cadre de travail, la figure 3.9 reprend la situation de la communauté présentée en figure 3.8 (mêmes parents, mêmes participants, mêmes enfants), mais sans présenter de calcul réparti.

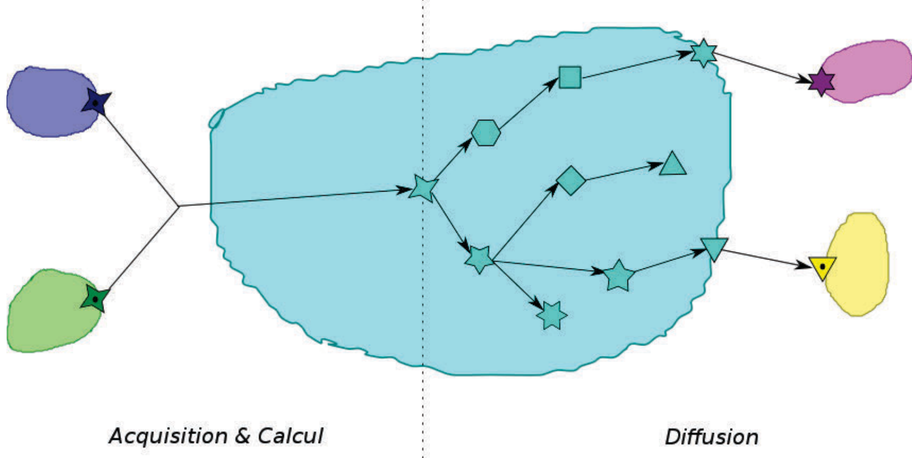


Figure 3.9: Autre répartition des tâches, sans calcul distribué

Une différence notable entre les figures 3.8 et 3.9 est le fait que, dans la seconde, le mécanisme d'échange de ressources proposé en section 3.2.3 est dûment mis en place : pour chaque relation entre communautés, un même participant gère les transferts des deux côtés. Aucune des deux unités du participant gérant l'échange avec la communauté violette n'est cependant « pointée », dans la mesure où ce participant, comme le montrait la première figure, est présent dans chacune de ces deux communautés sans l'intervention de ce mécanisme. Ce cas, quoique rare, est néanmoins envisageable du fait de la connexité des requêtes, et doit donc être pris en compte.

Qu'elle soit originaire de la communauté considérée ou qu'elle résulte d'un envoi de ressources de la part d'une communauté parente, l'unité chargée d'acquérir les données et de calculer les résultats pour le compte de la communauté enfante se trouve dans une position particulière, que nous appelons *tête de communauté* :

Définition 16 (Tête de communauté)**.**
Dans un système QTor sans distribution de traitement au sein d'une même communauté, une unité particulière, désignée comme tête de communauté est chargée

- *d'acquérir si besoin les données nécessaires au calcul de la requête, par l'intermédiaire d'autres unités gérées par le participant à qui elle appartient, qui sont membres des communautés parentes,*
- *de procéder seule au calcul des résultats de la requête,*
- *de participer à la tâche de diffusion desdits résultats, étant considérée comme une source (secondaire) de données.*

Naturellement, les sources primaires de données sont toujours en position de tête dans leurs propres communautés (celles-ci étant les seules du système à n'avoir aucun parent, et donc aucun besoin d'acquérir d'informations auprès de ces parents). Pour les communautés ne correspondant pas à des sources, en revanche, le choix de cette unité particulière est une étape importante.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Algorithme 2 Sélection de la tête de communauté

```
1:  Appliqué par : une communauté c
2:  Entrées : une requête réécrite r affectée à la communauté
              (dont l'ensemble des parents est connu)
3:  Sorties : une unité h pour servir de tête de communauté
4:  Variables : une unité u, une communauté parente c_p,
5:                un ensemble d'unités U initialement vide.
6:
7:  Pour chaque communauté c_p parmi les parents de r, faire :
8:      Envoyer à c_p une demande d'aide
9:  Fin pour
10: Tant que toutes les communautés parentes n'ont pas répondu, faire :
11:     Attendre la réponse d'une prochaine communauté c_p
12:     Pour chaque unité u proposée par c_p, faire :
13:         Si u est acceptée par la communauté, alors :
14:             Insérer u dans U
15:         Fin si
16:     Fin pour
17: Fin tant que
18: Pour chaque unité u de c, faire :
19:     Si u est capable de traiter r, alors :
20:         Insérer u dans U
21:     Fin si
22: Fin pour
23: Si U n'est pas vide, alors :
24:     Marquer c comme active
25:     Choisir une unité h dans U
26:     Si l'unité h provenait d'une communauté parent, alors :
27:         Insérer h dans c
28:     Fin si
29:     Pour chaque unité u de U autre que h, faire :
30:         Si l'unité u provenait d'une communauté parente, alors :
31:             Indiquer au participant concerné qu'il peut libérer les ressources de u
32:         Fin si
33:     Fin pour
34:     Pour chaque communauté c_p parmi les parents de r, faire :
35:         Si le participant gérant h n'a pas d'unité dans c_p, alors :
36:             Insérer une nouvelle unité de ce participant dans c_p
37:         Fin si
38:     Fin pour
39: Sinon :
40:     Marquer c comme inactive
41:     Interrompre
42: Fin si
43: Renvoyer h
```

La routine de choix utilisée ligne 25 repose sur trois facteurs : les capacités du participant, le taux de présence dans les communautés parentes et la distance à la source. Naturellement, les communautés parentes ne proposent l'aide que de participants volontaires et ayant les ressources suffisantes.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Lors de chaque déplacement d'une communauté (comprenant la première insertion lors de la création de cette communauté), il convient donc, pour pouvoir prendre en compte la possibilité d'échange de ressources dans les deux sens, de consulter également les participants des communautés parentes qui auraient des capacités suffisantes pour traiter la réécriture concernée, et accepteraient de le faire. L'algorithme 2 présente la démarche que nous proposons pour effectuer cette consultation et sélectionner le meilleur candidat à la position de tête de communauté.

Notons que cet algorithme prévoit la possibilité, pour la communauté c devant sélectionner une nouvelle tête, de refuser l'aide de certains participants. Ce cas n'est pour l'instant envisagé que de manière théorique (en pratique, nos implémentations actuelles acceptent systématiquement toute aide proposée), mais permettrait de prendre en compte des critères liés l'intimité numérique des participants ou à d'éventuelles restrictions concernant l'accès aux données.

Par ailleurs, l'algorithme prévoit également la possibilité qu'aucune des unités considérées ne dispose de capacités suffisantes pour prendre la position de tête de communauté. Cela peut malheureusement se produire lorsque les communautés parentes n'ont pas les moyens de proposer de l'aide, et que les ressources des unités de la communauté enfant, très limitées, ne permettent pas de traiter la requête, malgré les réécritures. Dans ce cas, les résultats de la communauté ne pourront pas être correctement calculés. La communauté peut alors soit être mise en attente, soit être connectée par le biais d'un participant dont, faute de mieux, les capacités seront outrepassées (pour ce dernier cas, on choisira systématiquement un participant de la communauté enfant).

Cette situation, très problématique, mérite d'être étudiée spécifiquement. La configuration la plus simple à envisager est que les capacités insuffisantes sont ici celles de calcul, puisqu'il s'agit de procéder au traitement de la requête. Il convient dans ce cas de noter la réécriture affectée à la communauté a été choisie parce qu'elle présentait le coût le moins élevé parmi les différentes possibilités connues. Si aucun participant dans la communauté n'est capable de traiter cette requête dans cette forme, alors aucune autre autre disposition ne serait susceptible de leur permettre de traiter cette requête sans répartition des calculs.

Un effet de bord notable de notre mécanisme d'échange de ressources, dans le cas où il est géré par un seul participant, est cependant que les capacités de diffusion peuvent également être bloquantes dans ce cas. En effet, même si plusieurs participants disposent de capacités de calcul suffisantes, il peut arriver, pour des réécritures utilisant un nombre important de vues, qu'aucun d'entre eux ne soit en mesure d'offrir un lien de diffusion à chaque communauté concernée. Cette configuration est cependant assez peu probable en cas réaliste, dans la mesure où, même en capacités réduites, le nombre de vues mobilisées par une requête donnée est généralement limité, et dépasse rarement les capacités de diffusion d'un participant ordinaire.

Dans un cas comme dans l'autre, toutefois, répartir les tâches d'acquisition et de calcul entre différents participants est susceptible de lever ces difficultés. Comme mentionné précédemment, cette répartition est l'objet de travaux en cours[54], parallèles à ceux proposés dans ce document. En leur absence, il est également possible d'envisager une autre manière d'aborder ce problème : il est en effet possible qu'un participant situé dans une autre communauté sans rapport direct avec celle concernée soit en mesure de réaliser ces calculs. C'est pour cette raison que nous proposons de marquer les communautés comme actives ou inactives : utilisant le mécanisme de propagation d'information mentionné dans la section précédente (signal ou publication des modifications du graphe), la communauté inactive peut transmettre un appel à l'aide au reste du système. Des participants altruistes et suffisamment puissants peuvent alors décider d'intégrer cette communauté pour

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

lui permettre de fonctionner correctement.

Enfin, précisons que la tête de communauté peut être modifiée en cours « à la volée » sans qu'il ne soit nécessaire de recourir à l'algorithme 2. En effet, lors de l'arrivée d'un nouveau participant dans la communauté, une simple comparaison entre la tête actuelle et ce nouveau venu permet de déterminer s'il peut être intéressant de remplacer la première par le second, sans qu'il n'y ait pour cela besoin de consulter l'ensemble des autres participants concernés. En cas de départ de la tête de communauté, en revanche, il est préférable de relancer l'algorithme.

Concrétisation de la diffusion des résultats

Une fois les résultats obtenus par la tête de communauté, ceux-ci doivent être transmis à l'ensemble des autres participants. Pour cela, n'importe quelle organisation de type système à miroir peut être mise en place, considérant la tête de communauté comme la source du sous-système que constitue notre communauté. Le but de ces organisations est en effet de permettre à chacun des participants concernés d'obtenir le flux initial propagé par sa source, ce qui correspond ici au flux de résultats.

Afin de conserver l'organisation physique aussi simple que possible, nous proposons pour cet aspect la mise en place d'un arbre de diffusion, dont la tête de communauté est la racine, et qui contient l'ensemble des unités de la communauté. Une problématique est ici de maintenir un arbre aussi peu profond que possible, afin de limiter la latence ; tout en évitant les débranchements et rebranchements entre participants n'apportant aucune amélioration. Pour cela, nous proposons l'algorithme 3 :

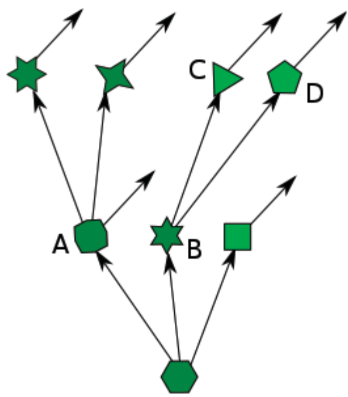
Algorithme 3 Organisation de l'arbre de diffusion

1: **Appliqué par :** une communauté c, ayant pour tête l'unité h
2: **Entrées :** la liste U des unités, ordonnées selon les capacités des participants
3: **Sorties :** [ne renvoie aucune valeur, mais connecte les unités entre elles]
4: **Variables :** un entier naturel n
5:
6: **Retirer** l'unité h de U
7: n reçoit les capacités utilisables de h
8: **Tant que** U n'est pas vide, **faire :**
9:     **Retirer** de U les n premières unités
10:     **Connecter** ces unités à la fin de l'arbre, sans briser de liens si possible
11:     n reçoit la somme des capacités utilisables de ces unités
12: **Fin tant que**

Cet algorithme est ici très simplifié, une version plus détaillée étant proposée en annexe quatre. Son mode de fonctionnement est d'organiser l'arbre « par étages », permettant de ne pas déconnecter les unités de leur parent précédent lorsque leurs positions relatives restent inchangées, c'est-à-dire lorsqu'elles à une distance d'un seul étage (qu'aucune unité n'ait bougé ou que les deux aient suivi le même mouvement).

La figure 3.10 propose une illustration du résultat de cette organisation en présentant le cas d'une communauté de huit participants dont les capacités de connexions vont de 3 à 1. Un point notable sur cette figure est la situation des nœuds `C` et `D` : appliquer l'algorithme sans tenir compte des branchements précédents aurait conduit à déplacer l'une de ces unités vers le nœud `A`, qui dispose encore d'une connexion sortante. Toutefois, le gain de cette opération aurait été nul, le nœud `B` auquel ils sont raccordés étant à la même profondeur dans l'arbre. Cette opération inutile est donc évitée.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés



Une version progressive de cette organisation, qui en montre plus explicitement les différentes étapes, est proposée en annexe cinq.

Figure 3.10: Organisation de l'arbre de diffusion

Précisions que la notion de « capacités utilisables » pour une unité donnée renvoie ici au fait que certains participants (en particulier ceux dont au moins une unité est en situation de tête) intègrent plusieurs communautés et doivent donc partager leurs ressources entre celles-ci. L'algorithme (qui peut, pour cela, interroger le participant ou bénéficier des connaissances d'un *tracker* commun à toutes les communautés) considère donc qu'une portion des capacités totales du participant sont réservées et ne peuvent pas être utilisées pour le déploiement de cet arbre interne. Les participants peuvent donc, si le besoin s'en fait sentir, être exploités au maximum de leurs capacités disponibles, sans toutefois remettre en cause les liens qu'ils auraient déjà réservés à d'autres communautés.

Ordonner les unités en fonction des capacités totales (plutôt que des capacités utilisables) est une heuristique permettant de garantir un ordre stable, qui sera commun à l'ensemble des communautés sans nécessiter de concertation. Cela garantit donc que les positions relatives des participants présents (exception faite de la tête de communauté, placée en racine de l'arbre même si d'autres participants sont plus puissants) seront les mêmes dans toutes les communautés qu'ils intégreront, à la manière de ce qui est mis en place dans Ripple[94].

Distribution de l'organisation concrète

L'algorithme 2 de sélection de la tête de communauté est conçu de sorte à pouvoir être appliqué directement par une communauté donnée, de manière autonome. C'est pour cette raison, notamment, que l'étape de négociation se fait par l'envoi d'une demande d'aide aux communautés parentes et l'attente des réponses de celles-ci : pour un *tracker* centralisé, autorité unique de décision maintenant en local une représentation du système, cette demande d'aide se traduit par une vérification directe des capacités des nœuds concernés, déjà connus (ceux-ci doivent néanmoins être interrogés pour garantir le respect de leurs intérêts). Cet algorithme est donc adapté à un fonctionnement de classe 2 dans la classification Bortzmeyer[12], qui est également la classe actuellement envisagée pour la version répartie de notre organisation abstraite. Un tel algorithme pourrait cependant être modifié pour un fonctionnement de classe 3, par exemple en remplaçant la routine de choix par un mécanisme d'élection[91] par les membres de la communauté.

L'algorithme 3 d'organisation de l'arbre de diffusion, pour sa part, repose en l'état sur un fonctionnement à autorité unique : aucune latitude n'est laissée aux participants vis-à-vis de leur placement dans l'arbre. Il est donc de classe 1, quoique son impact localisé

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

le rende compatible avec un système général de classe 2 sans remettre celle-ci en cause. Toutefois, n'importe quelle organisation de type système à miroirs peut être employée à ce niveau, la seule contrainte étant de permettre à l'ensemble des participants d'obtenir les résultats calculés par la tête de communauté. Il est donc possible de remplacer cet algorithme par tout autre mode organisation jugé plus adapté à la situation envisagée, notamment l'absence d'un *tracker* centralisant l'organisation (plusieurs systèmes à miroir ont un fonctionnement de classe 3). Notons par ailleurs que rien n'oblige les différentes communautés du système à adopter la même organisation interne, et qu'il est tout à fait possible d'envisager que plusieurs communautés autonomes aient recours à des solutions différentes pour l'organisation de leur diffusion.

3.4 Propriétés et analyse

Ayant défini le processus d'organisation notre proposition, nous pouvons désormais étudier les propriétés qu'il est possible de mettre en lumière au niveau théorique ; avant de procéder à une évaluation expérimentale qui sera l'objet du prochain chapitre.

Cette section présente quatre aspects principaux : une étude des coûts des différentes parties de l'organisation, une étude de l'impact de notre organisation sur les charges de calcul des participants, une description de la façon dont cette organisation s'adapte à l'absence d'un mécanisme de réécritures sur le langage de requêtes, puis une analyse du niveau d'adaptabilité de notre proposition, relatif notamment au mécanisme d'échange de ressources entre les communautés.

3.4.1 Étude des coûts de l'organisation

Comme le montrait la figure 3.5 (page 72), l'organisation du système dépend d'un certain nombre d'étapes n'étant pas nécessairement identiques d'une itération à l'autre de la fonction d'organisation. Évaluer le coût de cette dernière demande donc de distinguer les différents cas pouvant être rencontrés.

Les événements correspondant au changement de capacités d'un participant ou au retrait d'une requête conduisent immédiatement à une modification interne aux communautés concernées. Dans le cas favorable, aucune tête de communauté n'est alors remise en cause, et le seul coût à prendre en compte est ici celui de la modification de l'arbre de diffusion interne. L'algorithme proposé à cet effet (algorithme 3, page 83) consiste en un simple parcours de liste ordonnée, de complexité $O(|c|)$ pour une communauté c donnée. Cette liste ordonnée peut être soit déterminée à chaque itération à partir de l'ensemble des participants (la complexité est alors celle d'un algorithme de tri, que l'on peut envisager en $O(|c|log|c|)$, sans mémoire), soit mémorisée au cours de la vie du système (auquel cas il suffit de retirer le participant de cette liste ou de l'y repositionner, pour une complexité en temps de $O(|c|)$, mais qui nécessite un espace de stockage).

Si un tel événement entraîne un changement de tête de communauté, en revanche, il est nécessaire de passer par l'algorithme de choix de tête de communauté (algorithme 2, page 81). Cet algorithme nécessite pour sa part de comparer l'ensemble des candidats potentiels, c'est-à-dire l'ensemble des participants de la communauté enfant et celui de chacune des communautés parents (qu'il est nécessaire d'évaluer même si aucun d'entre eux n'est susceptible de fournir les ressources requises). La complexité de cet algorithme est donc de l'ordre de $O(|c| + |c_1| + \cdots + |c_n|)$, avec $c_1, \cdots, c_n$ les différentes communautés parentes.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

De plus, l'insertion des unités créées par la nouvelle tête de communauté dans le cadre de l'échange de ressources demande de relancer une organisation de la diffusion dans les communautés concernées. Le pire des cas est donc ici celui où les changements de proche en proche conduisent à réévaluer l'ensemble du système, communauté par communauté. Ce cas est cependant très peu vraisemblable en pratique, du fait notamment que les participants déjà intégrés dans la plupart des communautés sont favorisés pour le rôle de tête, évitant par là de modifier ces communautés.

Les insertions de nouvelles requêtes, pour leur part, nécessitent de procéder à des recherches dans le graphe abstrait, et potentiellement à des travaux sur les réécritures. L'algorithme que nous proposons pour la recherche de communautés (algorithme 1, page 73) est un algorithme de parcours en largeur avec élagage. La complexité d'un tel algorithme est connue[28] comme étant dans le pire des cas en $O(n+m)$, où n est le nombre de sommets considérés et m le nombre d'arcs. Le graphe exploré étant l'hypergraphe des communautés, n correspond ici au nombre de communautés dans le système ($n = |C|$). Le nombre m de sommets est plus dur à évaluer, dans la mesure où, si chaque communauté (hormis celles dédiées aux sources de données) présente un degré entrant de un, les réécritures mobilisant plusieurs vues augmentent le nombre d'arcs sortants dans le graphe, qui est celui pris en compte ici. De ce fait, dès lors que les relations utilisées peuvent dépasser le cadre de la stricte inclusion de requêtes, nous aurons $m \geq |C| - |S|$.

Dans le cas où une équivalence est trouvée, l'organisation se ramène alors aux étapes de concrétisations, déjà décrites. Le participant est alors inséré dans la communauté concernée sans nécessiter de déclencher de choix de tête parmi l'ensemble des candidats potentiels (mais sa comparaison avec la tête de communauté déjà en place peut tout de même entraîner un remplacement, et donc des modifications sur les communautés voisines). L'insertion d'un participant dans une communauté déjà existante nous semblant être un événement se produisant fréquemment dans une situation réaliste, la complexité d'une itération de la fonction Ω sera donc fréquemment de l'ordre de $O(|C| + |c|)$, ce qui, dans ce cas réaliste, présente une réduction appréciable de la taille du problème (dès lors que plusieurs participants expriment des requêtes équivalentes, $|C| + |c| < |P|$).

Dans le cas de la création d'une nouvelle communauté, l'organisation abstraite se poursuit par la recherche de réécritures et la sélection du meilleur résultat. Comme cela a déjà été mentionné, la complexité de la première varie en fonction du langage utilisé et des modalités de fonctionnement (pour un système de réécritures basé sur l'inclusion de requête, par exemple, la complexité sera vraisemblablement de l'ordre de $O(n)$, avec n le nombre de vues à considérer, du fait qu'il n'est pas nécessaire de former des combinaisons de plusieurs vues). Le gain à ce niveau, par comparaison aux approches n'utilisant pas de groupements en communautés, sera donc d'autant plus sensible que l'ensemble C est petit par rapport à l'ensemble P. Nous pouvons par ailleurs remarquer que, hors cas de faux négatifs de la fonction de vérification d'équivalence, les vues fournies à la fonction de réécritures seront toutes différentes, ce qui assure d'éviter certains traitements inutiles du type de tenter de combiner plusieurs vues identiques.

La sélection du meilleur résultat, en revanche, reste de l'ordre de $O(n)$, avec n le nombre de réécritures fournies par la fonction w. Celui-ci peut être théoriquement réduit par la diminution du nombre de vues redondantes ; mais, en pratique, dès lors que la connexité des requêtes entraîne un nombre important de possibilités de réécritures, les limitations imposées à cette fonction w seront le facteur important à ce niveau.

La complexité de la propagation de la notification d'arrivée d'une nouvelle vue dépend du mode de fonctionnement utilisé, mais c'est de la recherche de nouvelles réécritures pour chacune des communautés susceptibles de se déplacer que vient l'essentiel de la complexité

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

des réorganisations. Le meilleur des cas est ici celui où la communauté nouvellement insérée n'est pertinente pour aucune autre requête ; le pire celui où toutes les communautés préalablement insérées (exception faite des communautés sources) s'interrogent sur l'utilité d'un déplacement.

Dans l'ensemble, notre approche présente donc un pire des cas de l'ordre de ce qui peut être rencontré sur les systèmes organisant directement l'ensemble des participants entre eux ; mais un cas moyen bien plus simple à mettre en œuvre en situation réaliste, le groupement en communautés réduisant efficacement la taille des différents sous-problèmes concernés. Remarquons toutefois que plusieurs des étapes considérées (application de l'algorithme 1 sans *tracker*, consultation des participants relative à leurs intérêts) nécessitent des échanges réseau, pouvant influer fortement sur le temps réel d'exécution des procédures concernées.

3.4.2 Répartition des calculs et conditions d'optimalité

Dans le cas où la réécriture de requêtes est disponible, il convient d'étudier la façon dont les charges sont réparties au niveau du graphe abstrait aux communautés. Notons en effet que, dès lors qu'il y a groupement en communautés, le système peut s'assurer[3] que chaque requête exprimée n'est calculée qu'une seule et unique fois par communauté, ce qui permet d'ores et déjà une nette diminution de la redondance des calculs par rapport à des systèmes où deux participants peuvent traiter la même requête de manière indépendante.

Ce point dûment pris en compte, la question de la répartition des calculs porte donc sur les bénéfices apportés par l'organisation des communautés en cascade, donc sur les gains apportés par l'utilisation des réécritures.

Comme évoqué précédemment, le choix d'une approche incrémentale peut signifier accepter de s'éloigner d'un optimum global (qui n'aurait cependant pas nécessairement pu être atteint autrement), mais de rechercher autant que possible des optimums de Pareto, donc des situations où, même si le coût global n'est pas optimal, aucune configuration ne permettrait à chaque intervenant – ici, chaque communauté – de conserver un coût identique ou moindre.

Nous définissons donc le coût global du système comme suit :

Définition 17. *Coût du système*

Pour chaque communauté, le coût nécessaire pour obtenir les résultats correspondant à sa requête est le coût de la réécriture qui lui est affectée, tel qu'évalué par le modèle de coût. Les réécritures affectées étant modélisées par les hyperarcs entrants pour chaque communauté, le coût global du système correspond donc à la somme des coûts de l'ensemble de ses hyperarcs :

$$k(\langle C, \rightarrow \rangle) = \Sigma k(c_1, c_2, \cdots, c_n \rightarrow c) = \Sigma k(c_t), c_1, c_2, \cdots, c_n, c \in C$$

La fonction de satisfaction utilisée dans la définition de la Pareto-optimalité (définition 12, page 62) correspond ici à une diminution des coûts – une communauté du système est d'autant plus satisfaite que le coût de la réécriture qui lui est affecté est faible.

Comparer les coûts de plusieurs systèmes possibles reviendra donc à comparer plusieurs graphes ayant le même ensemble de nœuds C, mais utilisant différents ensembles d'hyperarcs. Nous noterons V_C l'ensemble des graphes *valides* pour un ensemble C donné, c'est-

[3] Cela n'est bien sûr pas une obligation. Réaliser plusieurs fois les calculs pour une même requête peut s'avérer intéressant pour, par exemple, garantir la résilience du système ; mais nous considérons dans cette section le cas où minimiser les coûts est la priorité.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

à-dire l'ensemble des graphes sans cycles pour lesquels chaque communauté non-source reçoit exactement un hyperarc entrant correspondant à une requête équivalente.

Notons que le mécanisme d'échange de ressources permet de considérer que les communautés seront toujours de capacités de diffusion suffisantes, et donc que les graphes de V_C n'ont aucune limitation de degré sortant.

Le coût dépendant du système de réécritures, et donc du langage de requêtes, plusieurs cas peuvent se manifester, selon l'expressivité possible des réécritures.

Théorème 2 (Optimalité en l'absence de cycles potentiels)**.**
Un système à torrent de requêtes pour lequel il n'existe aucune configuration circulaire parmi les réécritures connues est optimal.

Preuve. En l'absence de restrictions de degré sortant, la seule chose pouvant empêcher une communauté de sélectionner une réécriture donnée est que ce choix conduirait à créer un cycle. En conséquence, si les relations de réécriture possibles ne contiennent aucun cycle, la réécriture de coût minimal sera toujours disponible pour chaque communauté. Après stabilisation des réorganisations, le graphe obtenu sera donc nécessairement optimal.

Classiquement, toutes les réécritures basées sur le concept d'inclusion de requêtes, c'est-à-dire limitées à l'utilisation d'une seule vue dont les résultats correspondent à un sur-ensemble de ceux de la requête réécrite, sont dans ce cas. En effet, le seul cas de configuration circulaire possible par l'inclusion de requête correspond à celui de l'équivalence :

$$r_{out} \subseteq r'_{out} \wedge r'_{out} \subseteq r_{out} \Leftrightarrow r_{out} = r'_{out} \Leftrightarrow r \equiv r'$$

Or, deux requêtes équivalentes étant groupées au sein d'une même communauté, le graphe abstrait aux communautés est donc nécessairement sans cycles. Pour d'autres sortes de réécritures, même si les cycles sont théoriquement possibles, une situation dans laquelle aucune configuration circulaire ne peut être trouvée parmi les requêtes actuellement présentes peut éventuellement perdurer pendant la majorité de la vie du système.

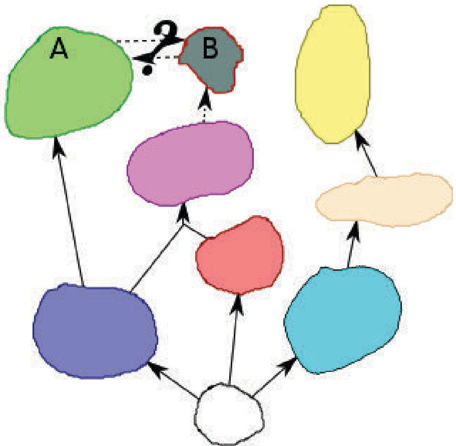


Figure 3.11: Cas de cycle simple entraînant une conservation de Pareto-optimalité

Théorème 3 (Pareto-optimalité en cas de cycles de taille 2)**.**
L'apparition d'une configuration circulaire de réécritures de taille 2, c'est-à-dire impactant seulement deux communautés, conserve l'état de Pareto-optimal d'un système à torrent de requêtes.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Preuve. Nous considérons la situation dans laquelle un graphe, initialement Pareto-optimal, se voit augmenté d'un nouveau nœud c pouvant se réécrire à partir d'un unique nœud c' préexistant, mais pour lequel c' peut également se réécrire à partir de c, sans pour autant que cette configuration circulaire n'affecte d'autres nœuds du système.

Dans ce cas, le nœud c sera placé avant toute tentative de déplacement de c'. Il peut alors se produire trois cas :

- Il existe une réécriture autre que celle impliquant c', de coût moindre. Dans ce cas, cette réécriture sera sélectionnée, et c' pourra être déplacée, ce qui conserve l'optimalité pour l'ensemble $\{c, c'\}$. Le reste du système, non-affecté, demeure optimal ou Pareto-optimal s'il l'était auparavant.

- La réécriture impliquant c' est celle présentant le moindre coût. Dans ce cas, elle sera sélectionnée, et c' ne pourra pas se déplacer. Si la réécriture $c \rightarrow c'$ est de coût inférieur à celui du placement actuel de c', le graphe ne sera donc plus optimal ; mais il sera alors impossible de diminuer le coût de c' sans augmenter celui de c, conduisant à une situation de Pareto-optimalité.

- La réécriture impliquant c' est de coût identique à celui d'une autre réécriture (et inférieur à celui des autres réécritures possibles). Dans ce cas, le choix entre ces deux réécritures peut se faire de manière altruiste, en considérant que l'un des deux choix bloquera c' et en s'orientant donc sur l'autre, les deux étant équivalents de son point de vue. Si chacune des réécritures présente cette caractéristique de cycles restreints, choisir celle dont la communauté concernée a le moins d'intérêt à se déplacer permet de garantir également la Pareto-optimalité.

Ces cas de cycles simples sont illustrés en figure 3.11. Dans cet exemple, la communauté `B`, nouvelle venue, peut être réécrite en fonction de `A`, qui peut elle-même être réécrite en fonction de `B`. Il faudra alors trancher entre leurs intérêts (en donnant la priorité à la nouvelle venue), ce qui fait que le système ne pourra pas demeurer optimal ; mais pourra rester Pareto-optimal.

Une telle situation peut par exemple se produire si, portant sur un même jeu de personnes, l'une des requêtes demande les âges et l'autre les dates de naissance, les deux pouvant être déterminés l'un par rapport à l'autre en connaissant la date courante. On peut également la rencontrer sur des cas d'agrégats, par exemple une somme et une moyenne de valeurs, leur nombre étant connu.

Dans la dernière configuration envisagée (présente de plusieurs réécritures de coût identique), la Pareto-optimalité n'est cependant conservée qu'à condition d'évaluer, chaque fois qu'il y a égalité, les possibilités de réécritures pour l'autre requête concernée, et les coûts qui y sont associés. On pourra considérer qu'une telle évaluation présente un coût organisationnel trop important, et choisir dans ce cas de renoncer à la Pareto-optimalité. Cela peut notamment être envisagé lorsque les possibilités de réécritures peuvent présenter des situations plus complexes :

Propriété 3 (Perte de Pareto-optimalité)**.**
L'apparition d'une configuration circulaire de réécritures impliquant plus de deux nœuds peut entraîner la perte de Pareto-optimalité au niveau du système.

En effet, lorsqu'un cycle implique plus de deux nœuds, il peut se présenter une situation dans laquelle les possibilités de déplacement sont conditionnées par les choix effectués

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

antérieurement, alors que la communauté complétant le cycle n'était pas encore présente dans le système, et ne pouvait de ce fait pas être prise en compte.

Considérons, à titre d'exemple, qu'à l'étape t, une communauté c ait deux possibilités de réécritures r_1 et r_2 de coût identique, sans risques de cycles. Selon le choix effectué, cela entraînera, à l'étape $u > t$, deux graphes distincts, v_1 et $v_2 \in V_C$, de coûts globaux jusqu'alors identiques.

Une communauté c' insérée à l'étape u devient un successeur de c, ce qui provoque l'apparition d'une configuration circulaire impliquant la réécriture r_1. Dans v_2, c utilise la réécriture r_2, et peut donc se déplacer sans difficulté. Dans v_1, en revanche, c utilise la réécriture r_1, et ne peut plus se déplacer sans créer de cycles. On a donc $k(v_2) < k(v_1)$.

Or, le seul moyen de passer de v_1 à v_2 est de déplacer c sur r_2 plutôt que sur r_1, ce qui est un déplacement sans gain de coût, et est donc interdit pour assurer la stabilisation du graphe. Il s'agit donc d'une situation dans laquelle il existe un graphe de V_C de coût moindre que le graphe actuel, qui n'augmente le coût d'aucun nœud, mais ne peut pour autant pas être sélectionné, conduisant à une perte de la Pareto-optimalité.

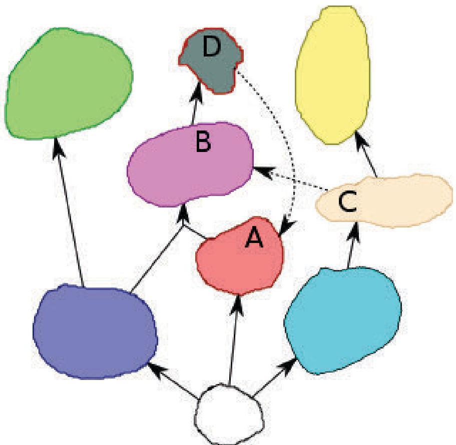


Figure 3.12: Cas de cycle complexe entraînant une perte de Pareto-optimalité

La figure 3.12 en illustre un exemple simple : la communauté `B` empêche la communauté `A` de se déplacer pour bénéficier de l'arrivée de `D`. Si `B` avait été branchée sur `C` (réécriture de coût identique), `A` n'aurait pas été immobilisée et le coût global aurait pu être moindre sans dégrader la situation de quiconque.

Cette propriété entraîne la nécessité, dans le cas d'un langage de réécritures incluant des risques de cycles complexes, de procéder à une évaluation expérimentale de la charge, plutôt que de se contenter d'une analyse théorique.

Notons, en complément de ces points, que la fonction de coût peut tenir compte d'éléments susceptibles d'évoluer au cours de la vie du système, comme la volumétrie des flux de données initiaux ou leur adéquation avec certaines requêtes effectuées (une requête filtrant sur un mot-clef, par exemple, peut recevoir un nombre très variable de résultats au cours du temps, si les sujets abordés dans les publications de la source varient). Il serait donc envisageable de mettre en place une réévaluation périodique des charges en fonction des données obtenues dynamiquement, laquelle présenterait également le risque d'aboutir à la situation de perte d'optimalité décrite dans la propriété 3.

L'étude de ces différentes situations nous conduit à penser que la mise en place d'une fonction d'optimisation $\mathcal{W}$, en complément de la fonction d'organisation Ω, peut poten-

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

tiellement s'avérer judicieuse dans certaines situations. Si elle est mise en place, une telle fonction peut cependant n'être utilisée que de manière très ponctuelle dans la vie du système. En effet, seules les apparitions de nouvelles communautés créant des configurations circulaires sont susceptibles de provoquer le besoin d'une telle intervention. Même lorsque le mécanisme de réécritures est susceptible de créer de telles configurations, celles-ci ne se produisent pas nécessairement lors de la majorité des créations de communautés (qui ne représentent elles-mêmes pas la majorité des événements guidant l'organisation). Pour cette raison, la définition d'une telle fonction, peu utile dans les situations jusque là rencontrées, est laissée à titre de perspective.

3.4.3 Cas de l'absence d'un mécanisme de réécritures

Comme nous l'avons mentionné, la propriété du langage minimale que nous envisageons pour la mise en place d'un système à torrent de requêtes est la possibilité de vérifier l'équivalence entre deux requêtes. Il est donc nécessaire de s'interroger sur ce que peut donner un système reposant sur un langage n'offrant aucune possibilité plus évoluée.

Sans réécriture, toutes les requêtes devront être calculées directement à partir des flux bruts émis par les sources de données. Cependant, la possibilité de vérifier l'équivalence entre requêtes permet tout de même la mise en place de communautés, chacune de ces communautés disposant de son arbre de diffusion interne.

De ce fait, un tel système semble se rapporter au cas des systèmes d'échange par intérêt décrit en section 2.3.2 : une première couche de calcul, constituée des sources primaires de données et de la tête de chaque communauté (ou de l'ensemble des unités en charge de la tâche de calcul, dans un cas plus général autorisant celle-ci à être distribuée au sein de la communauté), autour de laquelle s'organise un arbre de diffusion par requête exprimée, tous disjoints les uns des autres.

Toutefois, l'approche QTor apporte, par rapport à ces systèmes, une caractéristique particulière : les sources primaires de données font elles-mêmes partie d'une communauté chacune, gérant toute requête identité sur la source concernée. De ce fait, un système à torrent de requêtes dépourvu de réécritures (donc un torrent « plat », pourrait-on dire) se présente sous la forme d'un système légèrement plus complexe.

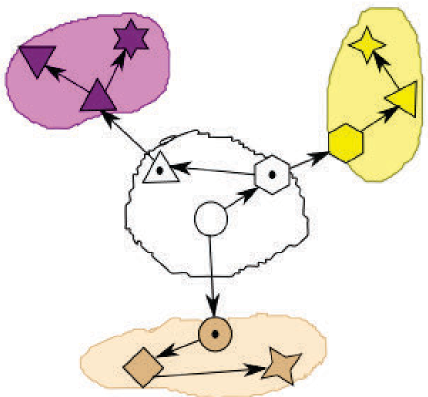


Figure 3.13: Exemple de « torrent plat »

La figure 3.13 présente un exemple de tel système, dans lequel trois communautés, correspondant à trois requêtes différentes, sont connectées à une unique communauté source. Nous pouvons voir que la source (au centre) rejoint l'une de ces communautés

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

afin de procéder au calcul de la requête ; tandis que, pour les deux autres communautés, l'unité en situation de tête doit rejoindre la communauté parente pour y apporter des ressources.

Quoique l'absence de réécritures empêche de diminuer les coûts de traitement des différentes requêtes, cette organisation, présente l'avantage, notable par rapport aux systèmes classiques d'échange par intérêt, qui est de s'assurer qu'une source de données ne pourra pas pas être débordée, même si le nombre de requêtes différentes à alimenter dépasse ses capacités, grâce au mécanisme d'échange de ressources permettant de peupler sa communauté. De ce fait, le calcul et la diffusion sont répartis de manière efficace entre les participants.

3.4.4 Adaptabilité de la proposition

L'adaptabilité est l'un des objectifs principaux de notre approche. Comme mentionné en section 2.1.2 (pages 26 et suivantes), ce concept général se décline selon quatre aspects principaux que sont la *popularité des requêtes* et leur *connexité*, la *volumétrie des flux* et les *capacités des participants*. Au cours de cette section, nous analysons la situation obtenue vis-à-vis de chacun de ces aspects.

Adaptabilité à la popularité des requêtes

L'adaptabilité à la popularité des requêtes consiste en le fait de limiter, autant que possible, les traitements redondants lorsque plusieurs requêtes sont rigoureusement identiques, pour un coût d'identification de ces traitements redondants aussi limité que possible.

À ce sujet, soulignons de nouveau que l'organisation en communautés fait que chaque nouvelle requête exprimée qui est identifiable comme équivalente à une autre requête est insérée dans le graphe sans avoir besoin de recourir aux aspects coûteux de la recherche de réécritures, la mise en cascade de ces communautés permettant encore de gagner en efficacité au niveau des recherches d'équivalences. Ainsi, comme pour les changements de capacités des participants, l'augmentation de la popularité de tout ou partie des requêtes se gère naturellement par des modifications de la structure concrète, simples à calculer et à mettre en œuvre, sans remettre en cause le graphe abstrait.

Réciproquement, la diminution de popularité de tout ou partie des requêtes exprimées, qui, pour sa part, ne nécessite pas d'exploration du graphe abstrait, n'entraîne aucune remise en cause de celui-ci tant qu'il reste au moins un participant s'intéressant à cette requête et que la communauté est donc conservée. Dans cette optique, notons qu'une communauté servant de vue pour le calcul d'autres requêtes peut être pérennisée par le mécanisme d'échange de ressources, même dans le cas où les départs successifs de participants font qu'elle n'est plus exprimée nulle part dans le système.

En effet, le manque de capacités dans la communauté parente entraîne la nécessité, pour les communautés enfants, de leur envoyer des ressources. Un participant issu d'une communauté enfant peut donc continuer de maintenir une communauté même lorsque ses participants d'origine ont quitté le système (ou recentré leurs activités sur d'autres requêtes). Dans une moindre mesure, un participant issu d'une communauté parente de celle menacée de disparition peut également remplir ce rôle en intégrant lui-même les communautés enfants, ou continuant de fournir les résultats aux unités issues de celles-ci.

Cette situation peut entraîner des variations en fonction de l'historique du système, étant donné que nous ne prévoyons pour l'instant pas de mécanisme inverse, permettant de créer des communautés *ex-nihilo* pour former des vues utiles à la réécriture d'autres requêtes. Notons cependant que, dans le cas où un participant d'une communauté enfant

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

se trouve seul à gérer une communauté parente, le coût de la requête associée peut être pris en compte lors de la recherche d'une meilleure réécriture, afin d'estimer de manière plus réaliste les bénéfices d'un déplacement. Ainsi, le maintien en vie artificiel d'une communauté désertée par ses participants d'origine peut ne durer que le temps de sélectionner un meilleur placement dans le graphe, sans générer d'interruption de validité ni avoir à pratiquer cette opération en urgence.

Dans un cas comme dans l'autre de variation de la popularité des requêtes, l'approche QTor permet donc de modifier les relations concrètes entre participants du graphe sans calculs superflus concernant les relations abstraites entre communautés.

D'où,

Conjecture 1 (Adaptabilité à la popularité)**.**
Le regroupement des participants en communauté permet à un système à torrent de requêtes de tirer parti de la popularité en réduisant ses coûts d'organisation, pour obtenir un état dans lequel les calculs redondants sont évités autant que possible. Un tel système est donc hautement adaptable aux variations de la popularité des requêtes.

Adaptabilité à la connexité des requêtes

L'adaptabilité à la connexité des requêtes consiste en le fait de limiter, autant que possible, les traitements redondants lorsque plusieurs requêtes, sans être équivalentes, nécessitent des calculs communs, pour un coût d'identification de ces traitements redondants aussi limité que possible.

Si l'organisation en communauté permet de prendre en compte la popularité des requêtes, leur connexité est ce qui détermine les relations entre ces communautés. En effet, les relations de pertinences servant de base à la détermination des réécritures correspondent au fait que les requêtes concernées partagent certains résultats en commun, et donc présentent une certaine connexité.

Baser l'algorithme de recherche d'équivalence (algorithme 1, page 73) sur les relations de pertinences, s'en servant pour élaguer certains parcours inutiles, revient donc à utiliser les relations de connexité pour simplifier l'organisation du système. Le fait que ces relations soient évaluées au niveau des communautés plutôt qu'au niveau des participants permet par ailleurs de s'affranchir d'une partie de la complexité d'analyse de cette connexité, en diminuant le nombre de comparaisons redondantes à effectuer. En d'autres termes, l'approche QTor utilise la connexité et la popularité comme deux aspects complémentaires dont chacun facilite la gestion de l'autre, là où d'autres manières de procéder conduisent à considérer l'un comme gênant la prise en compte de l'autre.

Pour ce qui concerne les coûts de fonctionnement du système, l'usage d'un système de réécritures efficace permet de considérer l'ensemble des communautés du graphe comme autant de vues possibles permettant de réécrire chacune des requêtes considérées. Le choix entre ces réécritures s'effectue ensuite à l'aide d'un modèle de coût, permettant de sélectionner les combinaisons les plus économiques. L'usage de vues matérialisées étant reconnu[47] comme un moyen efficace de réduire les coûts de calcul des requêtes, notre approche tire donc profit des relations de connexité pour améliorer d'autant la charge de ses participants.

Un système dans lequel la connexité entre les requêtes (ne tenant pas compte de la popularité, donc considérant que deux requêtes équivalentes ne sont pas considérées comme connexes) est très minimale ramène de fait le système à un cas dans lequel il est impossible de procéder à des réécritures, de la même manière que si c'est le langage qui n'offre pas cette possibilité. Or, nous avons montré en section 3.4.3 que notre système continue de

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

fonctionner dans cette configuration, avec une efficacité que l'on peut raisonnablement estimer comme satisfaisante.

Une variation de la connexité des requêtes dans un sens ou dans un autre peut donc être prise en compte au niveau des relations entre communautés, au prix de modifications minimes sur les relations entre participants (changement de têtes en cas de déplacements n'impactant que les communautés concernées).

D'où,

Conjecture 2 (Adaptabilité à la connexité)**.**
L'usage d'un système de réécritures permet à un système à torrent de requêtes de tenir compte de la connexité en réduisant ses coûts d'organisation, pour obtenir un état dans lequel les calculs redondants sont évités chaque fois que le système de réécritures le permet. Un tel système est donc adaptable aux variations de la connexité des requêtes.

Pour autant, des progrès pourraient encore être réalisés à ce niveau en considérant que certaines sous-requêtes communes à plusieurs requêtes du système, mais non-exprimées par les participants présents, pourraient être utilisées pour limiter encore plus cette redondance des calculs. Une telle amélioration est envisagée à titre de perspective.

Adaptabilité aux évolutions des flux

L'adaptabilité aux évolutions des flux consiste en le fait de savoir réévaluer les branchements dans le système en cas de changements importants de volumétrie, dans le but de réajuster l'usage des ressources, pour un coût de mise en place limité. Dans notre cas, ceci porte sur deux aspects : d'une part, l'organisation des communautés dans l'hypergraphe abstrait ; d'autre part, les relations effectives entre participants au sein des communautés.

L'absence d'un mécanisme de remise en cause des placements tel que celui décrit en fin de section 3.4.2 fait que notre système n'est pas directement adaptable aux variations de volumétrie des différents flux de données, du moins au niveau abstrait. Pallier ce problème peut s'avérer complexe dans une configuration dans laquelle un *tracker* centralisé gère l'organisation. En effet, ce *tracker* devrait pour cela surveiller la volumétrie différents flux du système, afin de relancer une évaluation des coûts des différentes requêtes lors de changement, pour vérifier qu'aucun déplacement ne pourrait améliorer la situation.

Une organisation distribuée, dans laquelle chaque communauté décide de son placement dans le graphe abstrait de manière autonome, apporte une solution plus réaliste à ce problème. Dans ce cas, en effet, c'est la communauté elle-même, forcément consciente du coût réel de traitement de sa requête, qui peut alors juger si un meilleur placement s'avérerait opportun ou non, en ayant par exemple la possibilité d'interroger les autres communautés identifiées comme parents possibles à propos de la volumétrie des flux qu'elles manipulent de leur côté (et d'autres aspects tels que la latence).

Nous pouvons donc apporter ici un autre argument au fait de se dispenser d'un *tracker* : dans un cadre dans lequel le modèle de coût s'efforce d'évaluer une requête de manière aussi proche que possible du coût de traitement effectif, laisser à chaque communauté le soin d'évaluer le meilleur placement pour elle au fur et à mesure de l'évolution de la volumétrie peut se faire sans nécessiter de réévaluation globale périodique.

Même si notre système ne s'adapte pas à cet aspect au niveau du graphe des communautés, celui-ci a néanmoins la possibilité de s'adapter indirectement à de telles variations à l'intérieur des communautés. En effet, nous laissons aux différents participants la responsabilité d'évaluer les capacités dont ils disposent, et organisons le système physique en fonction de ces capacités. Or, les capacités réelles d'un participant donné dépendent

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

en pratique de la volumétrie des flux (diffuser un flux très volumineux à deux participants peut ainsi demander autant ou plus de travail que d'envoyer un flux réduit à dix participants). De ce fait, dès lors qu'un participant réévalue ses capacités en fonction des flux qu'il a à gérer, et informe le système des éventuels changements, notre organisation peut adapter l'usage de ces ressources au sein des différentes communautés.

Adaptabilité aux capacités des participants

L'adaptabilité aux capacités des participants consiste en le fait de tirer profit des ressources disponibles dans le système pour ajuster au mieux son fonctionnement, cette fois encore pour un coût raisonnable. Notre algorithme de déploiement d'un arbre de diffusion interne est conçu pour tenir hautement compte des capacités des différents participants ; mais le mécanisme d'échange de ressources augmente encore l'adaptabilité de notre approche à cet aspect.

En effet, rien n'empêche *a priori* un participant doté de capacités suffisantes (et qui l'accepterait) d'intégrer plusieurs communautés. Cela permet, pour un même graphe des communautés, donc une même organisation logique du système, de varier grandement les dispositions physiques correspondantes. Ceci peut être illustré par trois exemples, détaillés sur les figures 3.15, 3.16 et 3.17, correspondant toutes trois au graphe des communautés proposé en figure 3.14.

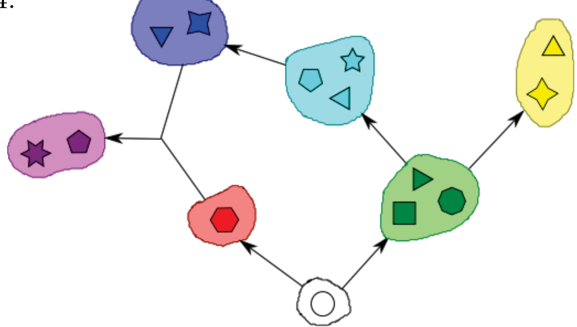
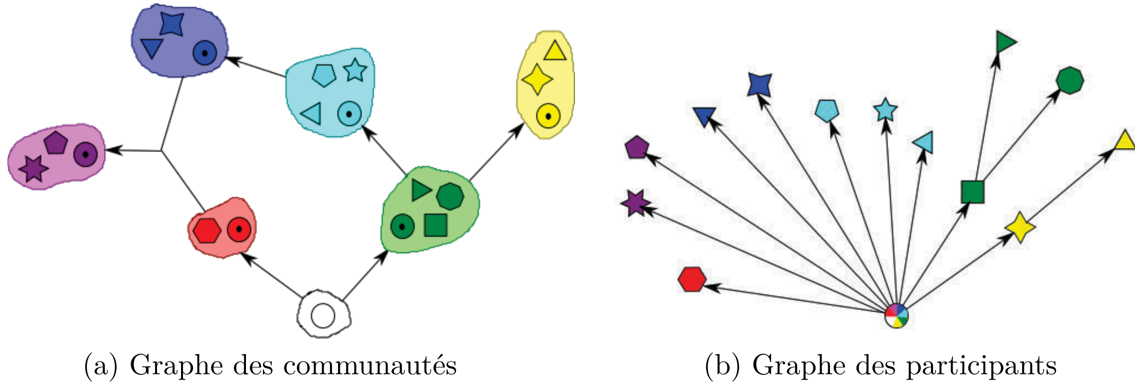


Figure 3.14: Graphe des communautés commun aux figures 3.15, 3.16 et 3.17

Dans ce graphe, la communauté source envoie ses données à deux autres communautés, ici représentées en rouge et vert. La communauté verte sert à son tour deux communautés, une cyan et l'autre jaune. La communauté cyan envoie ensuite ses résultats à une communauté bleue ; et une communauté magenta a besoin de combiner les résultats des communautés rouge et bleue.

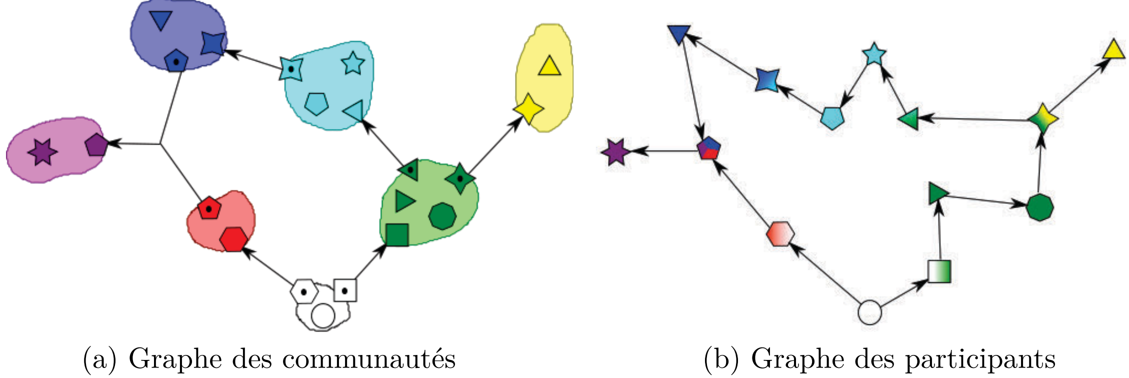


(a) Graphe des communautés (b) Graphe des participants

Figure 3.15: première situation, source puissante

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

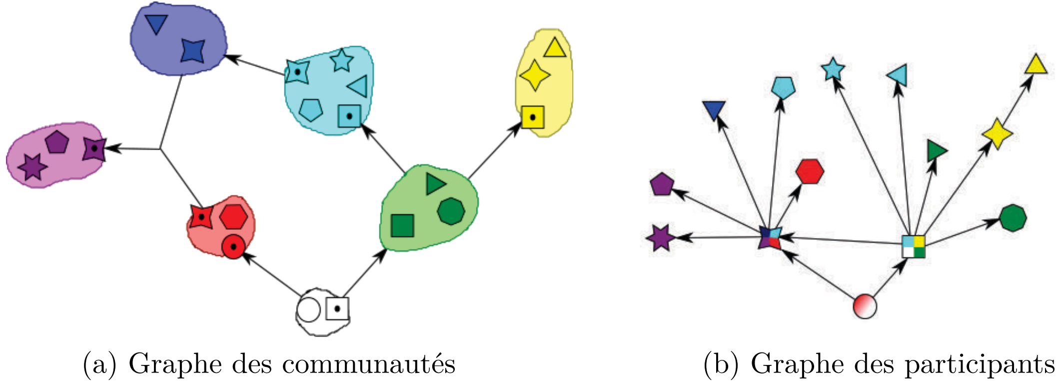
Une source de données suffisamment puissante ayant tout intérêt à s'assurer que le plus possible de participants recevront les résultats qui les intéressent, elle peut intégrer chacune des communautés et y prendre la position de tête, afin de calculer autant de requêtes que possible, et délivrer directement les résultats à un nombre important de participants. Toutefois, si elle n'est pas assez puissante pour diffuser à l'ensemble des membres de ces communautés (où si les différences possibles d'organisation interne font que cela ne lui est pas demandé), elle peut ne pas être en charge de l'ensemble de la diffusion dans toutes les communautés dans lesquelles elle effectue les calculs, comme le montrent, dans l'exemple, les communautés verte et jaune.



(a) Graphe des communautés (b) Graphe des participants

Figure 3.16: deuxième situation, participants très peu puissants

Dans le cas où tous les participants, source comprise, sont de capacité très réduite, aucune communauté n'est en mesure d'envoyer de ressources vers ses communautés enfants, et l'échange de ressource doit nécessairement se faire dans l'autre sens. Dans cette configuration, les échanges sur plusieurs niveaux sont *a priori* inexistants. Il est en revanche nécessaire que les participants devant récupérer les flux auprès de plusieurs communautés constituent plusieurs unités dans ce but, comme c'est ici le cas pour la tête de la communauté magenta. Naturellement, une telle configuration risque d'entraîner des latences élevées, d'autant plus que les arbres de diffusion, au sein de ces communautés, ont une profondeur assez élevée.



(a) Graphe des communautés (b) Graphe des participants

Figure 3.17: troisième situation, participants puissants altruistes

Même si la source est de puissance relativement limitée (elle ne peut intégrer ici qu'une seule des deux communautés qui sont directement reliées avec la sienne), il peut arriver que des participants puissants et altruistes décident d'intégrer plusieurs communautés voisines pour apporter leur aide. Ici, un participant originairement issu de la communauté verte

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

intègre la communauté de la source (où il n'a cependant rien de particulier à faire), ainsi que les communautés jaune et cyan. Un participant originairement issu de la communauté bleue vient récupérer les données de la communauté cyan ; et calcule les résultats pour la communauté magenta, opération pour laquelle il a besoin d'intégrer également la communauté rouge. Dans chacune des communautés qu'il intègre pour acquérir les données, il contribue partiellement à la diffusion des résultats obtenus.

Nous voyons donc qu'une même structure abstraite permet un grand nombre d'organisations concrètes différentes. Il est bien sûr possible de passer de l'une à l'autre de ces différentes situations dynamiquement, en fonction des modifications de capacités des participants, sans jamais avoir à remettre en cause l'organisation abstraite ni devoir effectuer de comparaisons entre les requêtes.

De ce fait,

Conjecture 3 (Adaptabilité aux capacités)**.**
La présence d'un mécanisme d'échange de ressources entre les communautés, de même que le fait que les capacités de diffusions soient le seul moteur de l'organisation intra-communautaires, permet à un système à torrent de requêtes de prendre efficacement en compte les capacités de ses participants pour réduire la latence générale sans causer de surcharge. Un tel système est donc hautement adaptable aux variations de capacités de ses participants.

Synthèse de l'évaluation théorique

Comme nous avons pu le mettre en lumière au cours de cette section, l'approche QTor, basée sur la mise en relations de communautés dédiées à des requêtes équivalentes,

- présente un fonctionnement efficace même si la seule propriété disponible au niveau du langage de requêtes est la possibilité de vérifier l'équivalence entre deux requêtes ;
- atteint l'optimalité des coûts de calcul dans le cas où elle se base sur l'inclusion de requêtes, et peut demeurer Pareto-optimale pour des relations plus complexes, quoique les mesures visant à stabiliser le graphe et éviter les cycles puissent faire perdre cette propriété ;
- est fortement adaptable aux variations de popularité des requêtes comme de capacités des participants, ce dernier point la rendant également indirectement adaptable aux variations de la volumétrie des flux.

Ces propriétés fondamentales nous font considérer que cette approche répond efficacement au problème posé. Toutefois, il demeure nécessaire de quantifier plus précisément cette efficacité, par une évaluation expérimentale.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Chapitre 4

Évaluation expérimentale

L'objet de ce chapitre est à présent de compléter l'analyse théorique effectuée au chapitre précédent par une étude expérimentale, permettant d'en vérifier les conclusions et de comparer le système que nous proposons avec d'autres approches issues de l'état de l'art.

La première section de ce chapitre est consacrée à présenter l'environnement expérimental que nous avons mis en place pour ces expérimentations. Nous présentons ensuite nos résultats selon trois axes : une évaluation globale du système dans une situation correspondant au cas d'application type que nous envisageons ; puis un focus sur deux paramètres essentiels, que sont la popularité des requêtes et les capacités des participants.

4.1 Environnement expérimental

Évaluer une proposition répondant au problème général de requêtage sur des flux est complexe. Il y a, en effet, un nombre important de paramètres à prendre en compte et de critères à mesurer, requérant de procéder à de nombreuses évaluations pour obtenir des résultats complets. Si des mesures statistiques (profondeur du graphe, nombre de calculs effectués…) permettent d'évaluer efficacement certains aspects, la possibilité de visualiser l'organisation obtenue et d'observer la façon dont le système s'adapte aux différents événements rencontrés est également essentielle.

Après avoir rappelé les différentes approches auxquels nous nous comparons, nous présentons le logiciel mis au point pour ces expérimentations, puis la constitution des jeux de tests que nous utilisons.

4.1.1 Éléments comparés

L'analyse de l'état de l'art nous a permis de sélectionner trois référents de comparaisons importants : une solution de type système à miroirs (section 2.3.1, pages 38), pour sa diminution efficace de la latence, et deux approches basées sur l'usage de réécritures, *Semantic Peer-to-Peer Overlays for Publish/Subscribe Networks*[16] (présenté en section 2.4.3, pages 52 et 53) et Delta[53] (pages 53 et suivantes).

Nous nous comparons donc à une solution de **Multicast** dans laquelle chaque participant rejoignant le système se voit demander de créer une unité servant de miroir au flux d'origine. Cette unité alimente ensuite celles chargées de procéder au traitement de chacune de ses requêtes. Cette solution repose sur la mise en place d'un arbre de diffusion unique. En effet, des solutions plus complexes du type de SplitStream[14] pourraient être

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

mises en place au sein de nos communautés aussi bien que sur l'ensemble du système : l'utilisé dans un cas et pas dans l'autre créerait donc un biais indésirable.

Pour *Semantic Peer-to-Peer Overlays for Publish/Subscribe Networks*, ne disposant que des informations fournies dans le document[16], nous avons dû mettre en place notre propre implémentation. Nous avons veillé à suivre scrupuleusement le mode de fonctionnement décrit, mais en y ajoutant une optimisation : nous avons fait en sorte que les participants n'aient pas besoin de réappliquer les filtres déjà appliqués par leurs prédécesseurs. Cette optimisation, sans effet sur les coûts d'organisation, améliore les performances en termes de calcul, et permet ainsi de passer d'un système de routage guidé par l'inclusion de requêtes à un système à réécritures complet (quoique toujours limité à une vue par réécriture). Afin d'intégrer une prise en compte des limitations de capacités, nous avons également développé une extension spécifique. L'usage de cette extension est toutefois optionnel : nous présentons ci-après comme **SemPO (1)** la version basée sur le fonctionnement d'origine, sans prise en compte des limitations de capacités, et comme **SemPO (2)** la version modifiée.

Nous remercions les auteurs de **Delta** de nous avoir fourni le code utilisé dans leurs propres expérimentations. Nous avons ainsi pu intégrer leur développement à notre démonstrateur. La proposition d'origine[53] ne fait que mentionner l'usage d'une fonction d'organisation Ω sans en préciser le fonctionnement, se concentrant sur la fonction d'optimisation $\mathcal{W}$. Pour éviter tout biais, l'implémentation que nous utilisons laisse les participants en attente entre deux appels à cette fonction sans tenter de les connecter. Nous n'évaluons donc ici que les coûts et bénéfices de leur optimisation globale, et non les aspects intermédiaires.

En complément, nous avons également envisagé le cas des systèmes centralisés. Nous avons ainsi mis en place un **Unicast (1)**, dans lequel la source est seule en charge de la diffusion, mais ne procède à aucun calcul, et un **Unicast (2)** dans lequel elle procède à la fois au calcul et à la diffusion. Dans ce second cas, la source utilise la fonction de vérification d'équivalence pour ne traiter qu'une fois chaque requête, mais ne fait appel à aucun autre mécanisme d'optimisation interne.

Enfin, nous avons choisi d'évaluer une version dérivée de l'approche que nous proposons, pour laquelle l'utilisation du système de réécritures est désactivée. Ceci correspond au cas, décrit en section 3.4.3, où ce mécanisme n'est pas disponible : comparer les résultats de cette variante et de notre proposition complète permet donc de d'évaluer l'apport de la prise en compte de la connexité des requêtes. Nous désignons ce mode d'organisation comme **fQTor** (« *flat QTor* »).

Le tableau 4.1 dresse une rapide comparaison de ces différentes approches, considérant cinq critères : la propriété du langage la plus avancée utilisée, la reproductibilité de l'organisation (déterministe, en fonction ou non de l'ordre d'arrivée des requêtes, ou non déterministe), le risque de surcharge des capacités de diffusion, et la présente, ou non, de plusieurs unités pour les participants n'exprimant qu'une seule requête. Concernant le risque de surcharge, précisons que le risque n'est réellement nul, pour les approches concernées, que si chaque participant est capable de diffuser au moins une fois le flux de chaque requête sur laquelle il travaille, donc si son degré sortant maximal est au moins égal au nombre de ses unités. Enfin, notons que, afin de biaiser le moins possible les résultats, le même code est utilisé chaque fois que deux propositions doivent utiliser un algorithme similaire (exploration du graphe dans SemPO et QTor, organisation interne dans QTor, fQTor, SemPO (2) et Multicast).

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Approche considérée	**Propriété du langage**	**Organisation déterministe**	**Risque de surcharge**	**Unités par requête**
Unicast (1)	Aucune	Oui	Fort	1
Unicast (2)	Équivalence	Oui	Fort	n (source)
Multicast	Aucune	Selon ordre	Nul*	2 (miroir)
SemPO (1)	Inclusion	Selon ordre	Fort	1
SemPO (2)	Inclusion	Selon ordre	Faible	1
Delta[53]	Réécriture	Non	Faible	1
fQTor	Équivalence	Selon ordre	Nul*	n
QTor	Paramétrable	Selon ordre	Nul*	n

(En-tête des colonnes : **Caractéristiques envisagées**)

Tableau 4.1: Caractéristiques des différentes approches considérées.

∗ *Sous hypothèse, confer page précédente.*

4.1.2 Notre démonstrateur, FleDDi

Afin de pouvoir comparer efficacement toutes ces approches, il était nécessaire de mettre en place un environnement d'expérimentation commun. Nous avons donc développé un démonstrateur[34], baptisé FleDDi (pour *FLExibility of Data DIssemination*), dans l'objectif de permettre d'effectuer des simulations, de permettre un usage interactif et de servir à un déploiement réel.

Le premier objectif de FleDDi est donc de fournir un **simulateur**, capable de procéder rapidement à des expérimentations de chacune des approches. Pour cela, nous avons choisi de baser le fonctionnement sur l'usage d'un *tracker*. Celui-ci collecte les requêtes des différents participants sous forme textuelle, puis détermine l'organisation du système, et communique aux participants le code à exécuter pour leurs différentes transformations, ainsi que les instructions concernant les destinataires du flux de résultat de chaque transformation. Il suffit donc de sélectionner, au démarrage, l'approche que le *tracker* devra utiliser, et de laisser le *batch* se dérouler.

Ce simulateur utilise PeerSim[66] pour gérer les communications entre nœuds, permettant de mettre en relation un nombre important de participants de manière efficace. L'exécution d'une expérimentation génère, à intervalles spécifiés dans le jeu utilisé, des informations sur les différentes métriques mises en place :

- Le nombre total de participants dans le système
- Le temps écoulé, en millisecondes,
- Le nombre de vues dont la pertinence a été évaluée
- Le nombre d'appels à la fonction de recherche de réécritures
- Le nombre de vues effectivement utilisées pour les réécritures
- Le nombre de requêtes réécrites obtenues
- La distance moyenne des requêteurs à la source, en nombre de connexion
- La distance la plus élevée des requêteurs à la source, en nombre de connexion

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

- Le nombre moyen d'analyses effectuées sur les tuples
- Le nombre moyen de diffusions des tuples
- Le pourcentage de charge de diffusion de la première source de données
- Le nombre de nœuds ayant été déplacés dans le graphe
- Le nombre de nœuds dont les capacités sont surchargées

Toutes ces informations sont réinitialisées après chaque relevé, permettant de suivre progressivement l'évolution du système. Toutes les courbes présentées dans ce chapitre ont été obtenues de cette manière.

Un inconvénient de ce mode de fonctionnement est cependant que tous les événements à prendre en compte doivent être renseignés à l'avance dans le fichier d'instructions fourni et sont exécutés quoiqu'il arrive. Pour pallier ce problème, notre simulateur intègre également une interface graphique permettant un **contrôle interactif**. Utilisant la bibliothèque GraphStream[36] pour la visualisation du graphe, cette interface permet, en cours de vie du système, de déclencher des événements arbitraires, afin d'observer dynamiquement la façon dont le système se modifie. Ce second cas d'usage de notre démonstrateur présente un fonctionnement analogue au premier, lisant ses instructions d'origine depuis un fichier dédié (ces instructions pouvant également être générées de manière aléatoire au lancement), mais son utilisation se limite généralement à un nombre plus réduit de participants afin de conserver sa lisibilité. Les captures d'écran présentées dans cette section sont issues de cette interface.

Enfin, la troisième fonctionnalité majeure de notre démonstrateur consiste en un **déploiement réel**. L'objectif est ici de faire communiquer par le réseau plusieurs machines (notre prototype utilise pour cela un ensemble de *rapsberry pi 2*). Chaque participant peut ainsi être contrôlé par l'intermédiaire d'un service Web embarqué, permettant à la fois de consulter les informations reçues et d'émettre de nouvelles instructions. Ce troisième cas d'usage, n'étant plus simulé, ne repose en effet plus sur l'usage d'un fichier d'instruction, mais sur des événements directement générés par les usagers du système.

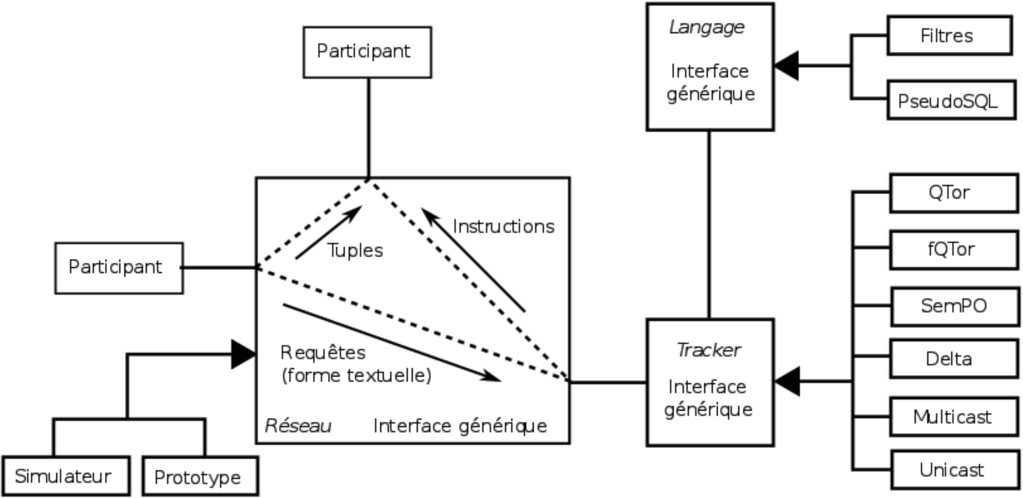


Figure 4.1: Schéma de fonctionnement de FleDDi

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Un point important à noter ici est donc qu'une même implémentation d'une proposition donnée permet à la fois de réaliser des simulations (pour obtenir des statistiques représentatives de son comportement général aussi bien que pour visualiser dynamiquement certains comportements particuliers, permettant une bonne compréhension du fonctionnement) et de mettre en œuvre un véritable fonctionnement réparti du système. La figure 4.1 présente succinctement le mode de fonctionnement du démonstrateur. L'ensemble est codé en Java, portable (une archive Jar permet de faire fonctionner le projet entier sans devoir dépendre de bibliothèques externes[1]) et conçu pour être aisément extensible en cas de besoin. L'annexe trois fournit une documentation technique du fonctionnement actuel du démonstrateur et des possibilités d'ajout de nouvelles fonctionnalités.

4.1.3 Jeux de test type

Les expérimentations présentées dans ce chapitre reposent sur l'usage d'un langage de filtres par mots-clefs, autorisant les opérateurs *et logique* (conjonction) et *ou logique* (disjonction). Les requêtes sont donc constituées d'un ensemble de mots-clefs séparés par les opérateurs logiques correspondants, tandis que les tuples sont pour leur part constitués d'une succession de mots-clefs. L'évaluation de base d'une requête consiste donc en le fait de rechercher différents mots-clefs dans le tuple, de manière paresseuse (si un élément d'une conjonction est évalué à `faux`, la suite n'est pas évaluée ; de même si un élément d'une disjonction est évalué à `vrai`). Afin d'observer l'apport de l'usage de réécritures, nous utilisons la granularité des filtres plutôt que celles des requêtes dans nos taux d'analyse des tuples.

Précisons que, dans le cas des réécritures utilisant de multiples vues, la vérification de l'identificateur physique, permettant de déterminer si un tuple donné avait déjà été reçu ou non, est considérée comme une analyse simple, du fait de la présence d'un identificateur. Le modèle de coût correspondant a donc tendance à favoriser les réécritures utilisant plusieurs vues lorsque le nombre de mots-clefs résultants est strictement inférieur au nombre de mots-clefs résultants de l'utilisation d'une vue unique.

Des précisions supplémentaires concernant ce langage (ainsi que le langage pseudo-SQL que nous utilisons plus fréquemment dans le prototype), et notamment les algorithmes de réécritures utilisés, sont fournies en annexe deux.

Comme mentionné au chapitre précédent, notre fonction d'organisation est sensible à trois sortes événements : l'expression d'une nouvelle requête par un participant (qui intègre le système s'il n'était pas déjà présent), le retrait d'une requête (un participant quittant le système au moment où toutes ses requêtes ont été retirées), et le changement de capacités. Notons que, conformément à l'hypothèse de travail présentée dans ce document (les travaux concernant la répartition des calculs étant menés parallèlement[54]), tout participant est ici considéré comme disposant de capacités de calcul suffisantes pour, au minimum, procéder au calcul de sa requête dans son état initial. Nous ne prenons donc en compte ici que les limitations des capacités de diffusion.

À ces trois événements de base, nous avons ajouté divers événements de contrôle destinés au simulateur. Nos fichiers d'instructions, au format TSV, sont donc constitués d'une succession des événements suivants :

- **calibrate** permet de préciser les capacités d'un participant, en spécifiant un identifiant à lui donner, ainsi que de modifier les capacités d'un participant préalablement inséré, si l'identifiant indiqué est déjà présent dans le système

[1] Précisons que pour cette raison comme pour des raisons de licence, l'implémentation de Delta, reposant sur l'utilisation du solveur linéaire Gurobi[74], ne peut malheureusement pas être diffusée de cette manière.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

- **register** signale l'expression d'une source de données (utilisé en début d'expérimentation, les simulations ici présentées étant basées sur l'utilisation d'une source unique). L'identifiant du participant doit être précisé, ainsi éventuellement que l'expression concernée (inutile dans le cas du langage de filtres)
- **subscribe** signale l'expression d'une nouvelle requête. L'identifiant du participant doit être précisé, ainsi que la description textuelle de la requête
- **withdraw** signale le retrait d'une requête. L'identifiant du participant et l'expression de la requête doivent être spécifiés
- **optimize** déclenche l'appel à l'éventuelle fonction d'optimisation $\mathcal{W}$ du *tracker* utilisé (cette option ne nous servant dans l'immédiat que pour Delta)
- **publish** déclenche la production d'un tuple par une source de données, le numéro de celle-ci (par ordre d'arrivée) et une chaîne décrivant le tuple à émettre étant précisées
- **print** déclenche l'écriture (dans le fichier de sortie précisé) des mesures actuelles
- **pause** permet, lors de l'utilisation du simulateur avec interface graphique de contrôle, de suspendre la lecture du jeu d'instructions, ce qui permet à l'expérimentateur·trice de déclencher manuellement des événements (non-utilisé ici)

Afin de simplifier les fichiers, nous avons toutefois rendu l'usage du **calibrate** initial optionnel : l'identifiant d'un participant peut être remplacé, dans les instructions **register** et **subscribe**, par la simple précision des capacités. Ceci empêche de déclencher d'autres événements pour le même participant, mais évite la multiplication inutile de lignes dans le cas usuel où un participant ne doit avoir qu'un seul événement associé.

Un jeu d'instructions donné est donc caractérisé par différents aspects : en premier lieu, le nombre de participants et les capacités dont ceux-ci disposent. Hors cas de l'étude des variations de capacités, où ces informations sont précisées spécifiquement, les expérimentations ici présentées reposent sur une répartition des capacités suivant une loi de Poisson autour de 30 connexions maximales (cette valeur correspondant notamment au degré sortant maximal envisagé dans Delta[53]) pour les études statistiques, et de 10 connexions maximales pour les visualisations (le nombre réduit de participants rendant de la précédente valeur trop importante).

La répartition des requêtes (ainsi que les niveaux de connexité et de popularité associés) caractérise en second lieu les jeux d'expérimentation. Ces données, variant selon les expérimentations, sont précisées pour chacune d'elles. Précisons que, dans la plupart des jeux ici envisagés, publications et souscriptions sont les principaux événements rencontrés, retraits de requêtes et changements de capacités étant spécifiques à certaines expérimentations particulières.

Les résultats présentés sur les différentes courbes sont le fruit de moyennes effectuées sur cinquante jeux tirés synthétiquement selon des caractéristiques similaires. Les captures d'écran sont pour leur part constituées d'après un jeu unique, les caractéristiques étant ajustées au nombre de participants présenté. Notons que, sauf mention contraire explicite, les participants sont toujours considérés comme altruistes, acceptant de fournir leur aide au système lorsqu'ils en ont les capacités.

La plupart des expérimentations présentées dans cette section, en particulier celles relatives aux mesures de temps d'exécution, ont été conduites sur une machine virtuelle sous Debian GNU/Linux "wheezy" (l'actuelle branche *oldstable*, embarquant la version 3.2 du noyau et la version 7 de l'interpréteur Java) présentant huit processeurs 1862MHz

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

compatibles x86 64bits, et pour laquelle chaque instance de la machine virtuelle Java disposait de 4Gio de RAM.

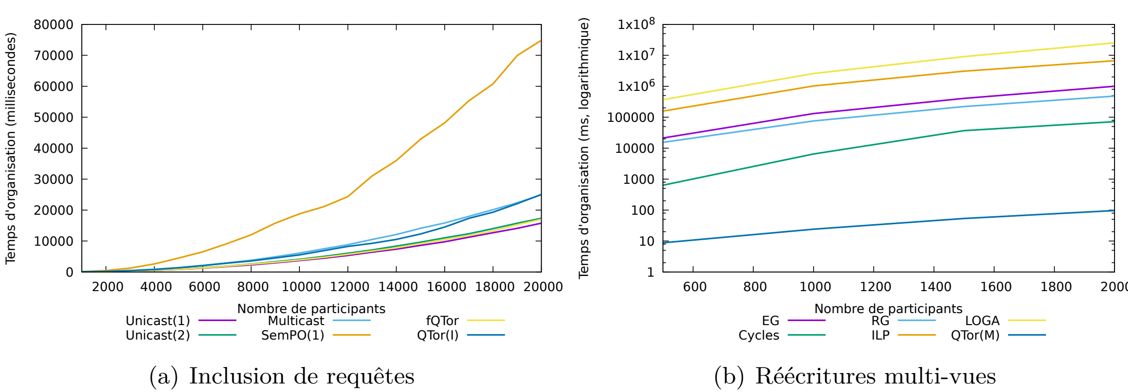
4.2 Évaluation dans le cadre général

La première étape de notre analyse consiste à étudier les aspects généraux relatifs aux différences de structure induites par les différentes approches. La répartition des mots-clefs, dans les contextes de recherche réelle, se rapprochant d'une répartition suivant la loi de Zipf[65, 4], nous avons généré les tuples et mots-clefs d'après une telle loi. Cela entraîne une situation dans laquelle quelques requêtes populaires réunissent un nombre important de participants ; tandis qu'un grand nombre de requêtes présentent une popularité réduite.

Les expérimentations présentées dans cette section ont pour objectif d'étudier l'évolution du système au fur et à mesure de l'arrivée de participants. Chaque jeu est initialement constitué d'un ensemble de 2 000 tuples et 20 000 requêtes, mais les requêtes sont insérées par étapes successives : les 1 000 premières requêtes sont exprimées, puis les 2 000 tuples sont publiés, suite à quoi les 1 000 requêtes suivantes sont exprimées, ce qui est de nouveau suivi par les 2 000 mêmes publications, et ainsi de suite. Il est ainsi possible d'observer l'adaptation au nombre de participants pour une situation donnée.

Nous avons ici considéré deux catégories d'approches : celles pour lesquelles chaque participant dispose au maximum d'un parent par requête (réécritures limitées aux inclusions et approches sans réécritures), et celles pour lesquelles des réécritures utilisant de multiples vues sont possibles, dans l'intention d'évaluer le fonctionnement de QTor dans ces deux cas. Cependant, les durées rencontrées pour l'organisation du système dans le cas de Delta (plus d'une heure en moyenne pour le premier ensemble de 1 000 participants) nous ont conduits à réduire la taille des jeux pour le cas des réécritures à vues multiples : les courbes fournies dans cette section ont été obtenues sur les seuls 2 000 premiers participant, les publications étant effectuées quatre fois (à 500, 1 000, 1 500 et 2 000 participants).

Nous présentons ici en premier lieu notre analyse du coût de l'organisation, puis la structure résultante de cette organisation, et enfin, les coûts de fonctionnement du système.



(a) Inclusion de requêtes

(b) Réécritures multi-vues

Figure 4.2: Expérimentation 4.2 : temps d'organisation

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

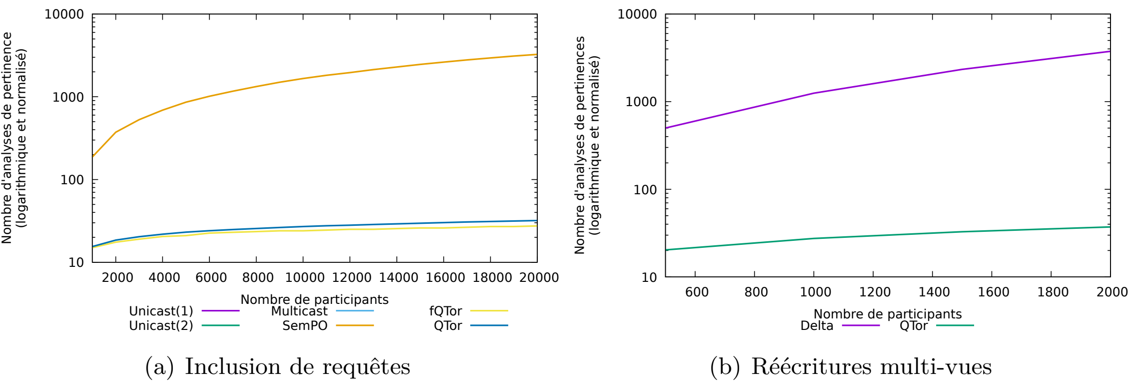
4.2.1 Coût d'organisation du système

Le temps d'expérimentation s'étant avéré problématique, nous nous y intéresserons en premier lieu. La figure 4.2 nous présente deux analyses à ce sujet. En (a), est présentée la moyenne des temps d'expérimentation rencontrés sur les jeux complets pour les cas d'inclusion de requêtes, tandis qu'en (b) est fourni le détail du temps d'organisation de Delta pour chacune des quatre étapes d'optimisation. Les différentes étapes de l'organisation sont ici étudiées : construction du graphe des pertinences (EG), suppression des cycles, construction du graphe des réécritures (RG), appel au solveur linéaire (ILP) et algorithme de réduction de latence (LOGA). Le temps d'organisation de QTor sur des réécritures multiples est précisé à titre de comparaison.

Comme on peut le constater sur la première courbe, le gain de temps induit par le fait de ne considérer qu'une requête par communauté plutôt qu'une requête par participant est notable, ramenant la durée d'organisation de QTor (en cas d'inclusion) au voisinage de celle de la mise en place d'un Multicast sur l'ensemble des participants. La durée d'organisation pour fQTor est d'ailleurs extrêmement proche de celle de l'Unicast (2), les deux procédant au même nombre d'analyses d'équivalences.

Signalons que le temps d'organisation de QTor sur les cas de réécritures multiples pour l'ensemble des 20 000 participants, non présenté ici du fait de la limitation à 2 000 participants pour Delta, était supérieur à son homologue en cas d'inclusion, la fonction de réécritures étant plus lente lorsqu'elle doit comparer plusieurs vues. Il restait néanmoins nettement inférieur au temps d'organisation de SemPO.

Précisons que ces courbes n'ont pas pour vocation à fournir une estimation représentative du temps que mettra l'organisation d'un système en condition réelle : le temps d'exécution réel dépend des caractéristiques de la machine et du langage de programmation utilisé (et des éventuelles optimisations qui l'accompagnent) autant que des jeux de données rencontrés. Il s'agit donc uniquement d'un critère de comparaison, les variations dues à ces aspects extérieurs ayant tout lieu d'être du même ordre de grandeur quelle que soit l'approche utilisée.



(a) Inclusion de requêtes

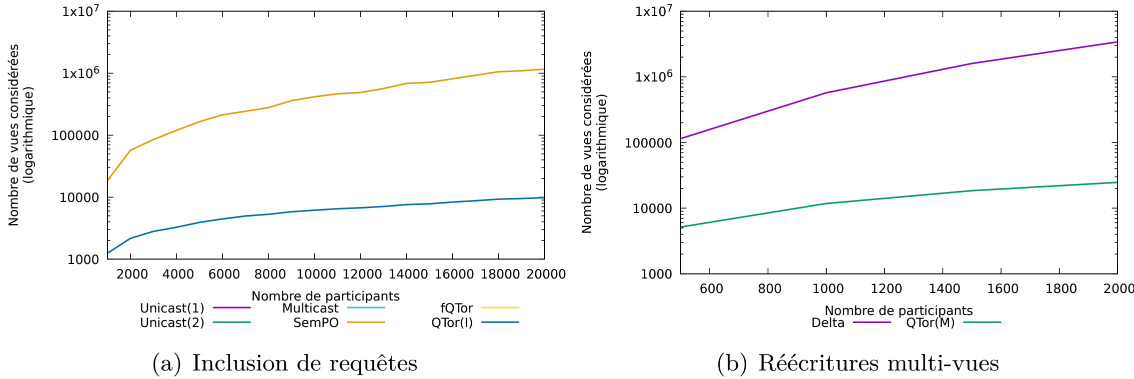
(b) Réécritures multi-vues

Figure 4.3: Expérimentation 4.2 : nombre d'analyses de pertinence

La figure 4.3 nous permet d'identifier l'une des causes de cette différence de durée : le nombre moyen de vues dont la pertinence (ou, dans les cas Unicast (2) et fQTor, indissociables sur la figure, l'équivalence) a dû être évaluée pour chaque nouvelle requête insérée (lequel est indépendant du mode de réécritures choisi). La différence entre le nombre de communautés et le nombre de participants entraîne la nécessité de présenter ces courbes selon une échelle logarithmique.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Précisons que la différence observée entre QTor et fQTor s'explique par deux caractéristiques. D'une part, la recherche d'équivalence peut bénéficier, dans QTor, de l'élagage du graphe des communautés mentionné dans l'algorithme concerné (algorithme 1, page 73), du fait des relations entre communautés. Dans fQTor, les communautés des différentes requêtes n'étant reliées qu'à celle de la source, cet élagage est impossible. D'autre part, QTor utilise un mécanisme de réorganisation après chaque insertion, entraînant lui-même un certain nombre de vérifications. Cette réorganisation n'a pas lieu pour fQTor, étant inutile dans ce cas. Le fait que la courbe de QTor ne soit que légèrement plus haute montre donc que l'élagage lors des recherches est suffisamment efficace pour compenser le coût de la réorganisation du système, illustrant par là l'intérêt de la prise en compte de la connexité.



(a) Inclusion de requêtes

(b) Réécritures multi-vues

Figure 4.4: Expérimentation 4.2 : nombre de vues pertinentes

La figure 4.4 nous montre la contrepartie des courbes précédentes, à savoir le nombre de vues effectivement considérées comme pertinentes après la vérification précédente. Ce nombre n'est ici pas normalisé en fonction du nombre de participants, dans la mesure où il nous renseigne sur le coût de la fonction de réécriture.

On peut constater sur ces courbes que le gain dû au passage aux communautés est ici encore significatif, réduisant ce coût de manière importante. Par ailleurs, les différences entre homologues respectifs (SemPO et Delta présentant, sur cet aspect, la même relation entre eux que les deux versions de QTor entre elles) montrent que la fonction de réécriture basée sur l'inclusion, de complexité moindre, a également moins de vue à traiter. Cela tend également à expliquer les différences de durées entre ces deux modes d'organisation.

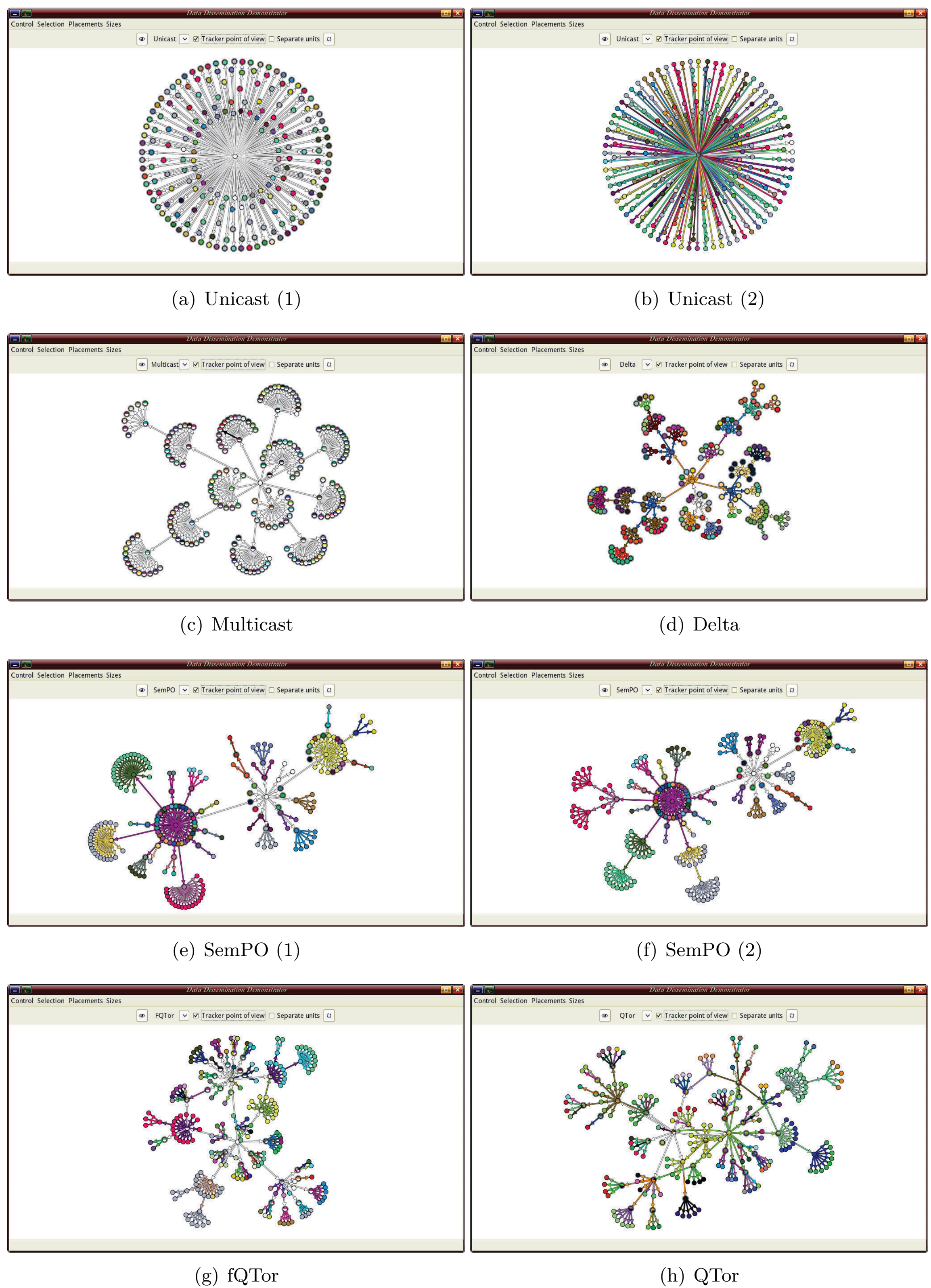
L'ensemble de ces courbes nous conduit donc à considérer que le groupement en communautés est une approche efficace, simplifiant grandement l'organisation du système dans cette configuration réaliste où la popularité suit une loi de Zipf.

4.2.2 Observation de l'organisation du système

Afin de pouvoir évaluer la structure obtenue par ces différentes organisations, nous présentons en figure 4.5 les captures d'écrans obtenues grâce au visualiseur intégré dans FleDDi, pour un jeu de données construit de manière analogue à ceux utilisés pour les expérimentations de cette section, mais réduit à 200 participants. Sur ces représentations, nous avons choisi de ne pas séparer les unités, ce qui conduit à une représentation similaire à nos graphes des participants.

Les deux captures correspondant au cas « Unicast » sont donc assez similaires dans la forme ; cependant, la différence de couleur permet de constater qu'en (a), la source

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés



(a) Unicast (1)

(b) Unicast (2)

(c) Multicast

(d) Delta

(e) SemPO (1)

(f) SemPO (2)

(g) fQTor

(h) QTor

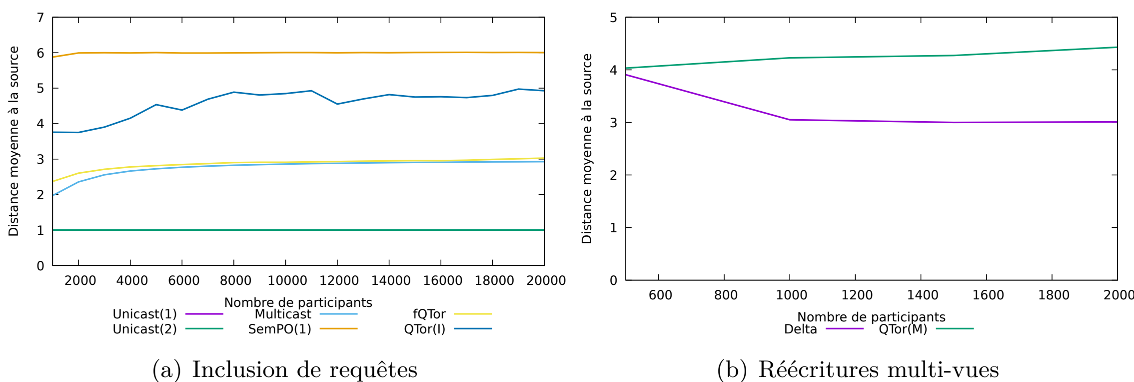
Figure 4.5: Expérimentation 4.2 : graphes des participants

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

envoie son flux brut à l'ensemble des participants, tandis qu'en (b), elle traite l'ensemble des requêtes du système (le nœud correspondant se trouvant donc très coloré) et envoie directement ses résultats à chaque participant.

En (c), la diffusion du flux initial de la source est répartie sur l'ensemble des participants, la quasi-totalité de ceux-ci étant cependant de deux couleurs différentes, puisque devant simultanément gérer ce flux brut et le calcul de leur propre requête. En (e), SemPO place les participants relativement à leurs requêtes, mais le premier requêteur rencontré pour chaque requête doit assumer l'envoi de ses résultats à tous ceux identifiés comme logiquement équivalents, ce qui peut dépasser ses capacités. Nous pouvons constater sur l'image (f) que l'extension que nous avons introduite pour tenir compte des capacités entraîne la mise en place d'arbres de diffusion pour les requêtes populaires. Toutefois, lorsque le nombre de participants devant utiliser une requête donnée, différente de la leur, comme vue pour leur réécriture est trop important par rapport aux capacités des participants traitant cette requête, ces derniers se trouvent tout de même surchargés.

L'image (d) présente l'organisation obtenue par Delta : les limites de capacités sont respectées, mais les participants n'y sont pas spécialement réunis en fonction de leurs requêtes. En (g), en revanche, des communautés sont formées, mais toutes connectées à la communauté source (malheureusement, le mode de placement des nœuds ne permet pas de mettre celle-ci en valeur, attendus que les participants concernés doivent également être proches de ceux de leur(s) autre(s) communauté(s). Tous les participants présentant au moins deux couleurs sont concernés). L'image (h) permet de visualiser l'état obtenu par QTor : là encore, les participants dotés de plusieurs couleurs sont ceux qui ont intégré plusieurs communautés pour apporter leur aide.



(a) Inclusion de requêtes

(b) Réécritures multi-vues

Figure 4.6: Expérimentation 4.2 : distance moyenne à la source

La figure 4.6 présente les distances moyennes à la source obtenues sur l'ensemble des jeux. Si les systèmes centralisés ont une profondeur fixe, au prix d'une surcharge de la source (confer captures 4.5 (a) et (b)), le Multicast présente la courbe la plus basse possible en respectant les capacités des participants sur ce type d'organisation, puisqu'elle tire parti de l'ensemble des capacités sans restrictions liées aux relations entre requêtes.

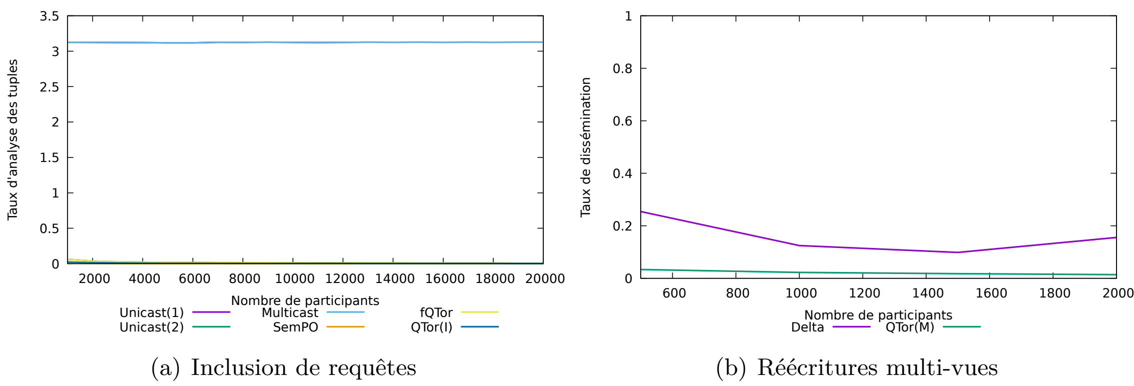
Comme nous pouvons le constater, l'organisation fQTor en donne des résultats très voisins, chaque communauté étant directement connectée à la source. En revanche, QTor et surtout SemPO présentent des courbes plus élevées. Cela s'explique par le fait que l'organisation du système est contrainte par les relations logiques : dans SemPO, la distance à la source de chaque participant est au moins égale au nombre d'étapes requises pour la concrétisation de la réécriture de sa requête (notion de schéma de réécritures, déf-

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

inition 10, page 56). Dans QTor, la distance de chaque *communauté* à la source présente le même aspect, mais le mécanisme d'échange de ressources entre communautés permet de diminuer la latence au niveau des participants.

Enfin, la différence entre QTor et Delta, dans le cas des réécritures mobilisant plusieurs vues, s'explique par le fait que l'algorithme de réduction de latence, dans Delta, est autorisé à réaugmenter les coûts de calculs. Dans QTor, au contraire, un seul participant au maximum intègre les communautés parentes, en conservant les relations entre requêtes sélectionnées par le modèle de coût.

4.2.3 Évaluation du fonctionnement du système



(a) Inclusion de requêtes

(b) Réécritures multi-vues

Figure 4.7: Expérimentation 4.2 : taux d'analyse des tuples

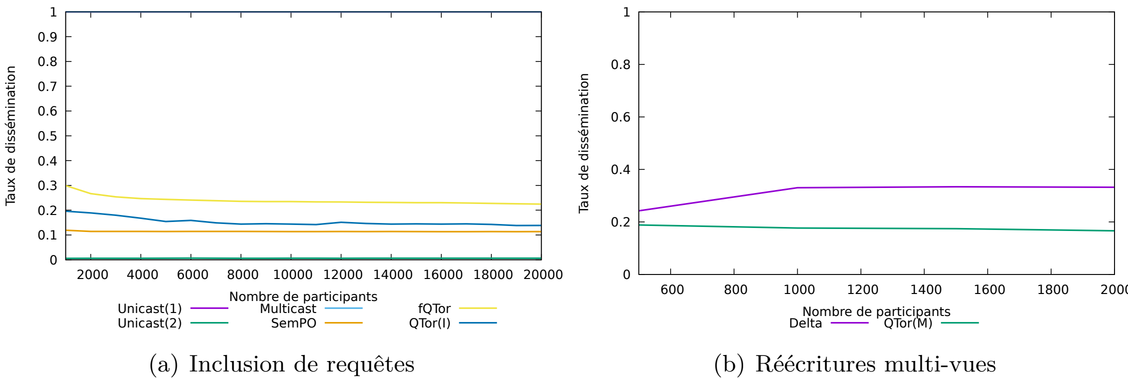
En complément de ces différents aspects organisationnels, nous devons également étudier les coûts de fonctionnement du système lui-même, c'est-à-dire les coûts de calcul et de diffusion des tuples.

La figure 4.7 présente le taux d'analyse des tuples, c'est-à-dire le nombre de filtres évalués pour chaque tuple reçu et pour chaque participant du système. Comme on peut le constater, les différentes approches agissant sur le traitement des requêtes (par réécritures, ou simplement en regroupant les équivalences entre elles) permettent toutes une diminution efficace du nombre de traitements effectués.

Le fait que ce taux, et donc la charge de calcul correspondante, soit plus faible dans QTor que dans Delta est la contrepartie du gain en distance à la source observé sur la courbe 4.6 (b) : les gains appréciables en termes de latence obtenus par le dernier algorithme de cette organisation correspondent à une perte au niveau des coûts de calcul.

Les constats analogues s'effectuent sur la figure 4.8 pour ce qui concerne le taux de dissémination, soit le nombre de diffusions de chaque tuple pour chaque participant du graphe. La différence entre fQTor et SemPO correspond à l'avantage de la prise en compte de connexité : dans les deux cas, seuls les tuples correspondant aux résultats des requêtes sont envoyés entre participants dont les requêtes sont équivalentes. Dans fQTor, toutefois, toute communauté devant envoyer des ressources dans la communauté de la source doit récupérer le flux de celle-ci, ce qui, sur ce type de requêtes, augmente le nombre de données à émettre. À nouveau, le mécanisme d'échange entre communautés place QTor dans une situation intermédiaire, les participants chargés de fournir des ressources aux communautés parentes devant partager un flux plus important.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés



(a) Inclusion de requêtes

(b) Réécritures multi-vues

Figure 4.8: Expérimentation 4.2 : taux de diffusion des tuples

Conclusion de l'évaluation générale

Ces expérimentations nous permettent de constater que l'approche QTor permet, dans cette configuration proche du cas moyen que nous envisageons de rencontrer en situation réelle, une diminution des coûts fonctionnels du système (calcul des requêtes et diffusion des résultats) comparable à celles que l'on peut observer pour d'autres systèmes basés sur les réécritures. Toutefois, notre approche présente, par rapport à ceux-ci, l'intérêt de réduire de manière très satisfaisante le coût d'organisation du système, du fait de l'usage de communautés. En contrepartie, la distance des participants à la source, élément majeur de la latence du système, sans s'avérer déraisonnable, est contrainte par les relations entre requêtes et reste plus élevée que ce que l'on peut rencontrer pour d'autres approches. Dans les cas où la limitation de latence est l'objectif premier, toutefois, la désactivation du mécanisme de réécritures permet de simplifier davantage encore l'organisation du système, et de se rapprocher de la distance minimale que l'on peut observer dans un cas de Multicast, pour un surcoût modéré en termes de coûts fonctionnels.

4.3 Influence de la variation de popularité

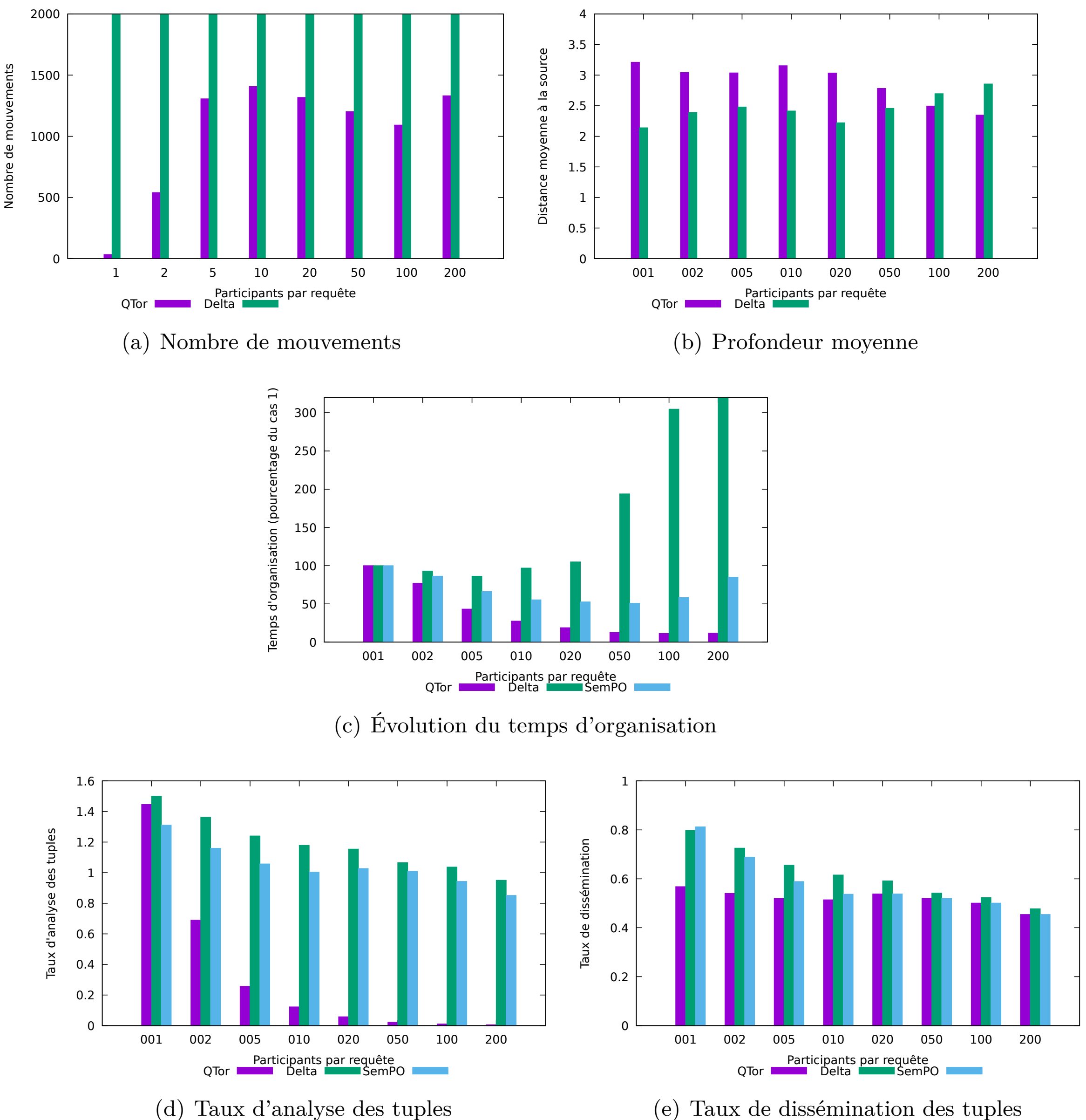
Les expérimentations présentées dans cette section sont destinées à évaluer l'influence de la popularité des requêtes sur l'organisation. Cette étude se compose de plusieurs aspects : elle porte tout d'abord, un ensemble de jeux dans lequel les popularités sont homogènes, puis sur quelques cas limites envisagés lors de l'analyse de ces jeux. Enfin, nous présentons le cas de variations dynamique de la popularité en cours de vie du système.

4.3.1 Étude en popularité homogène

Compte tenu des temps d'expérimentation mesurés lors des expérimentations précédentes, les jeux utilisés dans cette section ont été limités à 2 000 participants. Notre objectif étant ici d'étudier l'influence de la popularité des requêtes, nous avons décliné plusieurs situations : chaque jeu de participants (les capacités et ordres d'arrivées demeurant inchangés au sein d'un même jeu) a été d'abord lancé dans une situation où chaque requête du système était exprimée une unique fois, puis dans une seconde où chaque requête était exprimée par deux participants, et ainsi de suite, jusqu'à la situation où dix requêtes étaient exprimées dans le système, chacune par 200 participants. Une fois l'ensemble de ces participants placés, 2 000 tuples (les mêmes pour chaque déclinaison d'un jeu donné)

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

ont été émis dans le système. Seules les approches utilisant des réécritures ont ici été comparées, le Multicast n'étant pas concerné par ces variations de popularité. La figure 4.9 présente l'ensemble des résultats obtenus, aussi bien en termes d'organisation du système que de charges fonctionnelles.



(a) Nombre de mouvements

(b) Profondeur moyenne

(c) Évolution du temps d'organisation

(d) Taux d'analyse des tuples

(e) Taux de dissémination des tuples

Figure 4.9: Expérimentation 4.3.1 (tous résultats)

En (a), nous voyons le nombre de participants qui ont été déplacés (et donc déconnectés de leur parent précédent) au cours de l'organisation. Une estimation de ce nombre peut difficilement être fournie concernant Delta, dans la mesure où nous n'évaluons ici que la fonction d'optimisation. En effet, même lorsque la fonction d'organisation choisie ne provoque aucun déplacement de nœud préalablement inséré, le nombre de mouvements dépend fortement de la fréquence à laquelle la fonction d'optimisation est appelée. Nos expérimentations nous ayant permis de constater que cette fonction d'optimisation ne

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

conservait qu'un nombre réduit de liens d'une itération sur l'autre, nous considérons que le nombre de déplacements, pour cette approche, est au moins comparable au nombre de participants dès que l'optimisation est effectuée plus d'une fois. Pour cette courbe comme pour la suivante, SemPO n'est pas mentionné, dans la mesure où il n'aurait s'agit ici que d'évaluer l'extension que nous avons proposée pour prendre en compte les limitations de capacités, et non la proposition d'origine.

En (b), nous voyons la profondeur moyenne dans le graphe, représentative de la latence obtenue. Nous constatons que Delta entraîne une latence plus faible que QTor dans les situations de peu de redondance, et que le rapport s'inverse à mesure que la redondance augmente. Cela s'explique par l'utilisation, par Delta, d'un algorithme de réduction de la latence autorisé à réaugmenter les coûts de calcul et de diffusion (ce que l'on peut observer sur les deux dernières courbes), ce qui, dans QTor, correspondrait au fait de casser les communautés. Lorsque la redondance devient élevée, cet algorithme devient moins efficace que celui d'organisation interne des communautés dans QTor.

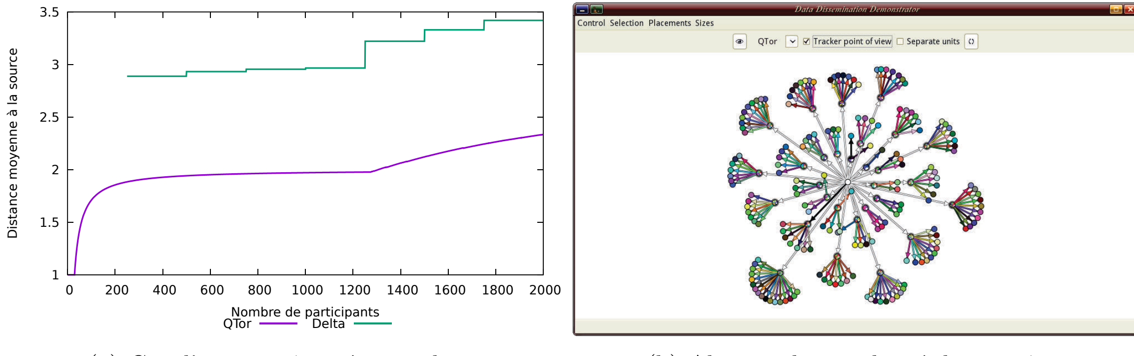
En (c), nous voyons l'évolution du temps d'organisation du système. Ayant pour objectif de mesurer l'évolution en fonction des variations de popularité, et non le temps réel (peu représentatif), nous avons décidé de ramener les temps mesurés à des pourcentages du temps mis pour organiser le système dans le cas de popularité nulle (d'où le fait que les trois valeurs soient de 100% dans le premier cas). Nous constatons donc que l'organisation de QTor est d'autant plus efficace que le taux de popularité est important ; tandis que Delta présente l'évolution inverse, atteignant une augmentation très importante lorsque la popularité devient très importante (la durée pour le dernier cas, coupée pour des raisons de lisibilité, avoisine 960%). La diminution s'explique, pour QTor, par le fait que les organisations internes aux communautés, plus simples à mettre en place, deviennent majoritaires par rapport à l'organisation du graphe des communautés. Pour Delta, en revanche, les participants exprimant des requêtes équivalentes forment autant de composantes connexes dans le graphe des pertinences, augmentant d'autant la difficulté, principalement sur les étapes de suppression des cycles et de réduction de latence. Notons tout de même que, pour le premier cas, qui lui était favorable, le temps d'organisation de Delta était nettement moins élevé que lors des expérimentations de la section précédente. Les temps observés lors de celles-ci étaient donc vraisemblablement dus à la présence, lorsque les requêtes suivent une loi de Zipf, de quelques communautés de taille importante.

Enfin, les histogrammes (d) et (e) présentent les coûts fonctionnels du système. L'effondrement observé concernant les coûts de calculs, pour QTor, correspond au fait que le traitement des requêtes n'est effectué qu'une seule fois par communauté, ce qui diminue donc d'autant la charge globale que le nombre de communautés est petit par rapport au nombre de participants. Utiliser de véritables réécritures (même si limitées à l'inclusion de requêtes) dans SemPO, comme nous le faisons sur les autres courbes présentées dans ce chapitre, conduit à une situation analogue. Nous avons toutefois ici décidé, à titre exceptionnel, de présenter les résultats obtenus en désactivant notre optimisation sur les calculs, c'est-à-dire en faisant en sorte que chaque participant applique l'ensemble de ses filtres sur toutes les données qu'il reçoit, même si celles-ci ont déjà été évaluées (le gain correspondant alors uniquement à la réduction du flux d'entrée). Ce choix a pour but de fournir un point de comparaison concernant le taux d'analyse des tuples dans Delta.

4.3.2 Cas limites de l'homogénéité

Deux cas limites particuliers peuvent être envisagés concernant la situation d'homogénéité des requêtes. Ainsi, la figure 4.10 (a) présente le cas où toutes les requêtes exprimées

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés



(a) Cas d'une requête très populaire

(b) Absence de popularité des requêtes

Figure 4.10: Cas limite des expérimentations précédentes

sont équivalentes. Cela entraîne donc, pour QTor, une situation dans laquelle il n'y a que deux communautés, celle de la source (constituée d'un unique participant et dont l'organisation donc est triviale), et celle de cette unique requête. L'organisation y est donc très proche de celle d'un Multicast applicatif. Delta, en revanche, très défavorisé par cette situation, fourni de moins bons résultats. Précisons que la forme de la courbe (notamment pour l'absence de valeurs avant 200 participants) est due au fait que la fonction d'optimisation a ici été appelée tous les 200 participants insérés.

Le premier cas présenté dans les histogrammes de la figure 4.9 correspond en fait au cas limite exactement opposé, puisqu'il représente un cas où aucune requête n'est populaire. De ce fait, il présente une situation assez défavorable pour QTor, dans laquelle le graphe abstrait est de même taille que le graphe des participants. Pour autant, le mode d'organisation de QTor s'adapte assez bien à ce cas de figure. En effet, le mécanisme d'échange de ressources entre communautés permet de faciliter l'organisation du graphe des communautés en dispensant de tenir compte des limitations de capacités à ce niveau, l'organisation interne de chaque communauté étant ensuite d'autant plus simple que le nombre de participants est petit.

La figure 4.10 (b) illustre cette situation dans laquelle chaque requête n'est exprimée que par un participant. Cette capture d'écran a été obtenue grâce à la visualisation de FleDDi sur un cas similaire à ceux présentés ici (quoique, cette fois encore, doté de moins de participants pour rester lisible). Nous y voyons un certain nombre de participants contribuer à plusieurs requêtes, du fait des relations entre les communautés auxquels ils participent initialement. Cela montre que notre approche s'adapte bien, y compris aux cas où la popularité des requêtes est très faible. Néanmoins, à la différence de l'algorithme de réduction de latence utilisé par Delta, cet échange de ressources, s'il concerne ici une majorité des participants du système, reste contraint par le modèle de coût, raison pour laquelle les coûts sont moindres et la latence plus élevée.

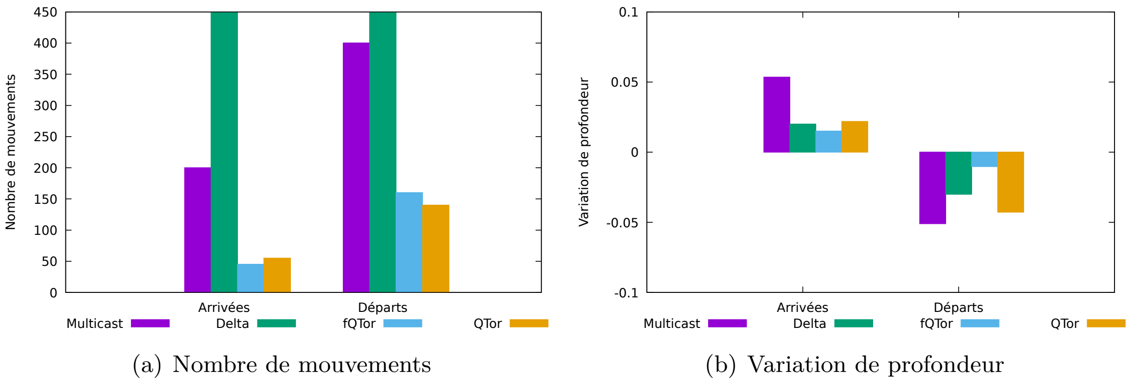
4.3.3 Variations dynamiques de la popularité

L'objectif de cette expérimentation est d'évaluer la façon dont le système réagit aux variations de popularité d'une requête. Afin d'envisager à la fois l'augmentation et la diminution de la popularité, nous avons constitué des jeux de base constitués de 2 000 participants (dont les requêtes sont tirées d'après une loi de Zipf, formant une situation proche de celle de l'expérimentation 4.2), puis étendu ces jeux de bases par deux séries d'événements. Après insertion de l'ensemble des requêtes et relevé de l'état du système,

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

nous déclenchons dans le premier cas l'arrivée de 200 participants dans une communauté donnée. Dans l'autre cas, nous déclenchons le départ de la quasi-totalité des membres de la même communauté. Nous avons veillé à ce qu'il reste toujours au moins un participant ayant exprimé la requête liée à la communauté, afin de ne pas parasiter les résultats observés par d'éventuels effets dûs aux déplacements des communautés enfants. Afin que les deux événements soient d'ampleur similaire, nous avons sélectionné la communauté concernée de telle sorte qu'elle contienne aux alentours de 200 participants lors de l'organisation de base (cette caractéristique faisant que la communauté sélectionnée était très fréquemment proche de la source).

La figure 4.11 présente les résultats observés en termes de variation moyenne de profondeur du graphe des participants, et de nombre de déplacements effectués (ne tenant pas compte des départs définitifs). Notons que le nombre de déplacements observé pour Delta (ici tronqué) vient du fait que l'optimisation affecte l'ensemble du système.



(a) Nombre de mouvements

(b) Variation de profondeur

Figure 4.11: Expérimentation 4.3.3 : impact des arrivées et départs

Comme nous pouvons le constater, les variations de profondeur ont été plus importantes dans le cas du Multicast que des autres approches (la profondeur résultant de ces variations demeurant la plus faible des quatre approches ici étudiées). Cet effet est dû à l'utilisation d'un unique arbre de diffusion pour l'ensemble des participants, expliquant aussi le nombre plus important de participants déplacés. Pour QTor et fQTor, celles-ci permettent de localiser les modifications, et donc de réduire d'autant le nombre de nœuds affectés, même si le mécanisme d'échange de ressources conduit quelquefois à impacter les communautés limitrophes. Nous notons, dans le cas des départs, que l'utilisation de ce mécanisme d'échange de ressources (plus importante dans QTor que dans fQTor, ce dernier n'autorisant que les branchements à la source) permet de stabiliser le graphe, les unités envoyées pour l'échange de ressource demeurant pour leur part en place.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Conclusion de l'évaluation concernant la popularité

Ces diverses expérimentations nous permettent de conclure donc que l'approche que nous proposons, conformément aux conjectures formulées lors de l'analyse théorique, s'adapte efficacement aux différentes variations de la popularité des requêtes, pouvant tirer profit d'une popularité importante de certaines requêtes à la fois pour réduire très fortement les coûts de calcul (chaque requête n'étant calculée qu'une fois par communauté) et pour limiter la distance à la source (l'organisation intra-communautaire ayant dans ce cas davantage d'influence que pour les communautés de très faible taille). Par ailleurs, le mécanisme d'échange de ressources entre communautés permet de prendre efficacement en compte les cas où la popularité demeure faible, notre approche offrant alors des résultats comparables à ceux d'autres approches à réécritures, pour un coût d'organisation demeurant raisonnable.

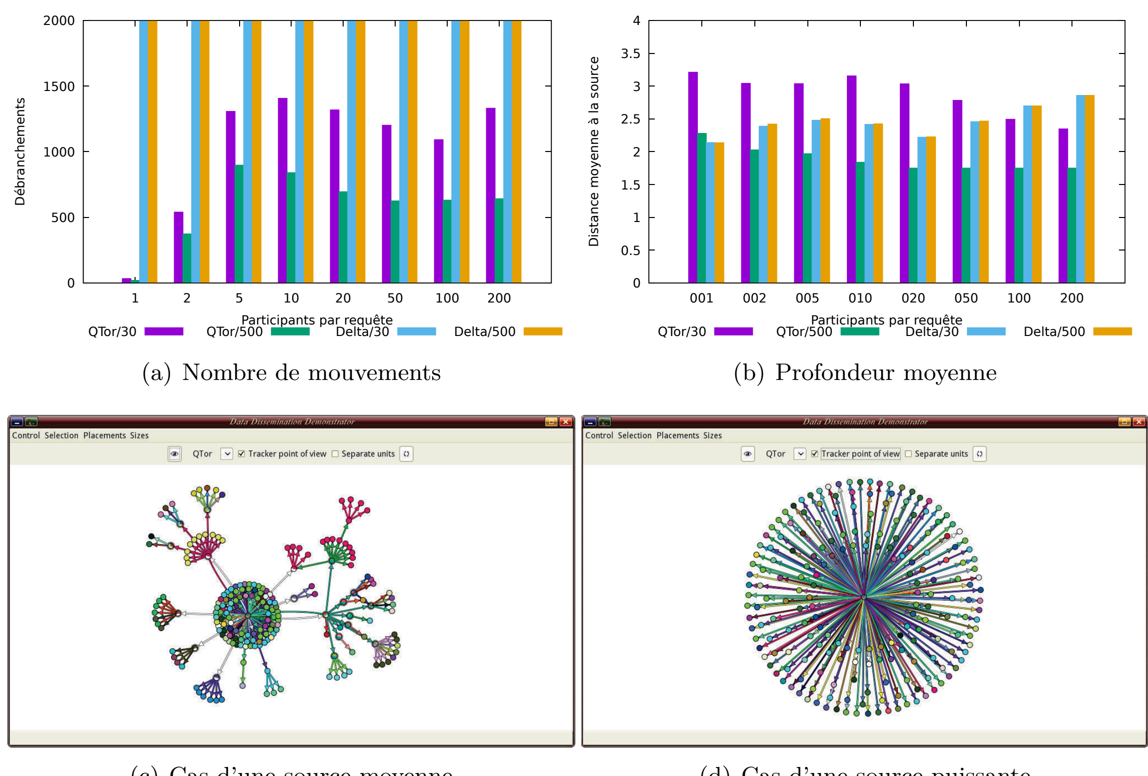
Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

4.4 Influence de variation des capacités

Les expérimentations présentées dans cette section sont destinées à évaluer l'influence des capacités des participants sur l'organisation. Nous n'y proposons aucune mesure des coûts de fonctionnement du système, dans la mesure où l'effet sur celui-ci est mineur : les variations de capacités influent sur la localisation des calculs, mais ceux-ci restent effectués dans tous les cas (les charges de diffusion peuvent néanmoins varier plus ou moins dans certaines approches, mais nous considérons ici ce point comme mineur). Nous avons séparé les expérimentations réalisées dans ce contexte en deux parties : d'une part, les variations concernant les capacités de la source de données (unique dans nos expérimentations), d'autre part, celles concernant les autres participants du système.

4.4.1 Variations côté source

Tout d'abord, intéressons-nous aux variations de capacité de la source. Ce participant particulier présente en effet la caractéristique d'être présent dans le système *dans le but* de permettre aux autres participants d'obtenir ses données. En d'autres termes, si les autres participants peuvent, ou non, choisir de se montrer altruiste, la source a *intérêt à l'être* pour atteindre ses objectifs. Nous pouvons donc faire l'hypothèse que, chaque fois qu'elle en aura la possibilité, une source choisira de se montrer altruiste et de fournir autant de ressources que possible au système.



(a) Nombre de mouvements

(b) Profondeur moyenne

(c) Cas d'une source moyenne

(d) Cas d'une source puissante

Figure 4.12: Expérimentation 4.3.1 bis : variations dues à la puissance de la source

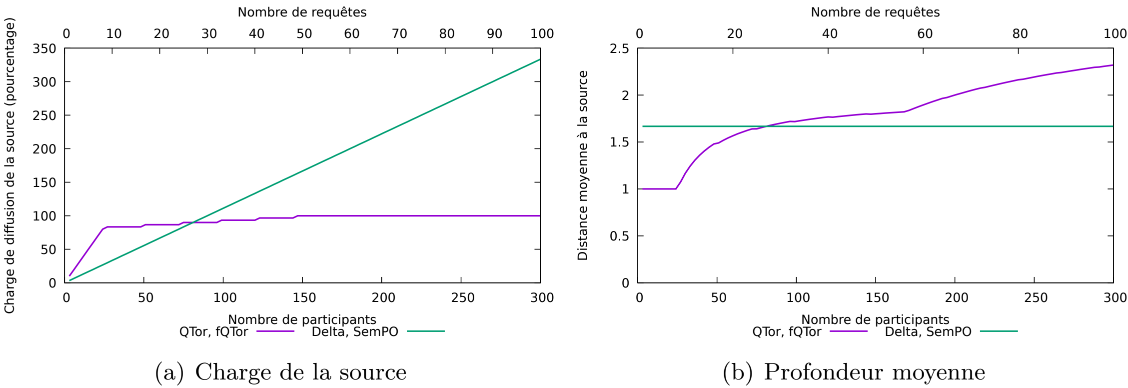
De ce fait, nous pouvons étudier le comportement du système en cas d'augmentation de la puissance de la source. Pour cela, nous avons repris les jeux d'expérimentation utilisés

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

en section 4.3.1, donnant cette fois à la source de données un degré sortant maximal de 500, lui permettant donc de servir directement un quart du système étudié. La figure 4.12 reprend les histogrammes présentés en figure 4.9 (a) et (b), pour les comparer aux résultats obtenus dans cette configuration. On constate que les résultats de Delta ne changent pas significativement[2]. En effet, l'un des objectifs de cette organisation est de décharger la source autant que possible : si le solveur linéaire peut la solliciter au maximum de ses capacités en cas de besoin, l'algorithme de réduction de latence n'est pas autorisé à ramener des participants directement auprès de la source. Il est donc normal que cette approche soit peu sensible aux variations de capacités de cette dernière.

Dans l'approche QTor, en revanche, comme l'illustrent les captures issues de la visualisation du simulateur fournies en figure 4.12, une source altruiste en mesure de fournir directement leurs résultats à la plupart des participants rejoindra les communautés concernées pour le faire. L'image (c) nous montre l'état obtenu pour jeu analogue à ceux utilisés pour les courbes, mais réduit à 200 participants. La source ayant la capacité de servir directement une partie importante du système, un grand nombre d'utilisateurs sont ses enfants directs. Ceux qui ne peuvent pas s'y connecter directement doivent pour leur part former une organisation communautaire plus classique. En (d), nous constatons que, lorsque la source est suffisamment puissante, QTor produit une structure du réseau analogue à celle de l'Unicast (2), quoique celle-ci soit ici obtenue sans outrepassements de capacités. Nous constatons donc que la distance moyenne à la source tombe à un dès lors que celle-ci a la capacité de servir directement l'ensemble du système.

À l'opposée, nous avons étudié la façon dont le système s'adapte aux risques d'outrepassement des capacités de la source lorsque celle-ci est de capacité ordinaire. Dans une approche basée sur les réécritures, ce risque correspond à celui d'avoir un nombre important de requêtes pour lesquelles aucune réécriture n'a pu être trouvée (même si elles peuvent potentiellement elles-mêmes servir de vues pour en réécrire d'autres).



(a) Charge de la source

(b) Profondeur moyenne

Figure 4.13: Expérimentation 4.4.1 : saturation des ressources de la source

Nous avons choisi d'expérimenter cette situation par un jeu présentant une source de puissance fixe (degré sortant maximal de 50) devant servir une centaine de requêtes (chacune exprimée successivement par trois participants) dénuées de toute connexité, donc incapables de se réécrire entre elles. L'absence de connexité a ici pour but d'observer la situation au niveau de la source elle-même, sans que les résultats obtenus (particulièrement

[2] Rappelons que cette organisation n'est pas déterministe, et que, de ce fait, des variations sont observées d'une expérimentation à l'autre même en réutilisant des jeux identiques. Les différences observées ici sont de cet ordre, malgré la variation de puissance de la source.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

en termes de profondeur) ne soient affectés par l'organisation des réécritures plus loin dans le graphe des communautés. Sur les courbes résultantes, présentées en figure 4.13, nous voyons l'évolution de la profondeur moyenne et du pourcentage de charge de la source à mesure que le nombre de participants (et donc de requêtes) augmente.

Dans une telle configuration, Delta n'a d'autre solution que de connecter un participant sur trois à la source, quelles que soient les limitations de celle-ci[3], de la même façon que SemPO. Nous voyons donc une profondeur qui reste constante, tandis que la charge augmente linéairement, dépassant les 100% dès lors que le nombre de requêtes dépasse les capacités de la source, et continue ensuite d'augmenter linéairement à mesure que de nouvelles requêtes arrivent.

Dans l'approche QTor, en revanche, nous voyons la capacité de la source augmenter d'abord plus rapidement, dans la mesure où elle envoie des unités dans les communautés enfants tant qu'elle dispose de suffisamment de ressources. Dans ces communautés enfants, elle gère l'ensemble de la diffusion, la distance moyenne restant alors de un. Dès que la source atteint 80% de ses capacités, elle conserve les ressources restantes pour sa propre communauté, et ce sont alors les participants des communautés suivantes qui doivent envoyer des ressources dans la sienne. Ces participants continuent cependant de pouvoir se connecter directement auprès de la source, et peuvent eux-mêmes envoyer des ressources dans les communautés arrivées après les leurs, ce qui explique que l'augmentation de charge de la source se fasse ensuite bien plus lentement, et que la distance moyenne reste relativement basse. Arrive toutefois un moment où la source n'a plus aucune capacité de disponible. Les participants suivants sont nécessairement placés à deux transferts minimum de la source, même s'ils intègrent sa communauté, ce qui explique l'évolution de la distance. Néanmoins, le mode d'organisation de QTor permet d'éviter tout risque de surcharge de la source.

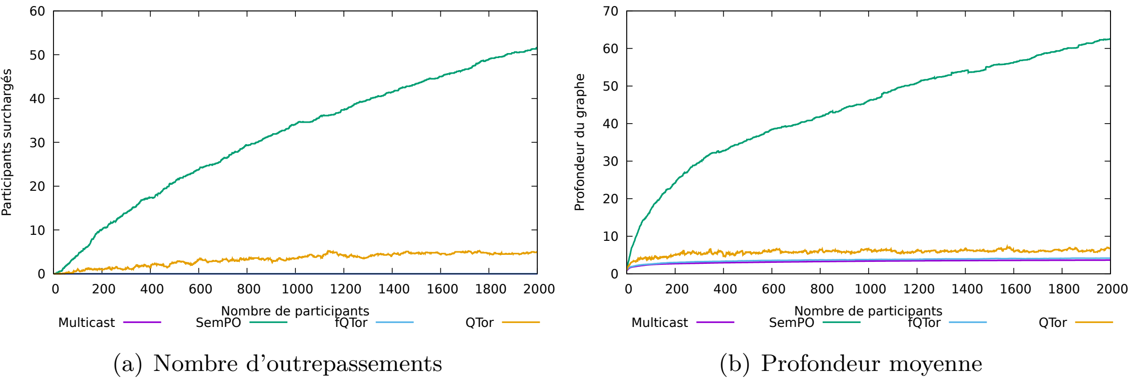
4.4.2 Variations côté requêteurs

Les participants requêteurs, pour leur part, ont pour responsabilité de gérer les aspects du système que la source ne peut prendre en charge, ce qui demande de disposer d'un minimum de capacités. Il est donc important de veiller à la façon dont le système se comporte lorsque la majorité de ses participants sont de capacité très réduite. Pour ce faire, nous avons effectué une expérimentation portant sur des jeux de 2 000 participants dont les requêtes suivent une loi de Zipf, mais dont les capacités sont nettement inférieures à l'ordinaire : celles-ci étaient tirées d'après une loi de Poisson autour de 5 (s'assurant toutefois que le degré sortant maximal n'était jamais nul).

Cette configuration, associée, pour QTor, au modèle de coût décrit en section 4.1.3, favorise l'apparition du problème décrit en section 3.3.2 (page 82) : une communauté dont le participant le plus puissant est doté de capacités très faibles (seulement deux ou trois connexions sortantes) peut se voir affecter une réécriture utilisant plus de vues qu'il ne peut en intégrer, sans que les communautés parentes ne soient en mesure de fournir d'aide. Comme le montre la figure 4.14, nous constatons toutefois que, même dans ce cas défavorable, le nombre de participants affectés reste faible (moins de cinq participants concernés sur 2 000 en moyenne). Cet effet est par ailleurs très fortement limité avec des

[3] Dans les faits, l'organisation de Delta finit par ne plus fonctionner, le respect des capacités étant une contrainte transmise au solveur linéaire. Ce dernier s'arrête en effet en provoquant une erreur lorsque le modèle qu'il tente de résoudre ne peut pas l'être. Une certaine tolérance sur la surcharge de la source est cependant acceptée dans l'approche, ce qui se traduit par la transmission au solveur d'une contrainte de capacité plus importante, d'où les résultats mentionnés.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

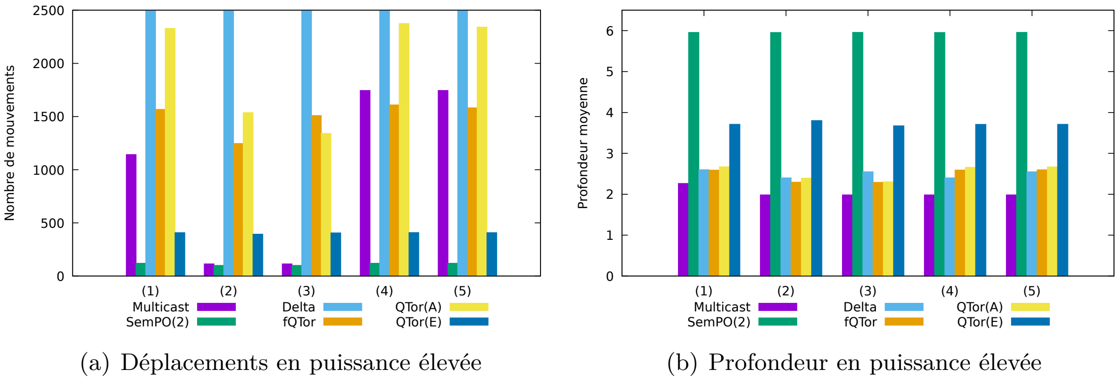


(a) Nombre d'outrepassements (b) Profondeur moyenne

Figure 4.14: Expérimentation 4.4.2 (a) : organisation en puissance limitée

modèles de coûts favorisant les réécritures à peu de vues, et totalement évité si l'on limite les relations possibles aux cas d'inclusion de requêtes.

Notons que Delta n'est pas présenté ici : sur ces configurations particulièrement sensibles, en effet, l'ILP, n'était pas autorisé à surcharger les participants, interrompait l'organisation sans avoir pu trouver de solution. Pour les autres approches, nous constatons des écarts relatifs assez semblables à ceux observés lors de l'expérimentation 4.2 (les requêtes étant obtenues de manière similaire), exception faite de SemPO. Nous présentons en effet ici la version de cette approche utilisant notre extension de prise en compte des capacités, car la configuration résultante se rapproche d'une organisation en communautés sans mécanisme d'échange de ressources. Les résultats obtenus s'expliquent donc à la fois par le fait que l'organisation des requêtes contraint fortement les possibilités de réduction de latence, et par des profondeurs supplémentaires importantes pour les participants exprimant des requêtes équivalentes. Nous constatons par ailleurs que cette extension ne suffit pas à éviter les surcharges, les relations de connexité entre requêtes n'étant pas impactées (ce qui entraîne assez souvent le fait qu'un participant soit déjà surchargé ou à la limite de l'être au moment où d'autres participants ayant une requête équivalente arrivent dans le système, réduisant d'autant les possibilités de répartition de la diffusion).



(a) Déplacements en puissance élevée (b) Profondeur en puissance élevée

Figure 4.15: Expérimentation 4.4.2 (b) : influence d'un participant puissant

Il convient par ailleurs d'envisager l'effet que peut avoir un participant de capacités élevées sur l'organisation du système. L'apport que peut avoir un tel participant

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

est naturellement affecté par le moment de son arrivée si l'organisation est sensible à ce paramètre. Dans le cas des systèmes basés sur les réécritures, cela peut également dépendre des relations de connexités de la requête concernée : cette requête aura en effet tendance à être placée d'autant plus près de la source qu'elle peut servir de vue à de nombreuses autres, et d'autant plus éloignée que de nombreuses autres peuvent lui servir de vue. Cette configuration influe donc sur les possibilités d'usage de ses ressources qui pourront être mises en place. Nous avons donc procédé à une expérimentation basée sur des jeux de 2 000 participants de capacités ordinaires (autour de 30), aux requêtes tirées d'après une loi de Zipf, dont les résultats sont présentés en figure 4.15. La première série de données de ces histogrammes présente le cas témoin, correspondant à l'organisation du système de base, aucun participant puissant ne se présentant. Dans les deux séries suivantes, un participant ayant un degré sortant maximal de 1 000 (lui permettant donc potentiellement d'envoyer des données à la moitié du système) arrive au début de l'expérimentation. Pour le second cas, sa requête est pertinente pour un grand nombre d'autres, mais reconnaît peu de vues comme pertinentes, et est donc placée à proximité de la source par les relations de connexité. Dans le troisième cas, elle est au contraire pertinente pour peu d'autres, mais reconnaît beaucoup de vues comme pertinentes, et est donc placée plus loin de la source. Les quatrième et cinquième séries reproduisent ces deux configurations vis-à-vis de la connexité des requêtes, mais en ne faisant arriver le participant qu'à la fin de l'organisation.

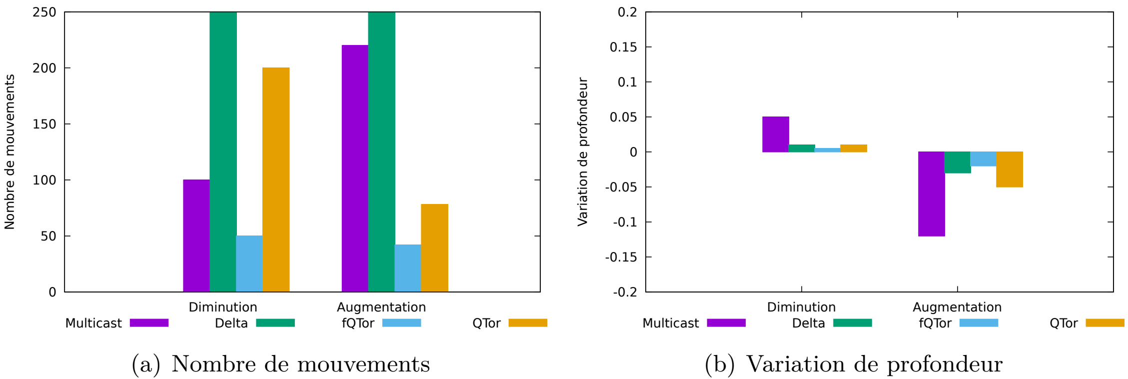
Pour cette évaluation, représentative du rapport entre capacités et connexités, nous avons rajouté aux systèmes étudiés précédemment la présence d'une deuxième version de notre approche. La variante ici présentée comme « QTor (A) » présente le cas, usuel jusqu'ici, dans lequel les participants ont tendance à se montrer altruistes, et donc à apporter, lorsqu'ils le peuvent, leur aide aux communautés qui dépendent des leurs. Dans la variante « QTor (E) », au contraire, les participants sont égoïstes, et refusent d'offrir cette aide lorsqu'elle leur est demandée.

Naturellement, nous constatons que la connexité de la requête exprimée par le participant puissant n'est d'aucune influence dans l'organisation du Multicast. Concernant SemPO, le peu de différences observées vient du fait que cette approche prévoit un strict respect des relations fixées par l'organisation des requêtes : dans un cas comme dans l'autre, le participant puissant n'est autorisé à envoyer ses données qu'aux participants effectuant des requêtes équivalentes, sans pouvoir contribuer au reste du système.

Dans le cas de Delta, les ressources d'un participant éloigné de la source ne peuvent pas être utilisées, dans la mesure où l'algorithme de réduction de latence ne peut exploiter que les réécritures calculées. Réciproquement, un participant proche de la source peut être davantage mobilisé, ce qui permet une appréciable diminution de latence, mais cette situation a pour contrecoup le fait qu'un grand nombre de participants concernés voient leurs charges de calcul augmenter du fait d'être ramenés à ce niveau. Enfin, le peu de variations entre les deux variantes de QTor dans le cas d'éloignement provient du fait que, dans le cas où le participant puissant est proche de la source, un refus de sa part de contribuer aux autres communautés conduit une partie importante de ses capacités à n'être pas mobilisées. L'envoi de ressources vers les communautés parentes, en revanche, rendue nécessaire si aucun participant de ces communautés n'accepte de prendre en charge l'échange, est toujours avantageux pour le participant, diminuant sa distance à la source en ne nécessitant que des calculs nécessaires à sa requête.

En complément de cette analyse se pose la question des changements dynamiques de capacités des participants. Il est cependant difficile d'envisager de situation type dans laquelle de tels événements se produiraient, ceux-ci dépendant généralement de facteurs

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés



(a) Nombre de mouvements

(b) Variation de profondeur

Figure 4.16: Expérimentation 4.3.3 bis : impact des changements de capacités

extérieurs au système. Le cas, évoqué au chapitre précédent, d'une variation de volumétrie du flux de la source pour un mot-clef donné peut être une piste à ce sujet, quoique les modifications de capacités dues à ce facteur pour des requêtes complexes (incluant ce mot-clef au sein de conjonctions ou de disjonctions) peuvent être délicates à estimer. Nous avons choisi d'évaluer un cas simple, pour lequel ces modifications sont localisées, en reprenant les jeux de l'expérimentation 4.3.3 pour y faire varier les capacités des participants d'une seule communauté, populaire et connectée à la communauté de la source. Les résultats observés, présentés en figure 4.16 peuvent donc former la première étape d'une modification de plus grande ampleur due à cette volumétrie, considérant que les communautés plus éloignées de la source réagiront à cette modification de volumétrie plus tardivement. Pour chaque jeu, nous étudions le cas d'une multiplication et d'une division des capacités d'origine par un facteur de 4.

Concernant l'augmentation de capacités, le fait que le Multicast présente un arbre de diffusion unique sur l'ensemble du système fait que les nouvelles ressources peuvent être aisément réutilisées, entraînant une diminution appréciable de la profondeur, mais au prix d'une certaine instabilité. La distance à la source augmente en revanche dans le cas des diminutions de capacités, mais les participants non-concernés par ce changement de capacités permettent de stabiliser l'arbre de diffusion. Pour les systèmes communautaires (QTor, fQTor), la partition du système en plusieurs communautés tend à diminuer l'importance des modifications, même si le fait de devoir changer plusieurs fois de tête de communauté lors des départs augmente le nombre de rebranchements indésirables (effet nettement moins observé pour fQTor du fait de l'absence de cascade, faisant qu'il n'y a aucune communauté enfant à impacter).

Conclusion de l'évaluation concernant les capacités

Ces diverses expérimentations nous permettent de conclure donc que l'approche que nous proposons, conformément aux conjectures formulées lors de l'analyse théorique, s'adapte efficacement aux différentes variations des capacités des participants, pouvant tirer profit de capacités élevées pour diminuer sensiblement la latence du réseau avec un coût de mise en place réduit. À l'inverse, les faibles capacités peuvent être efficacement prises en compte, quoique l'absence de mécanismes de répartition des tâches plus avancés puisse conduire à des outrepassements de capacités dans les configurations où le modèle de coût sélectionne des réécritures présentant de nombreuses vues. Nous remarquons égale-

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

ment que l'arrivée tardive d'un participant puissant est plus difficilement prise en compte, nos algorithmes actuels tirant profit des capacités déjà présentes au moment de l'arrivée de nouvelles communautés, mais ne remettant pas nécessairement en cause ces choix par la suite (ce qui, en compensation, assure une certaine stabilité du système).

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Chapitre 5

Conclusion

Les bouleversements de nos sociétés entraînés par l'essor du numérique en général et d'Internet en particulier induisent des problématiques nouvelles, mais également de nouvelles manières de répondre à des problèmes plus anciens. Nous avons, dans ce document, présenté quelques situations pour lesquelles de nouvelles réponses sont à apporter, du fait de ces changements, dans le domaine du traitement de requêtes et de l'échange de flux concernant de nombreux utilisateurs·trices. Puis nous avons détaillé les approches les plus représentatives à l'aune de nos connaissances actuelles pour répondre à un problème similaire au nôtre, et montré, après les avoir classifiées en fonction de leurs structures logiques, en quoi elles étaient insuffisantes pour répondre précisément au problème que nous soulevions. Nous avons alors proposé notre propre approche, à savoir la mise en place d'un système dans lequel les participants sont regroupés en communautés en fonction des intérêts communs représentés par leurs requêtes, et où ces communautés sont reliées entre elles en suivant les relations de connexité entre les requêtes, matérialisées par l'utilisation d'un mécanisme de réécriture de requêtes. Enfin, nous avons fourni les résultats expérimentaux obtenus lors de l'analyse de ce système, montrant que notre proposition offre un fonctionnement efficace pour des coûts d'organisation raisonnables. Quatre annexes fournissent des précisions concernant d'autres pistes d'organisation communautaire, les langages que nous avons jusque là expérimentés, le fonctionnement technique de notre démonstrateur et les possibilités d'étendre ce fonctionnement, et l'ensemble des entités et algorithmes utilisées dans la mise en place de notre système. Une cinquième annexe présente un résumé étendu de ces travaux sous format graphique.

Ce dernier chapitre est dédié à présenter nos conclusions sur l'état actuel des travaux, ainsi que les pistes d'évolutions futures que nous envisageons.

5.1 Synthèse des travaux réalisés

Comme cela a été mentionné, notamment en introduction, les travaux présentés dans ce document n'étaient pas effectués de manière isolée, mais s'inscrivaient dans un contexte plus général. Il convient donc, en faisant le point sur leur état d'avancement actuel, de les replacer dans ce contexte. Nous discutons ensuite des usages attendus de notre proposition et de l'état de la communication à ce sujet envers la communauté scientifique et le reste du monde.

5.1.1 Contexte et état d'avancement

Nous avons d'ores et déjà mis en place tout le nécessaire au bon fonctionnement d'un système d'échange de flux et de traitement de requêtes porté par ses utilisateurs·trices

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

fonctionnant sur un modèle communautaire. Les évaluations expérimentales réalisées en simulation autant que la mise en place d'un prototype réellement distribué montrent que notre approche est fonctionnelle et que son usage est intéressant.

La mise en place d'un système à torrent de requêtes tel que nous l'avons présenté dans ce document requiert cependant un certain nombre de caractéristiques particulières. Tout d'abord, cette organisation repose sur l'usage d'un *tracker* centralisé gérant l'ensemble des requêtes, qui peut apparaître comme une vulnérabilité. Toutefois, la plupart de nos algorithmes sont envisagés pour être aisément distribuables, et l'état actuel d'avancée des travaux à ce sujet nous donne bon espoir qu'une organisation totalement distribuée soit possible dès que les quelques écueils résiduels (à savoir, essentiellement la présence d'un mécanisme de synchronisation permettant d'éviter les déplacements simultanés de communautés) seront résolus.

L'approche étant portée par les ressources des utilisateurs·trices, il est nécessaire que l'ensemble des participants soient disposés à contribuer au système, sachant toutefois que les seuls calculs qui pourront leur être imposés en dehors de ceux propres à leurs propres requêtes sont ceux de requêtes intermédiaires dont les résultats leur sont requis. Pour tout autre traitement, leur accord préalable est requis.

Notre approche est générique vis-à-vis du langage de requêtes utilisé, dès lors que celui-ci fournit, au minimum, la possibilité de vérifier l'équivalence entre deux requêtes. Une organisation plus efficace est obtenue si ce langage permet également la mise en place d'un mécanisme de réécritures, le limitant possiblement aux relations d'inclusion de requêtes, mais l'organisation peut néanmoins fonctionner en son absence. Quoique ce mécanisme de réécritures, s'il est présent, conduise à diminuer appréciablement les coûts, il reste toutefois nécessaire qu'un participant ait les capacités de calcul suffisantes pour traiter seul la requête (réécrite si possible), le traitement distribué d'une transformation n'étant pas encore géré. Celui-ci fait toutefois l'objet de travaux séparés[54], parallèles à ceux présentés dans ce document, et nous avons bon espoir que notre modèle actuel soit rapidement complété sur ce point.

Étudiée en partie dans le cadre du projet ANR SocioPlug, visant à la mise en place de services distribués sur micro-ordinateurs de faible puissance (de type « plug »), notre approche tient fortement compte des capacités des participants. Il reste toutefois nécessaire que ceux-ci soient capables de communiquer entre eux, ce que les NAT[41] et autres réseaux fermés peuvent gêner. En l'état actuel des choses, il faut donc que toutes les machines sur lesquelles fonctionne le système puissent être jointes par le réseau, éventuellement en mettant en place de leur côté des mesures visant à dépasser les limitations imposées par ces réseaux fermés[93]. D'autres solutions [82, 91] peuvent être envisagées au niveau du système lui-même, mais elles nécessitent vraisemblablement de mettre en place un nouvel algorithme d'organisation interne aux communautés. Notons que le mécanisme de communication entre communautés fait que ce problème ne se produit pas à ce niveau.

Afin d'évaluer l'approche QTor, nous avons mis en place un démonstrateur, FleDDi, capable aussi bien de réaliser des simulations locales (sur un grand nombre de participants, dans le but de générer des statistiques, ou sur un nombre plus réduit, pour permettre une visualisation du système permettant d'en étudier la topographie) que d'être déployé pour offrir un fonctionnement réellement distribué.

Outre qu'il prouve le bon fonctionnement de notre approche, ce démonstrateur, portable sur de nombreuses plateformes et ne nécessitant pas de dépendances particulières, peut être utilisé comme base pour un déploiement de notre proposition en situation réelle. Il suffit pour cela d'implémenter les opérations propres au langage utilisé, si les langages

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

actuellement embarqués ne suffisent pas.

Mais, par sa structure générique, FleDDi peut également servir à tester et mettre en place d'autres approches que les nôtres fonctionnant de manière globalement similaire (expressions de requêtes par les participants auprès d'un *tracker* centralisé, qui leur transmet alors les tâches qu'ils auront à réaliser et les autres participants à qui ils devront envoyer leurs données ; puis publications de données par les sources et propagation dans le système). Nous avons ainsi pu tester en situation réelle d'autres approches que la nôtre, telles que *Semantic Peer-to-Peer Overlays for Publish/Subscribe Networks*[16] ou Delta[53] (cette dernière n'étant cependant pas intégrée dans le code public du démonstrateur, pour des raisons de licence et de portabilité liées au solveur linéaire utilisé).

5.1.2 Communication et usages

Quelques articles concernant ces travaux ont d'ores déjà été publiés au sein de la communauté scientifique, principalement liés à des présentations lors de conférences telles que Globe (DEXA, [33]) pour la communauté internationale et Bases de Données Avancées (BDA, [35, 34]) pour la communauté francophone. D'autres, dont au moins une démonstration et un papier revue, sont en cours de préparation ou de relecture au moment de la publication de ce document.

Mais la communication au sujet de nos travaux ne se limite pas au monde scientifique. Nous sommes en effet convaincus que notre proposition présente un intérêt important si les citoyen·ne·s s'en emparent pour mettre en place leurs propres moyens de communication. De ce fait, nous sommes en contact, notamment, avec des communautés d'utilisateurs·trices de logiciels libres, bien connues pour leur sensibilité aux problématiques auxquelles nous répondons. Une présentation de notre approche a ainsi été faite au festival Pas Sage en Seine (PSES) ; une autre est actuellement envisagée pour Capitole du Libre et devrait porter sur les enjeux d'une telle approche et son possible usage dans un système plus général de recherche d'informations.

La mise à disposition d'une application permettant de participer à un système à torrent de requêtes au sein d'un dispositif tel que la Brique Internet[93], ou tout autre système auto-hébergé même de faible puissance, permettrait en effet de prendre en charge efficacement le suivi de flux et l'appropriation de résultats publiés continuellement dans des cas d'utilisations tels que ceux décrits en introduction, où l'intérêt de réunir les utilisateurs·trices au sein de communautés d'intérêt mises en relations les unes avec les autres semble constitutif de l'intérêt global que peut présenter le système.

5.2 Pistes d'améliorations et perspectives

Les travaux présentés dans ce document ont ouvert la voie vers la mise en place d'un système complet et paramétrable. Toutefois, nous n'avons évidemment pas la prétention d'avoir résolu l'ensemble des problématiques possible, et il reste naturellement des pistes d'améliorations qui pourront faire l'objet de travaux futurs afin de compléter notre approche sur différents aspects. Nous envisageons en particulier trois pistes principales pour lesquelles, quoique nous n'ayons pas encore eu l'occasion de dépasser formellement le stade de la conjecture, il nous semble que notre approche offre des pistes d'étude intéressantes.

5.2.1 Généralisation du modèle

En premier lieu, le modèle logique du système communautaire présenté en section 3.2.1 (pages 65 et suivantes) n'est pas spécifique à l'organisation que nous proposons ici :

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

l'utilisation d'un système de réécritures pour guider l'organisation des communautés est un choix distinct, indépendant de celui-ci et de même importance dans le cadre général de notre proposition. Rien n'empêche, donc, d'envisager la mise en place d'un système communautaire pour lequel l'organisation des calculs reposerait sur un fonctionnement radicalement différent, si celui-ci s'avère plus adapté à une situation donnée.

Une telle approche nécessiterait vraisemblablement de proposer des algorithmes différents pour ce qui concerne l'organisation abstraite du système (recherche d'une communauté déjà existante traitant une requête équivalente à celle nouvellement insérée, et recherche des communautés en relation avec la nouvelle créée le cas échéant, que ce soit pour connecter celle-ci ou pour réorganiser le système), mais, d'une part, cela ne remettrait aucunement en cause les mécanismes de concrétisation (et particulièrement le mécanisme d'échange de ressources entre communautés), qui garantissent l'adaptabilité de notre approche aux capacités des participants. D'autre part, les bénéfices du passage par une abstraction (particulièrement la réduction de la taille du problème et la possibilité de s'affranchir des problématiques de limitation de capacités, mais également la diminution du nombre de cycles possibles du fait du regroupement des équivalences) demeureraient les mêmes quel que soit le mode d'organisation choisi pour cette abstraction.

L'organisation interne des communautés étant découplée de l'organisation des communautés entre elles, d'autres modèles d'organisation peuvent également être envisagés pour ces dernières. En plus de la distribution des calculs déjà évoquée[54] ou de la possibilité de remplacer l'organisation de la diffusion que nous proposons par la mise en place de n'importe quel autre système à miroir, au sens présenté en section 2.3.1 (pages 36 et suivantes), plusieurs pistes peuvent être étudiées. Il est notamment possible d'envisager la mise en place de plusieurs systèmes d'acquisition et de calcul indépendants, assurant une redondance des opérations appréciable lorsqu'elle est délibérément choisie.

Comme mentionné, l'annexe un fournit des précisions concernant la façon dont l'organisation interne d'une communauté peut être généralisée, notamment aux cas où chacune des tâches peut être distribuée séparément.

5.2.2 Extensions de l'organisation

Dans le même ordre d'idées, il est envisageable d'autoriser une communauté, dans un système basé sur les réécritures, à effectuer simultanément plusieurs transformations distinctes équivalentes à sa requête d'origine. Une telle mesure augmenterait naturellement les coûts de calcul au sein du système, puisque chaque calcul indépendant du même flux de résultat serait redondant, mais permettrait d'assurer, notamment, une meilleure vérification des données utilisées lorsque la confiance envers les communautés parentes n'est pas garantie (attendu que la probabilité que plusieurs communautés indépendantes mentent sur leurs résultats d'une manière qui conduit à la génération du même flux de sortie après application des deux transformations devient plus faible). Il est par ailleurs envisageable que cette augmentation des coûts, dans un système peuplé d'un nombre suffisant de participants, ne fasse que mobiliser des ressources de calcul laissées inoccupées par les gains de l'organisation communautaire.

Une possibilité intéressante à considérer ici est qu'une telle approche conduirait à une manière assez différente d'envisager les cycles. En effet, la présence d'une configuration circulaire entre deux communautés n'aurait pas le même caractère critiquement bloquant si chacune de ces deux communautés obtenait également, et simultanément, ses résultats par une autre voie ne présentant pas de cycle. Cela pourrait ouvrir la porte à des relations plus étroites entre communautés, s'échangeant mutuellement des résultats en cours de

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

vie du système sans remettre en cause leur activité extérieure. On pourrait ainsi voir apparaître un concept de « communautés de communautés » (ou « méta-communautés »), concernant les requêtes « équivalentes » *à une transformation réversible près* (par exemple le cas, évoqué lors de l'analyse des cycles simples en section 3.4.2, page 88, des calculs de somme et de moyenne sur des valeurs dont le nombre est connu).

On peut par ailleurs envisager le fait que certaines communautés, même dans une configuration où la connexité est importante et permet de nombreuses réécritures, continuent de travailler sur des calculs redondants, dans le cas où une requête qui leur fournirait une base intermédiaire commune n'a pas encore été exprimée dans le système. Par exemple, sur un langage de filtres conjonctifs, deux communautés traitant respectivement « $A \wedge B$ » et « $A \wedge C$ » pourraient gagner à se brancher toutes deux sur une communauté traitant « A »… à condition que celle-ci ait préalablement été créée.

La découpe des requêtes du système en sous-requêtes et autres calculs intermédiaires propre aux systèmes à organisation spécifique à un langage donné permet de tenir efficacement compte de ce genre de situations, ce qu'un système à réécritures classique ne permet pas. Il est cependant possible d'envisager de prendre en compte cet aspect de manière générique, en rajoutant une fonctionnalité à l'API que nous demandons au langage de requêtes. Cette fonctionnalité serait naturellement optionnelle et, comme pour le mécanisme de réécritures lui-même, l'organisation s'adapterait à sa présence ou à son absence, mais elle permettrait, étant donné deux requêtes, de déterminer leur plus grande partie commune, qui pourrait être ajoutée à la liste des vues considérées. Pour le cas où les réécritures correspondantes seraient sélectionnées, les participants des communautés concernées auraient alors pour charge de créer artificiellement une nouvelle communauté, dédiée à la vue supplémentaire et peuplée uniquement de leurs unités acquisitrices, mais qui pourrait ensuite être rejointe par des participants exprimant ultérieurement cette requête. Cette situation formerait ainsi un symétrique de la situation où, ses participants originaux ayant retiré leurs requêtes, une communauté est maintenue en vie de manière artificielle par les participants de communautés en dépendant, via le mécanisme d'échange de ressources. Certains des travaux que nous avons étudiés, tels que RoSeS[89], offrent des pistes intéressantes à ce sujet.

Quoique des communautés autonomes soient susceptibles d'évaluer l'utilité de créer ces communautés artificielles de manière autonome, ce type de modifications de l'hypergraphe abstrait peut également relever de la mise en place de la fonction d'optimisation. Nous avons, lors de l'analyse des coûts de fonctionnement du système, indiqué qu'une telle fonction, qui reste encore à définir, pourrait être utile dans le cas où l'apparition de cycles conduit à la perte d'optimalité du système. Proposer une telle fonction nécessite de déterminer précisément dans quelle situation l'appeler, ce qui peut se faire en évaluant l'ampleur que peut prendre dans ce cas cette perte d'optimalité. Toutefois, étendre le modèle à la création artificielle de communautés augmenterait vraisemblablement le nombre de cas où l'appel à une telle fonction d'optimisation pourrait s'avérer utile. Il convient donc d'étudier les possibles impacts à ce niveau. Un aspect intéressant est que, dans les deux cas, l'appel à cette fonction peut être envisagé concernant un nombre réduit de communautés, et non nécessairement l'ensemble du système.

Par ailleurs, la question de la découverte dynamique de sources mérite elle aussi d'être posée. Celle-ci peut présenter deux aspects complémentaires : d'une part, il est possible d'envisager que les utilisateurs·trices expriment des requêtes sans connaître au préalable les sources de données qui les intéressent, quoique celles-ci soient tout de même considérées comme faisant partie du système. Il est alors nécessaire de mettre en place un mécanisme de recherche qui puisse prendre un point de départ autre que les communautés sans hy-

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

perarc entrant, et qui soit donc capable de remonter vers les communautés parentes pour obtenir une vision globale du graphe.

D'autre part, le langage peut permettre d'exprimer une *wildcard* sur les sources de données, faisant qu'une source de données arrivant tardivement dans le système pourra s'avérer pertinente pour des requêtes préalablement insérées. Il est donc nécessaire, dans cette configuration, d'envisager les modifications du graphe correspondant à l'arrivée de cette nouvelle source, c'est-à-dire des déplacements partiels : pour les communautés qui devront être branchées à celle de la nouvelle source, il faudra remplacer l'hyperarc actuellement sélectionné par un autre constitué d'un sur-ensemble de ses sommets de départ. Par ailleurs, cette configuration, entraînant pour les requêtes un certain degré d'indépendance vis-à-vis des sources de données, permet d'envisager la conduite à tenir lors du départ d'une source de données, qui rendait de fait le système inopérant dans la configuration classique.

Enfin, l'existence de plusieurs communautés distinctes traitant la même requête a été considérée jusque là comme un problème de détection, conduisant à la fusion de ces communautés dès que l'équivalence entre elles est détectée. Toutefois, il est possible d'envisager que cette situation puisse également résulter d'un choix, lorsque plusieurs participants, quoique s'intéressant à des requêtes équivalentes, ne sont pas d'accord sur la meilleure manière de procéder à leur traitement ou à la diffusion des résultats correspondants. Il serait donc intéressant d'étudier les conséquences d'une fission de communauté, qui pourrait être envisagée comme un moyen d'augmenter l'autonomie des participants, dans l'optique de permettre au système une organisation de classe 4 dans la classification Bortzmeyer[12].

5.2.3 Intimité numérique et contrôle d'accès

Nous avons, à plusieurs reprises, évoqué dans ce document les problématiques d'intimité numérique (posées, notamment, par la présence d'un *tracker* centralisé récupérant la liste exhaustive des requêtes du système). L'intimité numérique a également été envisagée comme une contrepartie possible à un comportement altruiste des participants leur faisant contribuer à la bonne vie du système. Quoique cet aspect n'ait pas encore été étudié formellement, il nous semble en effet que la mise en place d'un système à torrent de requêtes entièrement distribué, y compris dans son organisation, est susceptible d'apporter certaines garanties intéressantes.

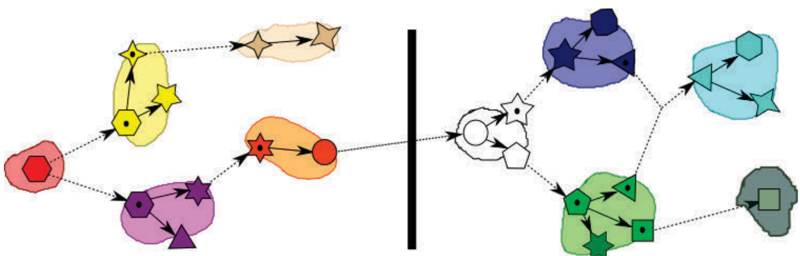


Figure 5.1: Exemple d'enchaînement de deux systèmes QTor

En l'état actuel des choses, il est possible d'envisager que deux systèmes QTor se succèdent, comme illustré en figure 5.1. Un participant au premier système doit pour cela récupérer les données désirées dans l'un et se placer en situation de source pour l'autre. Il peut aussi être la seule interaction entre les deux systèmes, empêchant les intervenants de l'un d'obtenir des informations sur l'autre. Il est donc possible, dans le cas où un groupe

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

de participants souhaiterait manipuler certains flux de manière isolée, de conserver les autres avantages propres à notre approche. Cette configuration empêchant cependant de procéder à des optimisations entre les deux systèmes, nous étudions également d'autres possibilités.

Envisageons à ce sujet le cas d'un participant souhaitant s'insérer dans le graphe sans que les requêtes qui l'intéressent ne soient connus de tous. En l'absence d'un *tracker* centralisé au niveau global, ce participant devrait procéder lui-même à l'exploration du graphe, et ainsi contacter les différentes communautés déjà présentes. Il n'aurait, à aucun moment, besoin de leur indiquer pour quelle requête il cherche à être ancré : c'est au contraire les différentes communautés qui l'informeraient de la requête sur laquelle elles travaillent, information supposée publique.

Le participant pourrait alors effectuer les opérations de vérification de pertinence et d'équivalence en local, sans qu'aucun tiers n'intervienne. Bien sûr, un même participant servant d'interlocuteur pour un nombre important de communautés différentes pourrait être en mesure, voyant par les demandes successives quelles vues sont parcourues et quelles autres abandonnées, de faire des suppositions concernant la requête ; mais un participant souhaitant éviter ce cas pourrait choisir, pour un surcoût modéré (correspondant au cas où le langage ne fournit pas de possibilité de vérification de pertinence), d'interroger toutes les communautés du graphe, plutôt que d'élaguer son parcours, afin de ne dévoiler aucune information.

Dans le mode de fonctionnement développé au chapitre trois, dans lequel la connaissance interne d'une communauté est globale, chaque membre de la communauté intégrée verrait le participant intégrer celle-ci. Toutefois, la possibilité d'échanges de ressources, à la fois pour les communautés liées entre elles et pour les communautés distantes, mais se signalant comme nécessitant de l'aide, fait qu'il reste toujours une incertitude sur les motifs pour lesquels un participant donné peut rejoindre une communauté : il peut y être aussi bien par intérêt personnel que par altruisme, et parfois pour un mélange des deux.

Notons d'ailleurs qu'un participant ne souhaitant pas que les requêtes qui l'intéressent soient connues pourrait également rejoindre différentes communautés sans lien avec ses requêtes, quand bien même celles-ci ne demanderaient pas explicitement d'aide, auquel cas il fournirait bel et bien des ressources au système en échange. Le fait que l'organisation soit ouverte et la liste des communautés connues lui permettrait, par ailleurs, de procéder à des recherches de réécritures en local, et de n'intégrer aucune communauté qui corresponde exactement à ses attentes, mais d'extraire celles-ci des résultats qu'il recevrait à l'extérieur du système.

À l'inverse, un participant ayant pour but de se renseigner sur les motivations des autres membres du système pourrait chercher à intégrer autant de communautés que possible, afin d'avoir une connaissance précise du système tendant vers l'exhaustivité. Toutefois, cette entité n'aurait alors aucun monopole sur cette connaissance, l'organisation demeurant suffisamment ouverte ; et, comme on vient de le voir, cela n'empêcherait pas les participants souhaitant masquer leurs intérêts de mettre au point des stratégies particulières en ce sens.

Il convient d'ailleurs de noter que, comme mentionné dans l'algorithme 2 (page 81), la possibilité peut être laissée aux participants de refuser l'arrivée dans une communauté d'une unité dont le but ouvertement affiché est de fournir de l'aide, sans s'intéresser elle-même aux résultats. Un tiers centralisateur pourrait donc être refusé à ce titre, si les membres de la communauté préfèrent travailler sur leurs requêtes eux-mêmes.

Plus généralement, une organisation du type de celle de QTor peut permettre, en plaçant des barrières à l'entrée des communautés, de fonctionner avec un certain niveau de contrôle d'accès. Ainsi, un participant s'intéressant à certains résultats en particulier,

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

mais ne disposant pas des autorisations requises pour y accéder directement, pourrait se voir refuser d'entrer dans la communauté les manipulant, et contraint d'intégrer une autre communauté, *a priori* située immédiatement derrière l'autre, ne travaillant que sur des résultats filtrés pour être accessibles au plus grand nombre sans contraintes.

Dans une telle configuration, bien évidemment, l'envoi de ressources vers une communauté parente serait rendu impossible, conduisant à des risques de surcharge ; mais les participants des communautés gérant les données sensibles pourraient monter fournir les résultats filtrés aux communautés enfants.

Synthèse générale

Dans le domaine général des systèmes répartis de traitement de requêtes continues sur des flux, l'approche que nous proposons est basée sur l'organisation du système en *communautés d'intérêt*, chaque communauté regroupant les requêtes identifiées comme équivalentes. Ces communautés sont ensuite organisées entre elles par l'utilisation d'un mécanisme de réécritures de requêtes, leurs relations étant matérialisées par un envoi de ressources assuré par l'un des participants concernés.

L'analyse théorique de cette approche a conduit à mettre en lumière certaines propriétés intéressantes, notamment le fait qu'elle permet d'atteindre une optimalité des coûts de calcul des requêtes dans tous les cas où le mécanisme de réécriture ne présente aucun risque de cycles. Les expérimentations réalisées ont permis de confirmer que cette approche présente une organisation efficace, capable de s'adapter aux différentes variations envisagées. Le démonstrateur développé pour réaliser ces expérimentations comporte également un prototype permettant un déploiement en conditions réelles.

Si plusieurs des pistes d'améliorations envisagées nécessitent d'importants travaux de recherches, d'autres sont bien avancés et pourraient prochainement faire l'objet de stages pour des étudiants de master. Notons parmi eux la mise en place d'une fonction d'optimisation pour les cas où le système de réécritures conduit à perdre l'optimalité du fait de risques de cycles, la mise en place de mécanismes de contrôle d'accès sur les requêtes, ou encore l'étude des effets obtenus par l'enchaînement de plusieurs systèmes à torrent de requêtes. Plusieurs travaux importants de la littérature peuvent également être réutilisés pour améliorer certains aspects de nos travaux, tels que RoSeS[89] pour l'extension du mécanisme de réécritures ou SplitStream[14] pour l'organisation interne aux communautés.

Dans l'ensemble, l'état d'avancée de nos travaux nous conduit à considérer que l'approche QTor apporte une réponse intéressante au problème auquel nous nous intéressons ici, et nous encourage à poursuivre les travaux à ce sujet.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Annexes

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Annexe 1

Organisation communautaire

Cette annexe est consacrée à approfondir les possibilités d'organisation interne d'une communauté. Pour cela, nous commençons par présenter une généralisation du modèle présenté en section 3.2.1 et discuter les propriétés qui en découlent ; avant de présenter davantage de pistes explorées pour l'organisation de l'arbre de diffusion interne, les propriétés qui en découlent, et les raisons pour lesquelles nous avons finalement opté pour le modèle plus simple présenté au chapitre trois.

1.1 Modélisation interne d'une communauté

Nous présentons ici une version totalement générique de la définition d'une communauté, acceptant la présence de plusieurs unités par participant (tant qu'elles sont dédiées à des tâches distinctes) et pour laquelle chacune des tâches considérées peut être distribuée. Une telle situation déjà était présentée, à titre de simple exemple rapidement écarté, dans la figure 3.8. Nous reproduisons celle-ci ici :

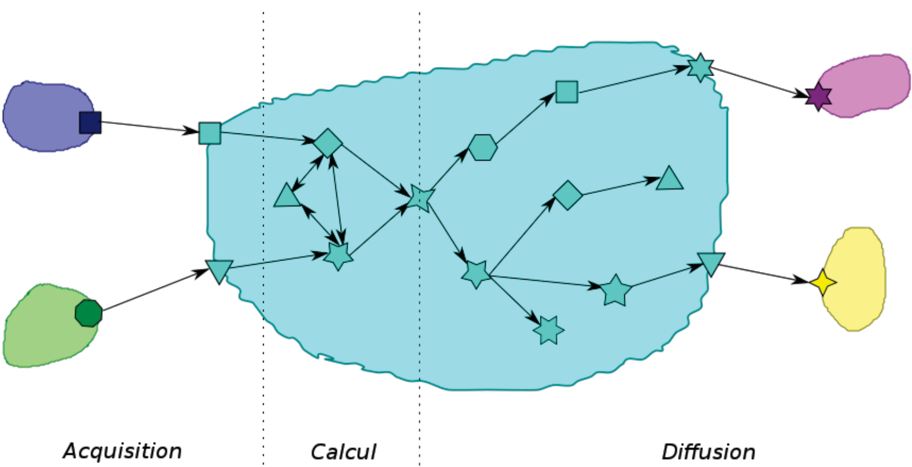


Figure 3.8: Possible répartition des tâches au sein d'une communauté

Nous y voyons donc deux participants acquérir les données, l'un en étant également intégré dans la communauté parente correspondante, l'autre en communiquant avec un autre participant du réseau. Ils transmettent ensuite leurs données à trois participants chargés d'effectuer le calcul de manière distribuée, puis une unique unité (qui assume le rôle de tête de communauté dans le modèle plus simple décrit au chapitre trois) se charge de réunir leurs résultats et de constituer le flux attendu, qui est ensuite partagé sur toute

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

la communauté. On remarque que tous les participants mobilisés pour l'acquisition et le calcul disposent d'une seconde unité chargée de récupérer le flux entier.

1.1.1 Répartition des tâches

Il existe donc trois tâches que doit gérer chacune des communautés : l'*acquisition* des données utilisées, le *calcul* de la requête proprement dite (ou du moins, de la transformation correspondante), et la *diffusion* des résultats obtenus. L'ensemble U_r des unités de la communauté est donc *a priori* constitué par l'union de trois sous-ensembles : U_a, U_c et U_d. On pourra toutefois, pour une définition totalement générique, y rajouter un ensemble U_i des unités *inactives*, destinées à récupérer le flux de résultat sans pour autant contribuer directement au fonctionnement de la communauté :

$$U_r = U_a \cup U_c \cup U_d \cup U_i$$

Les unités d'acquisition $u \in U_a$ sont caractérisées par le fait qu'elles récupèrent les flux des communautés parentes. L'exemple fourni en figure 3.8 nous présente deux unités récupérant chacune le flux d'un parent donné et le transmettant inchangé, mais d'autres situations sont possibles (division d'un flux parent pour alimenter plusieurs unités dédiées au calcul, réunion de plusieurs flux parents en un seul…). Quelle que soit la façon dont la tâche est répartie, ces unités doivent récupérer les flux de l'ensemble des parents de la communauté :

$$\{u_{tin}, u \in U_a\} = c_{tin}$$

Les unités $u \in U_c$ en charge du calcul proprement dit peuvent appliquer n'importe quelle solution de calcul distribué pour se répartir le calcul entre elles ; ou bien une unique unité peut prendre en charge ce calcul de manière locale. Dans tous les cas, il est nécessaire qu'au moins une unité soit en mesure de récupérer le flux de résultat correspondant au traitement considéré, reconstitué à partir des différents traitements effectués :

$$\exists u \in U_c : u_{tout} = c_{tout}$$

L'une au moins de ces unités est alors chargée de servir de source secondaire pour la tâche de diffusion (interne et externe) de la communauté. Dans le cas de notre modèle simplifié présenté au chapitre trois, ce rôle correspond à celui de la tête de communauté h, chargée seule de réaliser les traitements ($U_c = \{h\}$). On notera que, dans le cas où l'acquisition et le calcul sont distribués, rien n'oblige l'ensemble des unités en charge de ces tâches à manipuler le flux de résultat de la communauté : leur rôle peut être uniquement de gérer des flux intermédiaires.

L'objectif de la tâche de diffusion est que, d'une part, tous les participants intéressés par la requête reçoivent les résultats concernés ; d'autre part, que toutes les unités d'acquisition des communautés enfant récupèrent le flux pour leurs besoins propres. Pour cela, toutes les unités de U_d peuvent être mobilisées en fonction de leurs capacités et des besoins, chacune (en dehors des unités de $U_d \cap U_c$ servant de sources secondaires) servant de *miroir* pour les résultats communautaires :

$$\forall m \in U_d \setminus U_c, m_t : \{c_{rout}\} \mapsto c_{rout}$$

Les unités de U_i, en revanche, doivent recevoir le flux sans contribuer au fonctionnement de la communauté. En d'autres termes, elles sont nécessairement en position de feuilles dans l'arbre de diffusion, ou équivalent selon le mécanisme utilisé. Cela peut concerner, notamment, les participants fournissant suffisamment de ressources de calcul pour être

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

dispensés de contribuer à la diffusion et dont, pour autant, aucune des unités de calcul ne gère le flux de résultat entier, du fait d'une forte distribution des calculs.

En l'absence d'un mécanisme d'échange de ressources inter-communautaire tel que celui que nous proposons, leur situation correspond à celle des unités acquisitrices issues des autres communautés, qui elles aussi récupèrent le flux sans contribuer à la communauté. Dès lors qu'il existe au moins une communauté enfant à la communauté considérée, il existe au moins une telle unité acquisitrice :

$$\forall c' \in C, c \rightarrow c' \Leftrightarrow \exists u \in U'_a, c_{rout} \in u_{tin}$$

Afin de simplifier les notations, nous considérerons donc un ensemble étendu U_e devant récupérer les flux (qui, contrairement aux précédents, n'est pas strictement limité aux unités de la communauté) :

$$U_e = U_d \cup U_i \cup \{u \in U'_a \forall c' \in C, c \rightarrow c', c_{rout} \in u_{tin}\}$$

1.1.2 Conséquences sur les propriétés du système

Cette optique plus générique de la modélisation communautaire nous permet de préciser les conditions de validité d'un système communautaire :

Propriété 4 (Validité d'un système communautaire détaillée)**.**
Un système communautaire est valide si et seulement si :

- *Le système abstrait aux communautés* $\langle C, \rightarrow \rangle$ *est valide,*
- *La couche de calcul* U_c *de chaque communauté permet d'obtenir les résultats de la requête* c_r *à partir des flux récupérés par la couche d'acquisition* U_a*,*
- *Pour chaque communauté, toutes les unités de* U_e *reçoivent les résultats de la requête, s'appuyant sur les ressources des unités de la couche de diffusion* U_d*.*

Cette définition, quoique plus détaillée, est strictement équivalente à celle proposée dans la propriété 2 (page 67), avec cependant une légère différence de répartition :

- Le fait que la communauté forme un sous-système valide (3e point dans la propriété 2) correspond au fait que les résultats soient correctement produits (2e point dans la propriété 4) et diffusés (3e point).
- Le fait que les relations entre communautés soient concrétisées au niveau des participants (2e point dans la propriété 2) correspond au fait que les unités d'acquisition soient correctement intégrées dans les arbres de diffusion correspondants (3e point dans la propriété 4).

Cette seconde définition se concentre donc davantage sur l'organisation interne de la communauté, permettant de vérifier la validité globale du système de manière plus précise lorsqu'acquisition et calcul sont distribués, ou bien dans le cas où l'absence d'un mécanisme d'échange de ressources tel que nous le proposons fait que l'acquisition n'est pas gérée par un trivial échange local entre deux unités d'un même participant.

Notons toutefois qu'en l'absence d'échange systématique de ressources entre communautés, une communauté enfant ne rembourse plus nécessairement à la communauté parent les ressources de diffusion qu'elle mobilise pour acquérir le flux. De ce fait, il n'est plus

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

possible de considérer la communauté parente comme étant de ressources non-limitées : celle-ci peut se trouver débordée lorsque le nombre de ses enfants devient trop important.

Cela entraîne des conséquences notables sur l'organisation du système, qui peuvent être de deux sortes en fonction de la stratégie choisie. En effet, choisir de ne pas prendre en compte ce problème au niveau de l'organisation logique fait que l'on peut rendre certaines communautés dépendantes d'une communauté n'ayant pas les ressources suffisantes pour les servir, ce qui conduit à ne plus respecter les contraintes de capacité, et donc à aboutir à un système dont l'utilisation sera au minimum problématique.

En revanche, choisir de prendre en compte ce problème au niveau de l'organisation logique revient à y intégrer des aspects physiques (que sont les capacités de diffusion), et donc à perdre une partie des avantages apportés par la montée en abstraction. D'une part, l'optimalité au niveau des charges de calcul peut être perdue même dans les cas reposant sur l'inclusion de requête, dans la mesure où un branchement permettant de diminuer les coûts pourra ne pas être utilisable du fait du manque de ressources.

D'autre part, les capacités d'une communauté dépendant de celles des participants qui la composent, toute apparition ou disparition de participant pourra avoir des répercussions sur le graphe abstrait, même lorsqu'ils n'entraînent aucune création ou suppression de communauté. Ne pas considérer les changements à ce niveau entraînera un éloignement d'autant plus important vis-à-vis de l'optimalité.

1.2 Diffusion tenant compte des communautés enfants

Nous avons opté, dans la proposition exprimée au chapitre trois, pour une organisation interne des communautés aussi simple que possible, afin notamment de nous focaliser sur les avantages structurels propres à l'approche QTor elle-même, se détachant autant que possible des effets dûs à telle ou telle organisation interne spécifique. Toutefois, nos travaux préliminaires à ce sujet envisageaient d'autres manières de procéder, et notamment un classement légèrement plus complexe, tenant compte d'une notion de *poids* pour les unités liées aux communautés enfants.

Nous présentons ici cette notion et les raisons ayant motivé son utilisation, avec quelques-uns des effets intéressants que nous espérions pouvoir en attendre ; avant d'expliciter les raisons pour lesquelles nous avons fini par l'abandonner pour en venir à la proposition décrite dans le reste de ce document.

1.2.1 Notion de poids d'une unité

Notre proposition initiale était de gérer l'organisation interne d'une communauté en considérant, pour chaque unité concernée, deux caractéristiques : d'une part, les capacités de diffusion dont elle dispose[1], et d'autre part, le nombre de participants qui vont dépendre directement ou indirectement d'elles. Plus formellement,

Définition 18 (Poids d'une unité)**.**
Une unité $u \in U_e$ est caractérisée par un poids, déterminé comme suit :

- *Si l'unité u est chargée d'acquérir les données pour une communauté c' telle que $c \to c'$, son poids est égal à la somme des poids des unités de la communauté c'.*

[1] Cela concerne ici uniquement les capacités de diffusion, attendu que, sous l'hypothèse qui domine en ce document de calcul non-distribué, les capacités de calcul sont gérées séparément, au moment de sélectionner la tête de communautés. Dans ce contexte, le point étant traité à part, l'organisation de la communauté concerne uniquement les aspects de diffusion.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

- *Sinon, son poids est de un.*

L'objectif visé par la prise en compte d'une telle caractéristique est à la fois de diminuer la latence globale, et d'offrir une contrepartie à la prise en charge par les participants du fonctionnement du système. En effet, donner la priorité, pendant l'organisation de l'arbre de diffusion, aux unités en fonction de leur poids permet de diminuer la latence de tous les participants situés plus loin dans l'arbre de diffusion.

Pour un participant issu d'une communauté enfant, envoyer une unité dans la communauté parente lui assurera de descendre avec un certain poids, et donc d'être placé d'autant plus bas dans l'arbre. Et ce d'autant plus qu'il n'y a de communautés concernées : descendre avec la capacité de prendre la position de tête de la communauté parente permet de prendre une place d'autant plus près de la source, et ainsi de suite. Assurer que chaque communauté s'organise en tenant compte des poids permet donc de donner aux participants les plus éloignés de la source la possibilité, s'ils disposent des capacités suffisantes, de bénéficier (et de faire bénéficier leur communauté) d'un meilleur placement en mettant ces capacités au service du reste du système. L'effet d'un tel mécanisme est cependant réduit dans le cas où les communautés peuvent utiliser différents types d'organisation interne, dans la mesure où cela implique que le participant n'a pas forcément la garantie que son poids lui assurera une bonne place dans la communauté parente.

Réciproquement, un participant originaire de la communauté parente est d'autant plus encouragé à envoyer ses ressources aux communautés enfants que cela lui permet de gagner un poids plus important, et donc de descendre d'autant plus dans l'arbre de diffusion de sa propre communauté, pour réduire sa propre latence. Contrairement à l'autre cas, le gain est ici davantage assuré, dans la mesure où le participant prenant cette décision est déjà intégré dans la communauté dans laquelle son poids va conséquemment augmenter, et est donc assuré que l'organisation communautaire prendra cette augmentation en compte, même si l'organisation interne des autres communautés peut différer.

Ce mécanisme de prise en compte des poids assure donc une souplesse importante entre les charges (principalement de calcul, celles de diffusion pouvant être utilisées à plein au sein d'une même communauté) et la latence : sans remettre en cause l'organisation logique commune, décidée grâce à un modèle de coût commun, chaque participant peut réévaluer ses propres objectifs pour favoriser l'un ou l'autre.

1.2.2 Effet sur la latence globale

Afin d'obtenir les bénéfices sus-mentionnés, il est donc nécessaire de définir une organisation physique tenant compte autant des poids que des capacités. Conformément à la logique de fonctionnement de QTor, cette organisation se fait au niveau communautaire :

Propriété 5 (Optimalité locale de latence)**.**
Une communauté c basée sur un arbre de diffusion déployé sur U_e est considérée comme optimale d'un point de vue latence lorsqu'il n'existe aucune configuration possible de cet arbre de diffusion pour laquelle la somme $\Sigma(u_w \times u_d \forall u \in U_e)$, où u_w représente le poids de l'unité u, et u_d sa distance dans l'arbre à la tête de communauté h, est inférieure.

Cette propriété peut aisément se généraliser dans le cas où l'arbre ne dispose pas d'une tête de communauté unique : il faut alors utiliser la distance à la racine concernée dans la forêt interne de diffusion. Toutefois, sa généralisation aux cas où la diffusion repose sur un mécanisme différent (par exemple, une organisation basée sur SplitStream[14]) peut s'avérer plus problématique et doit être étudiée plus précisément.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Quoiqu'il ne repose que sur une organisation locale, nous pouvons montrer qu'un tel mode de fonctionnement peut s'avérer particulièrement efficace au niveau global :

Théorème 4 (Optimalité globale de latence)**.**
Étant donné une organisation des communautés donnée dans laquelle chaque communauté n'a qu'un seul parent, l'optimalité locale de chacune de ces communautés d'un point de vue latence entraîne une optimalité globale de la latence.

Preuve. Considérons une portion de système constitué d'un nombre variable de communautés qui est globalement optimal d'un point de vue latence, et étendons cette portion de système étudiée en ajoutant chacune des communautés parentes n'étant pas encore considérées. Pour chacune de ces communautés, minimiser la somme $\Sigma(u_w \times u_d \forall u \in U_e)$ revient à placer les unités chargées d'acquérir les données pour la portion de système déjà optimal au plus près possible des sources, en fonction des unités présentes dans la communauté parente. Étant donné qu'un système constitué uniquement de communautés sans liens les unes avec les autres est nécessairement optimal d'un point de vue latence dès que chacune de ces communautés est optimisée localement, le système complet est donc, par récurrence, optimisé de manière globale.

Notons que cette optimalité, dans ce cas, est spécifique à une organisation logique donnée : il peut être possible d'aboutir à une situation présentant une meilleure latence en disposant les communautés différemment. Cependant, modifier la disposition des communautés provoque la plupart du temps une augmentation des coûts de calcul. Il est d'ailleurs intéressant de noter que, dans le cadre d'un système basé sur l'inclusion de requête, remplissant les conditions dans les deux cas, un tel mode d'organisation fait coïncider l'optimalité des calculs avec celle de la latence.

En revanche, dès lors qu'une communauté peut avoir plusieurs parents distincts, cette optimalité au niveau global n'est plus garantie, dans la mesure où l'une des unités correspondantes peut alors se trouver prioritisée par rapport à d'autres dans une communauté, alors que son placement sera nécessairement contraint par la position des autres unités dans d'autres communautés. La figure 6.2 illustre cette situation.

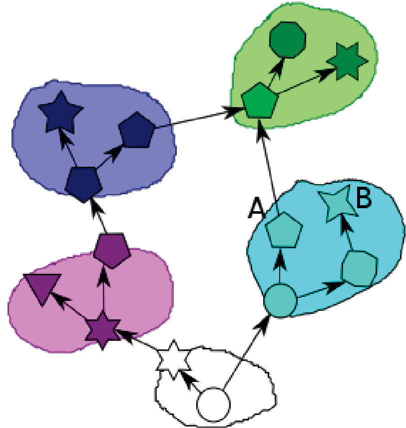


Figure 6.2 : Perte d'optimalité de latence en cas de branchements multiples

Dans cette configuration, l'unité `A`, du fait du poids de la communauté verte à laquelle elle est liée, est connectée directement à la tête de la communauté cyan, qui correspond ici au participant gérant la source de données. Pour autant, la latence de la même communauté verte est contrainte par le fait que celle-ci récupère également des données issues

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

de la communauté bleue, dans laquelle sa latence, tenant également compte du poids, est contrainte à trois sauts de la source. De ce fait, inverser sa position avec celle de l'unité B pourrait diminuer la latence de celle-ci sans impacter le reste du système.

1.2.3 Problématiques de mise en place

Pour mettre en place une telle organisation au niveau d'une communauté, notre proposition était d'utiliser une procédure analogue à celle décrite dans l'algorithme 3 (page 83), mais pour laquelle les unités ne sont pas ordonnées en fonction des capacités seules, mais de la combinaison des capacités et des poids.

Dans le cas où le mécanisme d'échange de ressources entre communautés n'est pas systématiquement présent, toutefois, il est nécessaire de veiller à ne pas se trouver avec un « étage » de l'arbre de diffusion dénué de capacités de diffusions s'il reste des nœuds à placer. Il convient donc de légèrement modifier cet algorithme pour que, lors de la récupération des n prochains nœuds destinés à constituer l'étage suivant, le dernier d'entre eux puisse, le cas échéant, être écarté au profit d'un nœud situé plus loin dans la liste ordonnée, mais doté de capacités non-nulles.

Ordonner correctement les nœuds peut cependant s'avérer problématique, dans la mesure où capacités et poids sont souvent contradictoires : les participants ayant le poids le plus important sont ceux qui travaillent dans des communautés comptant un nombre important de membres ou de descendants. Ils ont alors également tendance à voir leurs capacités de diffusions fortement utilisées dans ces communautés, et donc à être d'autant plus limitées dans les communautés dans lesquels leur poids est important.

Pour nous assurer d'obtenir une organisation optimale à ce niveau, une possibilité était d'utiliser l'algorithme $A*$. En effet, si le but originel de cet algorithme[48] est de trouver le plus court chemin dans un graphe, son mode de fonctionnement lui permet en fait d'être utilisé dans un contexte beaucoup plus général, dans le but de trouver l'état terminal le plus proche à partir d'un état initial donné, se basant sur une évaluation du coût du chemin parcouru et sur une heuristique. Si cette heuristique est optimiste, la solution trouvée est alors nécessairement optimale.

Dans notre cas, nous pouvons considérer l'état initial comme celui dans lequel la liste des nœuds à connecter (ici notée l) est vide, et tout état dans lequel elle contient tous les nœuds qui devront être pris en compte par l'algorithme 3 modifié (soit $U_e \setminus U_c$) comme un état terminal, passer d'un état à un autre correspondant à ajouter l'un des nœuds non-encore pris en compte dans cette liste. L'évaluation du coût du chemin parcouru correspond alors à la somme $\Sigma(u_w \times u_d, \forall u \in l)$, la distance u_d se déterminant simplement par l'organisation en étages. L'heuristique, pour sa part, doit correspondre à l'évaluation du coût attendu pour les sommets restants.

Le code que nous avons initialement mis en place à ce sujet s'est cependant avéré particulièrement lent sur les organisations complexes, du fait qu'une organisation optimale demande ici de placer les étapes les plus coûteuses relativement tôt dans le parcours, ce qui n'est pas exploré en priorité.

Indépendamment de l'algorithme utilisé, cependant, ce mode d'organisation présente un problème de fonctionnement majeur : il entraîne le fait que l'organisation interne d'une communauté donnée dépend fortement de l'évolution (en termes de nombre de participants) de chacune de ses communautés enfants.

En effet, chaque nouvel arrivant dans une communauté donnée entraîne une augmentation du poids des unités acquisitrices correspondantes (et réciproquement, chaque

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

départ une diminution). Dès lors, deux possibilités s'offrent au système. D'une part, ce changement peut ne pas être pris en compte immédiatement, mais attendre qu'une réorganisation communautaire se produise pour une raison directe (arrivée ou départ dans la communauté parente) pour réévaluer l'ensemble des poids et réorganiser la communauté en fonction d'eux. Dans ce cas, cela revient à reproduire les inconvénients d'une organisation globale périodique : la plupart du temps, le système ne sera pas optimal. Dans un système dans lequel les requêtes les plus éloignées des sources sont très populaires et les plus proches nettement moins, l'organisation optimale ne sera que très rarement atteinte.

D'autre part, ce changement peut être pris en compte systématiquement. Cela ne présente pas de difficultés techniques dans le cas d'un système organisé par un *tracker* centralisé ; en revanche, la propagation de l'information dans un système entièrement distribuée peut s'avérer plus délicate à mettre en place. Mais l'inconvénient principal est alors l'ampleur des modifications : chaque nouvel arrivant, même si ses capacités ne suffisent pas à obtenir des gains substantiels tels que ceux décrits en section 3.4.4, possède une propension élevée à entraîner des réorganisations en cascade, du fait de la modification des poids remontant de proche en proche jusqu'aux communautés sources.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Annexe 2

Langages et jeux de données

Notre approche ayant été conçue pour fonctionner, de manière générique, sur tout langage fournissant l'API détaillée au chapitre trois (section 3.2.2, page 67 et suivantes), il était nécessaire de disposer d'au moins un langage implémentant fidèlement cette API pour pouvoir effectuer des tests. Les systèmes de gestion de base de données relationnelle classiques embarquent un mécanisme de réécritures dédié au langage SQL qu'ils utilisent pour la description des requêtes ; mais celui-ci, interne au moteur d'évaluation des requêtes auxquels il est très lié, est rarement mis à disposition de façon autonome. De plus, le langage SQL (et plus encore ses extensions plus appropriées au traitement de flux, tels que CQL[8]) offre des possibilités assez avancées, notamment de jointure, qui n'auraient, dans un premier temps, fait que complexifier inutilement l'analyse de notre système. Nous avons donc choisi de reposer sur un langage qui serait aussi simple que possible tout en offrant des possibilités suffisantes pour tester les possibilités offertes par notre mode d'organisation du système.

Cette annexe est dédiée à présenter les travaux que nous avons effectués concernant les différents langages de requêtes utilisés dans notre démonstrateur, et plus particulièrement la façon dont sont implémentés les fonctions de notre API. Nous évoquons également la façon dont nous générons les requêtes et données utilisées dans les expérimentations, en fonction des différents langages utilisés. Au cours de nos travaux, trois langages ont été étudiés successivement : un langage simple de filtres conjonctifs par mots-clefs, son extension à la possibilité d'exprimer des requêtes disjonctives, et un langage à la syntaxe inspirée de celle du SQL pour effectuer des requêtes sur des flux issus de capteurs.

2.1 Filtres par mots-clefs et combinaisons conjonctives

Nos premiers travaux reposaient sur l'utilisation d'un langage de filtres conjonctifs, de quelque nature que puissent être ces filtres (nous envisagions, notamment, le cas des filtres d'analyse d'images, comme ceux évoqués dans *shared filters*[69], pouvant être coûteux lorsqu'ils sont appliqués sur, par exemple, des flux vidéos). Afin de pouvoir générer des données de test efficacement et lancer des simulations sans nécessiter de ressources trop importantes, nous avons choisi d'utiliser une implémentation d'un tel langage basé sur des mots-clefs à rechercher dans un texte.

2.1.1 Analyse des requêtes et réécritures

Nos premières requêtes étaient donc constituées d'une liste de mots-clefs séparés par un caractère délimiteur, le plus souvent une tabulation. Dans ces conditions, la vérifi-

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

cation d'équivalence entre deux requêtes était une opération triviale, dans la mesure où il suffisait de comparer deux listes d'objets simples après les avoir ordonnées et en avoir retiré les éventuels doublons. S'il est possible d'estimer à l'avance la sélectivité des filtres, l'ordre se fait en priorité d'après ce critère ; sinon, un ordre naturel (par exemple, l'ordre lexicographique[1] pour les mots-clefs) est utilisé.

Dans les premières étapes de nos travaux, nous nous limitions aux relations d'inclusion de requêtes, rendant également triviale la mise en place d'un système de réécritures : une vue est considérée comme pertinente pour réécrire une requête si l'ensemble de ses filtres correspond à une partie de l'ensemble des filtres de la requête ; et la requête réécrite correspondante est constituée de la différence ensembliste (Ainsi, la requête constituée des filtres $\{A, B, C\}$ pourra se réécrire à partir de la vue constituée des filtres $\{A, B\}$, et il ne restera plus qu'à traiter $\{C\}$).

Assez rapidement, toutefois, nous avons envisagé d'ajouter à ce mécanisme simple la possibilité de combiner plusieurs vues. Pour ce faire, nous utilisons l'algorithme 4 :

Algorithme 4 Réécritures conjonctives

1: **Entrées :** une requête q et un ensemble de vues V
2: **Sorties :** un ensemble de réécritures R
3: **Variables :** une liste de listes de vues L, une liste de vues l,
4: une vue v, une liste de filtres F.
5:
6: R est initialement vide, L contient une liste vide.
7: **Pour chaque** vue v de V, **faire :**
8: **Pour chaque** liste l de L, **faire :**
9: **Si** q contient l'ensemble des filtres de $l + \{v\}$, **alors :**
10: F reçoit la différence entre q et cet ensemble.
11: Ajouter dans R la réécriture appliquant F sur le résultat de $l + \{v\}$
12: **Si** $F \neq \emptyset$, **alors :**
13: Ajouter $l + \{v\}$ dans L
14: **Fin si**
15: **Fin si**
16: **Fin pour**
17: **Fin pour**
18: **Renvoyer** R

Cet algorithme permet donc de tester toutes les combinaisons de filtres susceptibles de servir à réécrire la requête, en s'arrêtant lorsqu'il devient inutile d'ajouter de nouvelles vues car l'ensemble des filtres de la requête d'origine est couvert. Quoique ces tests ne soient pas précisés ici par mesure de simplicité, il peut être interrompu à chaque ajout d'une nouvelle réécriture dans R si le nombre maximal de réécritures demandé est atteint, ou à chaque nouvelle itération de la boucle principale si le délai maximal est dépassé.

Lorsque deux vues ou plus sont mobilisées, le traitement de la requête revient à une intersection : un filtre spécifique est destiné à compter le nombre d'occurrences de chaque tuple reçu, et de ne le valider que lorsqu'il a été obtenu autant de fois qu'il n'est attendu. Chaque tuple disposant d'un identificateur unique, cette opération est considérée comme

[1] L'ordre *lexicographique*, considéré comme l'ordre naturel des chaînes de caractères dans la plupart des applications, notamment les SGBDr, est souvent désigné à tort comme étant l'ordre *alphabétique*. Celui-ci, utilisé dans le dictionnaire, présente un fonctionnement plus complexe, dans lequel plusieurs caractères (notamment toutes les accentuations d'une même lettre) ont le même niveau de priorité.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

peu coûteuse ; quoiqu'il soit possible d'ajuster le modèle de coût de telle sorte que le choix se porte plus fréquemment sur l'usage d'une seule vue parente.

2.1.2 Requêtes et données utilisées

Outre leur simplicité, les filtres conjonctifs par mots-clefs présentent également l'avantage de pouvoir être générés simplement. Il suffit en effet, après avoir déterminé le nombre de mots-clefs que contiendra la requête de manière aléatoire (suivant, par exemple, une loi de Poisson), de tirer les différents mots-clefs parmi l'ensemble de ceux attendus. Ce tirage peut se faire de manière équiprobable, ce qui peut notamment s'avérer pratique pour des tests aléatoires : il est facile de générer un ensemble de tuples permettant d'évaluer toutes les combinaisons possibles sur un ensemble d'une petite dizaine de mots-clefs sans grande difficulté. Pour une situation plus réaliste, telle que celle que nous cherchons à obtenir dans nos simulations, il est préférable d'établir une certaine popularité sur les mots-clefs, dont découlent popularité et connexité des requêtes. Pour cela, nous avons recours à une répartition suivant la loi de Zipf[65], simulant la répartition réelle de mots dans un texte.

Pour ce qui concerne les données, nos simulations utilisent des tuples de taille réduite, ne contenant que leur propre combinaison de mots-clefs sans texte parasite, afin de gagner en efficacité. Ces jeux de données sont générés en même temps que les requêtes et suivant la même loi de répartition (Zipf ou équiprobabilité) afin de garantir un fonctionnement cohérent. Toutefois, nous avons également travaillé sur la lecture de jeux de données plus réalistes, utilisant par exemple des flux RSS (en recherchant les mots-clefs dans le champ « description » de chaque item).

Nous avons notamment travaillé un certain temps avec l'ensemble de flux RSS collectés lors d'une étude du CNAM[49], fournissant un jeu de données réelles appréciable ; mais en complément duquel il manquait un jeu de requêtes réelles. Plusieurs jeux de requêtes sont en effet disponibles dans la littérature, mais ceux-ci présentent quelques difficultés vis-à-vis de notre situation, comme notamment le fait de ne pas être associés à des informations de popularité[2] ou le fait d'être sélectionnés spécifiquement dans le but d'avoir la connexité la plus faible possible.

Nous avons, durant un temps, envisagé de constituer ce jeu de requêtes à partir du jeu de données, utilisant pour extraire les mots-clefs un reconnaisseur d'entités nommées tel que celui de l'Université de Stanford[40], considérant que, dans les cas où ils dépendent de l'actualité (un certain nombre des flux RSS collectés étaient ceux de journaux en ligne), flux et requêtes se font sur des sujets communs. Toutefois, nous avons fini par abandonner cette idée, dans la mesure où le jeu ainsi constitué n'apporte pas une grande valeur ajoutée par rapport à une génération entièrement synthétique, à plus forte raison lorsque l'on envisage d'étendre le langage au cas des filtres disjonctifs.

2.2 Filtres par mots-clefs et combinaisons disjonctives

Afin de compléter notre langage, nous avons en effet ajouté la gestion des combinaisons disjonctives. Il s'agit d'une extension du langage précédent, qui n'est pas fournie séparément, dans la mesure où il suffit de n'exprimer que des requêtes entièrement conjonctives pour retrouver le comportement précédent. Cette extension, toutefois, amène son lot d'éléments spécifiques.

[2] Nous avons tenté, sur un jeu de requêtes, de retrouver cette popularité à l'aide d'un robot interrogeant Google Trends (comparant chaque mot à un référent commun) ; mais éviter le passage en liste noire par Google demande de mettre en place des délais rendant cette extraction peu intéressante.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

2.2.1 Analyse des requêtes et réécritures

Une augmentation des possibilités offertes allant de pair avec une augmentation de la complexité, des requêtes pouvant être à la fois conjonctives et disjonctives ne peuvent plus être représentées par une simple liste de filtres, mais ont besoin de structures légèrement plus complexes. Dans leurs expressions, il devient nécessaire de préciser quelles relations ont les mots-clefs entre eux, ce qui nécessite de spécifier les délimiteurs utilisés. Pour les requêtes générées automatiquement, nous avons choisi les caractères représentant les opérateurs « et logique » ($\wedge$) et « ou logique » ($\vee$). Pour taper manuellement des requêtes avec une disposition clavier rendant difficile l'usage de caractères rares comme ceux-ci, nous autorisons également l'usage de « `AND` » et « `OR` » en majuscules. Les parenthèses sont autorisées pour préciser la priorité des opérations.

L'une des caractéristiques principales des combinaisons disjonctives, par rapport aux combinaisons conjonctives, est d'inverser l'ordre des relations de pertinence. En effet, si la requête « $A \wedge B \wedge C$ » peut être calculée à partir de « $A \wedge B$ », c'est au contraire « $A \vee B$ » qui peut être calcul à partir de « $A \vee B \vee C$ ». Une autre de ces caractéristiques, qui nous intéressait particulièrement, était le risque de faire apparaître des cycles dans les réécritures, que le langage ne permettait pas jusque là. En effet, les requêtes « A » et « B », par exemple, peuvent toutes deux être réécritures d'après « $A \vee B$ ». Or, celle-ci peut être réécrite en combinant ces deux vues. Une telle configuration n'est pas possible dans un langage purement conjonctif. Les cycles étant un élément à éviter impérativement dans l'organisation, nous avions besoin de situations où ils risquaient de se produire pour vérifier qu'ils étaient correctement évités.

L'algorithme 4 continue de pouvoir être utilisé dans ce contexte, à condition de l'appeler sur des requêtes passées en forme normale conjonctive (soit des conjonctions de disjonctions), et à condition d'élargir la définition des routines de manipulation des ensembles de mots-clefs (considérant par exemple que la conjonction « A » contient « $A \vee B$ », et que la différence entre les deux est A). Toutefois, cet algorithme ne permet pas de prendre en compte toutes les possibilités du langage, raison pour laquelle nous lui avons ajouté l'algorithme 5, destiné pour sa part à être utilisé sur des requêtes en forme normale dis-

Algorithme 5 Réécritures disjonctives

1: **Entrées :** une requête q et un ensemble de vues V
2: **Sorties :** un ensemble de réécritures R
3: **Variables :** une liste de listes de vues L, une liste de vues l,
4: une vue v, une liste de filtres F.
5:
6: R contient les réécritures précédentes, L contient une liste vide.
7: **Pour chaque** vue v de V, **faire :**
8: **Pour chaque** liste l de L, **faire :**
9: **Si** q contient l'ensemble des filtres de $l + \{v\}$, **alors :**
10: F reçoit la différence entre q et cet ensemble.
11: Ajouter dans R la réécriture appliquant F sur le résultat de $l + \{v\}$
12: **Sinon si** $l + \{v\}$ contient l'ensemble des filtres de q, **alors :**
13: Ajouter $l + \{v\}$ dans L
14: **Fin si**
15: **Fin pour**
16: **Fin pour**
17: **Renvoyer** R

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

jonctive (soit des disjonctions de conjonctions).

Cette fois encore, cet algorithme peut en pratique être interrompu à l'ajout de chaque réécriture dans R (si le nombre maximal est atteint) ou à chaque nouvelle itération de la boucle principale (si le délai maximal est dépassé). Dans cette configuration, nécessitant d'évaluer davantage de vues avant de procéder aux réécritures, ce second mode d'interruption peut devenir intéressant, alors qu'il était à peu près inutile pour le mode conjonctif. Ces deux algorithmes sont appelés successivement (dans l'ordre dans lequel ils sont présentés ici) par le système de réécritures.

Toutefois, le mécanisme de réécritures n'est plus ici le seul point à considérer : les autres étapes d'analyse de requêtes deviennent également plus complexes, du fait de l'augmentation des possibilités. Il est d'ailleurs possible, dès l'expression initiale des requêtes, de procéder à quelques optimisations : notre *parser* commence par appliquer, sur la requête directement traduite de l'expression textuelle, une mise en forme « semi-normale », consistant en le fait d'aplanir la requête (les parenthésages inutiles sont supprimés), de supprimer les éventuels doublons et d'ordonner les filtres. Ensuite, les formes normales conjonctives et disjonctives de cette forme « semi-normale » sont calculées, et la moins coûteuse de ces trois formes (suivant les indications du modèle de coût) est considérée comme la requête de base du participant.

Étant donné que cette approche autorise plusieurs formes différentes pour les requêtes équivalentes (il n'est pas exclu que plusieurs formes « semi-normales » différentes soient de coût moindre que les formes normales conjonctives et disjonctives identiques auxquelles elles correspondent), la vérification d'équivalence commence par un passage en forme normale conjonctive avant d'effectuer les comparaisons. Il est cependant possible de désactiver cette opération, dans le but d'induire une certaine probabilité de production de faux négatifs, afin de vérifier le comportement du système dans ce cas.

Pour ce qui concerne la vérification de pertinence, les deux formes normales sont utilisées : une vue est considérée comme pertinente dès lors que la requête « contient » la vue en forme normale conjonctive, ou que la vue « contient » la requête en forme normale disjonctive, au sens étendu de la comparaison ensembliste évoqué plus haut.

2.2.2 Requêtes et données utilisées

Une difficulté introduite à ce niveau concerne la génération de requêtes synthétiques en situation réaliste. En effet, générer une liste de filtres suivant une loi de Zipf, puis ajouter aléatoirement l'un des deux opérateurs entre chacun de ces filtres diminue considérablement la popularité des requêtes, dans la mesure où les requêtes constituées de filtres identiques ont beaucoup moins fréquemment les mêmes opérateurs. L'équivalence entre requêtes générées de cette manière se fait donc beaucoup plus rare.

Pour pallier ce problème, nous avons considéré que les deux opérateurs logiques étaient rarement utilisés conjointement de manière importante. Nos requêtes synthétiques sont donc générées de telle sorte qu'une majorité d'entre elles ne contiennent qu'un seul des deux opérateurs, et que le reste contienne un opérateur principal utilisé la plupart du temps, et l'autre de manière occasionnelle. Associé à des paramètres de la loi de Zipf accentuant la popularité, cela nous ramène à une répartition de la popularité proche de celle que nous utilisions dans le cas purement conjonctif, tout en augmentant le nombre de relations de connexité à considérer.

En revanche, le tirage équiprobable des opérateurs utilisés fonctionne bien lorsqu'il est utilisé de pair avec le tirage équiprobable des mots-clefs sur un ensemble réduit, en autorisant les doublons. Ce cas, en effet, n'est pas destiné à obtenir des situations néces-

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

sairement réalistes, mais à vérifier le comportement du système sur des jeux de taille réduite, ou lorsque les expressions de requêtes utilisées sont très variables. La présence de doublons dans les mots-clefs permet alors un usage efficace du passage en forme « semi-normale », et le système résultant fournit une popularité satisfaisante.

Ce langage est celui que nous avons utilisé lors de la mise en place de notre premier prototype réellement déployé, se basant toujours, pour les jeux de données, sur l'usage de flux RSS. Nous avons ainsi pu utiliser le jeu collecté par le CNAM[49] pour dérouler rapidement des expérimentations autonomes, tout en vérifiant le fonctionnement général en utilisant les véritables flux RSS de journaux en ligne, en temps réel. Pour notre second prototype, toutefois, nous avons préféré passer à un langage différent.

2.3 Langage pseudo-SQL pour flux issus de capteurs

Nous disposons en effet d'un certain nombre de capteurs (température, humidité, taux de CO_2, luminosité, etc.) dont nous pouvons utiliser les données, et il nous a semblé judicieux d'utiliser ces capteurs pour mettre en place un prototype en conditions réelles utilisant un langage de requêtes plus habituel au monde des bases de données, reposant sur une syntaxe proche de celle du SQL.

2.3.1 Données utilisées et syntaxe des requêtes

La plupart des capteurs que nous pouvons utiliser sont conçus pour exprimer les données dans un certain intervalle et avec un pas donné. Ainsi, les capteurs de température supportent jusqu'à 60° et sont sensibles à des variations de 0,1°. Certains autres capteurs (détecteur de mouvement ou d'état d'ouverture de porte) ne fournissent que des informations booléennes (vrai ou faux). Ces capteurs étant regroupés entre eux, une même source de données peut produire, par exemple, des informations de température et d'humidité, ou de luminosité et de taux de CO_2.

Afin de faire en sorte que les tuples ne contiennent que les informations variables et utiles, nous avons fait en sorte que les métadonnées statiques concernant chaque source de données (modèle, emplacement…) soient renseignées une unique fois dans l'expression de la source. Il suffit alors d'interroger le *tracker* pour connaître la liste des sources présentes dans le système et pouvoir en extraire ces méta-informations, avant d'exprimer une requête permettant de récupérer les données elles-mêmes.

Dans ce nouveau langage, l'expression d'une source ressemble donc à :

```
SOURCE(c1 INTERVAL[-10:20:0.5], c2 INTERVAL[0:10]) @meta1=v1, meta2=v2
```

Seule la partie précédant l'arobase (ou, plus pragmatiquement, la partie entre les parenthèses) est interprétée par le *parser* utilisé par le système, le reste ne concernant pas directement le système. Afin de simplifier les définitions autant que d'améliorer la sémantique globale, nous avons défini autant de types de données que nos capteurs en géraient. Ainsi, le type « `HUMIDITY` » est un alias pour « `INTERVAL[0:100:0.1]` ».

La syntaxe des requêtes, elle, est conçue pour suivre une version simplifiée de la norme SQL. Une requête peut ainsi s'intéresser à l'ensemble des champs (`SELECT *`) ou bien à certains champs particuliers (`SELECT c1, c2`) d'une source donnée (`FROM s`), unique car ce langage simplifié ne gère pas encore les jointures. Lorsque plusieurs sources utilisent des schémas compatibles, il est cependant possible d'exprimer un `UNION ALL` pour obtenir leurs résultats dans une seule requête.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Naturellement, ce type de requêtes serait très limité s'il n'était possible d'y adjoindre, au minimum, une clause `WHERE`. Pour celle-ci, nous avons implémenté les opérateurs permettant de comparer la valeur d'un champ à une constante donnée (donc, les opérations $=, \neq, <, >, \leq$ et $\geq$, pour lesquels les alias usuels $==, !=, <>, <=$ et $>=$ peuvent également être utilisés). D'autres opérations sont envisagées dans l'avenir, mais cette combinaison nous semblait suffisante pour une première version, d'autant que les résultats obtenus peuvent être persistés dans une base de données, dans laquelle il devient possible d'effectuer des requêtes utilisant le langage SQL complet.

Une requête dans notre langage pseudo-SQL pourra donc ressembler, par exemple, là :

```
SELECT temperature FROM s1 WHERE temperature > 20;
```

(Le point-virgule terminal étant optionnel).

2.3.2 Mécanisme de réécritures

Sur de telles requêtes, la plus grosse partie de l'analyse concerne la clause `WHERE`. Or, il se trouve que celle-ci évalue une expression booléenne, dont les différents atomes correspondent à des filtres. De ce fait, les mécanismes développés dans le cadre du langage précédent, conçus pour être appliqués de manière générique sur n'importe quel langage de filtres, ont pu être réutilisés de manière très similaire.

Quelques ajustements ont toutefois pu être réalisés concernant la normalisation des filtres. En effet, les opérateurs d'inégalité peuvent présenter des relations entre eux, tandis que les filtres précédemment envisagés étaient indépendants. À titre d'exemple, le filtre « `temperature > 10` » contiendra par exemple le filtre « `temperature > 20` », ce qui nécessite d'être pris en compte au moment de la vérification des ensembles de filtres, dans les algorithmes précédemment présentés. De même, savoir qu'un champ donné présentera des valeurs par pas de 0,1 permet de savoir que, sur ce champ, des tests comme « `> 5` » et « $\geq$ `5.1` » seront équivalents.

Enfin, il a fallu prendre en compte le fait que deux filtres soient partiellement ou totalement opposés, conduisant à ce que leur combinaison n'amène aucun résultat, ou au contraire, ne soit plus sélective du tout (par exemple, le filtre « `x = 1` » s'oppose totalement à « `x` $\neq$ `1` », mais seulement partiellement à « `x > 10` »). Ce cas n'était pas encore géré jusque là, du fait de l'absence de négation logique ($\neg$).

La mise en place de ce langage pseudo-SQL a donc nécessité d'enrichir notre langage de filtres générique, sans pour autant nécessiter la mise en place de nouveaux algorithmes spécifiques. Les mécanismes de vérification d'équivalence et de pertinence, ainsi que de calculs de réécritures, consistent sur ce langage à vérifier la compatibilité des schémas (définis par les clauses `SELECT` et `FROM`, le schéma des sources de données initiales étant connu), avant de faire appel aux mécanismes pour filtres génériques sur la clause `WHERE`.

Quoique quelques jeux automatiques aient été mis en place pour tester ce fonctionnement (présentant, notamment, l'ensemble des sélections possible sur une source de peu de champs dont les intervalles de valeurs sont réduits), nous n'avons cependant pas encore mis en place de génération aléatoire réaliste de requêtes sur ce langage. En effet, nous n'avons pour l'instant utilisé ce langage qu'en démonstration déployée sur un faible nombre de machines : dans cette configuration, les données sont réelles, mais les requêtes sont saisies manuellement et donc peu nombreuses.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Annexe 3

Démonstrateur

Cette annexe fait suite au quatrième chapitre de ce document, dans le but de détailler l'organisation interne du démonstrateur utilisé. Le code de ce démonstrateur, diffusé sous les termes de la licence GNU GPL v3, n'est pas reproduit ici, étant donné le peu d'intérêt à fournir ce code sous format papier, mais peut être récupéré par l'intermédiaire d'un dépôt git dédié, publiquement accessible aux adresses suivantes :

`http://liris.cnrs.fr/~sdufrome/fleddidem/fleddi`
`git://fadrienn.irlnc.org/fleddi`

Nous détaillons ici le contenu de ce dépôt en nous intéressant successivement à chacun de ces quatre *packages* principaux : `lang`, contenant les informations relatives aux langages, `orga`, contenant les implémentations des différents systèmes considérés, `netw`, contenant les couches de communications entre les nœuds, et `inst`, contenant les différents outils d'instrumentation servant à en étudier le fonctionnement. Une dernière section est ensuite consacrée à la mise en place d'un fonctionnement entièrement distribué issu du code de ce démonstrateur.

3.1 Analyse et traitement des requêtes

Le *package* `lang` contient donc tout le code servant à analyser les requêtes et procéder aux traitements, ce qui comprend notamment la fonction de coût et le système de réécriture. Le principe de conception de ce *package* est de fonctionner de manière aussi générique que possible, fournissant une base à ce qui sera utilisé pour le reste du système, sans jamais dépendre du code du reste du simulateur (exception faite des classes purement utilitaires conventionnellement placées dans le package `inst.util`).

Les deux langages de requêtes implémentés à ce jour étant décrits dans l'annexe précédente, cette section s'attache plus spécifiquement aux aspects extérieurs, d'abord par la description des interfaces, présentes dans (`lang.base`), et en second lieu en précisant la façon dont les implémentations fournies peuvent être paramétrées.

3.1.1 Interfaces

Le *package* `lang.base` est destiné à contenir les seuls codes qui seront directement manipulés par le reste du simulateur, dans le but de fournir des implémentations des différents systèmes qui soient totalement indépendantes du langage utilisé. Cela se décompose en trois interfaces principales : `lang.base.Tuple` décrit un élément de flux, `lang.base.Query` décrit une requête (ou toute autre transformation au sens indiqué dans

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

la définition 2, page 30), et `lang.base.QueryModel` fournit les méthodes qui seront utilisées par le *tracker* pour procéder à l'organisation.

En complément de ces interfaces, une interface `LangLoader` est proposée. Afin de permettre de charger dynamiquement le langage sans avoir à modifier les classes des autres *packages*, chaque nouveau langage implémenté doit fournir une classe `lang.X.LangLoader` implémentant celle-ci, où `X` sera l'identifiant du langage pouvant être passé en argument à la ligne de commande. Cette classe, dont le constructeur principal ne doit pas prendre de paramètres, sera chargée de déterminer les propriétés du langage en fonction des autres arguments spécifiés.

Interface 1 (*lang.base.Tuple*)

- ***public String*** *getId();*
- ***public String*** *getSource();*
- ***public long*** *getTimestamp();*
- ***public List<String>*** *schema();*
- ***public boolean*** *hasKey(**String** key);*
- ***public Object*** *getValue(**String** key);*
- ***public Tuple*** *convert(**ListMap<String, String>** names)*

Cette interface reprend les éléments minimaux de la définition 1, à savoir la possibilité de récupérer l'identifiant du tuple, la source l'ayant produit et le *timestamp* indiquant le moment de sa production. Considérant qu'un élément de flux contiendra vraisemblablement d'autres données sur lesquelles pourront s'exprimer les requêtes, nous avons ajouté des méthodes permettant d'obtenir son schéma, de vérifier si le tuple contient une valeur pour une clef donnée, et de récupérer cette valeur. Enfin, une méthode de conversion permet d'obtenir un tuple contenant tout ou partie des colonnes considérées, en les renommant si besoin à la manière de ce que l'on peut obtenir lors d'une requête « SELECT X AS Y ». Des définitions plus précises peuvent être placées dans le *package* `lang.base.items`, qui contient une interface décrivant un élément constitué essentiellement de texte, ainsi que les implémentations génériques correspondantes.

Interface 2 (*lang.base.Query*)

- ***public String*** *id();*
- ***public String*** *expression();*
- ***public List<String>*** *providers();*
- ***public boolean*** *needsWork();*
- ***public boolean*** *isSource();*
- ***public Tuple*** *sample();*
- ***public List<Item>*** *perform(**Tuple** tuple, **TupleAnalysisLogger**... logs);*

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Cette interface, pour sa part, contient les informations minimales dont disposer concernant une requête. La question s'est posée, au cours du développement, d'intégrer ou non dans cette interface un ensemble d'opérations plus détaillées, telle que notamment une vérification d'équivalence. Le choix s'est fait de conserver ces fonctionnalités dans une classe dédiée, afin de minimiser autant que possible la complexité du code à échanger entre les différents participants.

À chaque requête est donc associé un identifiant unique, partagé par la requête originale et toutes les réécritures qui peuvent en être constituées. Cet identifiant étant apposé par le système, l'expression d'origine, telle que formulée par le participant, reste indiquée, afin de s'assurer que celui-ci puisse la reconnaître. La méthode *providers()* fournit pour sa part les identifiants des requêtes dont les flux de résultats seront utilisés, afin de diriger correctement les tuples reçus. Les méthodes *isSource()* et *needsWork()* sont des indicateurs rapides permettant d'identifier le type de requêtes utilisé : cette seconde méthode n'évalue pas le coût du travail effectué, mais indique simplement s'il s'agit ou non d'une requête identité. La méthode *sample()* est destinée à fournir un tuple dont le schéma correspond à celui des résultats qui seront obtenus, afin de préparer une éventuelle persistance dans une base de données.

Enfin, la méthode *perform(tuple)* est destinée à être appelée lorsqu'un participant reçoit une donnée particulière ; et renvoie alors les résultats générés par la requête après examen du tuple considéré. Il est optionnellement possible de lui fournir des objets chargés de générer des *logs* correspondant aux analyses réalisées (l'interface appropriée est jointe dans le même *package*). Notez que cette méthode est également utilisée pour les requêtes sources : on lui fournit alors un objet `TextTuple` contenant la description textuelle du ou des élément(s) demandé(s), et elle renvoie le ou les objets `Tuple` correspondant(s) à diffuser dans le système.

Interface 3 (`lang.base.QueryModel`)

- ***public Query*** *parse(**String** expr, **boolean** source);*
- ***public List<Query>*** *sources();*
- ***public void*** *dispose(**Query** query);*
- ***public double*** *cost(**Query** query);*
- ***public boolean*** *equivalents(**Query** q1, **Query** q2);*
- ***public boolean*** *mayMissEquivalences();*
- ***public Query*** *copy(**Query** query);*
- ***public Query*** *identity(**Query** query, **Query** parent);*
- ***public Query*** *derive(**Query** base, **Query** task, **List<Query>** parents);*
- ***public <T> T*** *model(**Class<T>** type);*

La troisième interface contient donc les méthodes de base qui peuvent être utilisées par le système, à commencer par la méthode permettant de générer un objet requête à partir d'une expression textuelle. Les sources de données utilisées devant être connues (afin, notamment, de s'assurer que les requêtes exprimées sur ces sources sont bien conformes

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

aux schémas correspondants), celles-ci doivent également pouvoir être consultées à tous moments. Par ailleurs, dans le cas où la génération d'un objet requête mobiliserait des ressources qui pourraient être nettoyées par la suite, une méthode est destinée à être appelée par le *tracker* en cas de retrait.

La méthode suivante correspond à la fonction de coût utilisée, qui évalue donc la charge de traitement correspondant à un objet requête donné, permettant par exemple de choisir entre différentes réécritures. Vient ensuite la possibilité de vérifier l'équivalence logique entre deux requêtes, propriété considérée comme minimale pour mettre en place un système de calcul. La détection d'équivalence n'étant pas nécessairement complète, la méthode *mayMissEquivalences()* permet au *tracker* de savoir s'il doit ou non prévoir le cas d'une détection tardive.

Les trois méthodes suivantes sont des méthodes de base de conversion qui ne dépendent pas de propriétés avancées du langage : la réalisation d'une copie stricte d'une requête donnée, logiquement équivalente, mais pourvue d'un identifiant différent, la production d'une requête identité qui récupérera les résultats d'une requête équivalente sans réaliser le moindre calcul, et la possibilité d'appliquer les traitements d'une requête donnée à d'autres entrées, considérées équivalentes. Cette dernière possibilité correspond, par exemple, au fait d'appliquer une requête sur un ensemble de miroirs plutôt que sur les sources primaires, ou au fait d'instancier l'acquisition de données pour une communauté en les récupérant auprès d'une unité particulière de cette communauté.

Les propriétés plus avancées telles que les réécritures n'étant pas nécessairement disponibles dans tous les langages, elles ne sont pas présentes dans l'interface de base, mais la dernière méthode permet d'obtenir, si disponible, un objet qui présentera les bonnes propriétés. Les interfaces correspondantes, plus réduites, sont situées dans le *package* `lang.base.models`. À titre d'exemple, obtenir le système de réécritures se fera par un appel à `model(lang.base.models.Rewriting.class)`, ce qui renverra un objet implémentant l'interface `Rewriting`, ou `null` si le langage ne permet pas cette possibilité.

3.1.2 Paramétrage des implémentations

Comme indiqué précédemment, chaque implémentation de la classe `LangLoader` permet de lire les arguments passés à la ligne de commande (ou lus depuis un fichier de configuration). Nos deux langages actuels, respectivement présents dans les *packages* `lang.filtering` et `lang.pseudosql`, reconnaissent en commun les paramètres suivants :

- `--base-volumetry=X` détermine la volumétrie de base à utiliser pour estimer les coûts (nombre réel, correspondant à l'espérance de volumétrie des sources par unité de temps). Ce paramètre n'a pas pour vocation à être forcément représentatif de la réalité (il n'est utilisé que pour une estimation, et commun à toutes les requêtes évaluées), mais doit correspondre à une valeur suffisamment élevée pour que les comparaisons aient un sens. La valeur par défaut est 200,0.

- `--rewriting-limit=X` fixe une limite maximale (entière) au nombre de réécritures calculées à chaque appel à la fonction de réécritures. Si le nombre indiqué est 0 ou moins, la fonction de réécritures est désactivée, et ne peut donc pas être utilisée par le *tracker*. La valeur par défaut est 30.

- `--max-delay=X` fixe une limite maximale à la durée d'exécution de la fonction de réécriture, pour les cas où la limite par nombre de réécritures trouvées ne semble pas la plus pertinente.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

- `--nfdisabled` désactive la mise en forme normale des requêtes lors de leur examen. Cela permet un léger gain dans le temps d'analyse des requêtes, tout en faisant apparaître un certain nombre de faux négatifs dans les vérifications d'équivalences (celle-ci se basant sur la comparaison des formes normales).
- `--no-containment` désactive le modèle de *containment*, et empêche donc le *tracker* d'utiliser les fonctions associées, ce qui peut être utile pour effectuer des tests.

Par ailleurs, le langage basé sur les filtres sur mots-clefs accepte ces deux paramètres supplémentaires :

- `--keywords=X` indique la liste de mots-clefs qui seront utilisés (liste de chaînes de caractères, séparées par des virgules). Cela permet d'estimer la sélectivité des filtres par mots-clefs, d'après une loi de Zipf (les mots-clefs étant fournis du plus fréquent au moins fréquent). Par défaut, ce paramètre correspond aux valeurs utilisées par les outils de générations de fichiers d'expérimentation.
- `--equiprobability` indique au système de considérer que les mots-clefs sont tous équiprobables, et ne correspondent donc pas à une loi de Zipf. La sélectivité de chaque filtre par mot-clef correspond alors à l'inverse du nombre de mots-clefs.

3.2 *Tracker* et organisation du système

Le *package* `orga` contient les différentes implémentations de fonctions d'organisation. Cette section en présente les mécanismes généraux communs à l'ensemble des organisations fournies, avant de détailler les spécificités de nos implémentations des approches Unicast, Multicast, Semantic Peer-to-Peer Overlays for Publish/Subscribe Networks[16] et, bien sûr, QTor (présente en deux versions).

Malheureusement, l'implémentation de Delta[53] dont nous disposons, afin d'être la plus proche possible de la proposition originelle, repose sur l'utilisation du solveur linéaire Gurobi[74], qui n'est lui-même pas diffusé sous licence libre. De ce fait, cette implémentation ne peut pas être présente dans un dépôt public.

3.2.1 Mécanismes généraux

Le contenu de ce *package* a donc pour objectif de coder le fonctionnement interne au *tracker*. Pour cela, reposant sur notre modélisation générique présentée au second chapitre, section 2.2, nous fournissons des classes `Participant` et `Unit`. L'identification entre les participants réels et les objets de type `Participant` est du ressort de l'implémentation réseau, et donc extérieure à ce *package*, de même que le mécanisme permettant de transmettre les informations aux participants.

La classe *Unit* est dédiée à gérer les unités de calcul, au sens de la définition 6 (page 32). Il est donc possible de préciser une requête à laquelle est associée l'unité – ne correspondant pas nécessairement aux requêtes exprimées par le participant. Celle-ci peut toutefois être laissée à `null` pour certains usages arbitraires. Une unité peut être considérée comme *principale* ou non (un booléen `main` fourni à sa création), signalant si les transformations affectées seront, ou non, équivalentes à cette requête (ce qui est le cas pour une réécriture traitée par un unique participant, mais pas lorsque le traitement d'une même requête – réécrite ou non — est divisé entre plusieurs participants). Les approches actuellement implémentées n'utilisent que des unités principales.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Une interface `Tracker` est fournie dans ce *package* ; toutefois, il est conseillé de ne pas l'implémenter directement, mais plutôt d'implémenter la classe `AbstractTracker`, qui automatise tout ce qui peut l'être, et fournit notamment une méthode *owner(id)* fonctionnelle, qui renvoie, à partir de l'identifiant d'une requête, le nœud logique (unité ou communauté) en charge de la traiter. Dériver de cette classe ne laisse plus que quatre méthodes à implémenter :

Interface 4 (*`orga.base.Tracker`*)

- ***protected boolean*** *insert(**Participant** participant, **Query** query);*
- ***protected boolean*** *delete(**Participant** participant, **Query** query);*
- ***protected boolean*** *recalibrate(**Participant** p);*
- ***public void*** *optimize();*

Les deux premières sont respectivement chargées d'insérer la requête du participant dans le système et de l'en retirer (le participant pouvant toutefois rester s'il a plusieurs requêtes), c'est-à-dire de créer ou disposer l'objet `Unit` correspondant, ainsi que les éventuelles autres unités, et de réaliser les branchements ; tandis que la troisième sert à effectuer les modifications relatives à une annonce de changement de capacité. Ces trois méthodes renvoient un booléen indiquant si elles ont effectué des modifications ou non : il est possible – quoique déconseillé – de laisser les changements en attente. La dernière méthode sert à déclencher une optimisation globale, qui peut donc ne rien effectuer si l'organisation ne présente pas de tel mécanisme. Si des changements peuvent être laissées en attente, cette méthode doit, au minimum, forcer le *tracker* à s'assurer que l'ensemble du système est dans un état cohérent avant de procéder à la phase suivante des expérimentations.

Comme pour le langage, une classe abstraite `TrackerLoader` permet de charger automatiquement un *tracker* sans avoir à modifier le reste du code du projet. Il est pour cela nécessaire d'en fournir une implémentation de la forme `orga.X.TrackerLoader`, où `X` représente l'identifiant du mode d'organisation qui sera précisé par la ligne de commande.

3.2.2 Exemple simple : Unicast

Le premier exemple de *tracker* fourni, dans le *package* `orga.unicast`, correspond au cas le plus simple où chaque participant est directement relié aux sources de données concernées. Cette approche est cependant paramétrable, dans la mesure où il est possible de réaliser soit un unicast classique (la source envoie son flux brut à chacun des participants, qui applique sa requête en local), soit un système de calcul centralisé (la source dédie une unité à chacune des classes d'équivalence de requêtes qu'elle reçoit, effectue les calculs en local et envoie directement à chacun ses résultats détaillés). C'est le premier comportement qui est activé par défaut, le second pouvant l'être par l'usage de l'option `--compute` lors du chargement par la ligne de commande.

Ce *tracker*, comme tous ceux présentés ici (Delta étant la seule organisation actuellement implémentée ayant besoin de recourir à un mécanisme d'optimisation), applique les changements immédiatement. En revanche, il ne tient aucun compte des capacités des participants ni de leurs modifications : le résultat probable d'une telle organisation est de grandement surcharger la source de données.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

3.2.3 Multicast et arbres de diffusion

Le second exemple, fourni dans le *package* `orga.multicast`, correspond à la mise en place d'un arbre de diffusion commun, dans lequel tous les participants du système s'échangent les flux sources bruts, avant de procéder au traitement de leurs requêtes en local. Chaque participant se voit donc doté d'au moins deux unités, un miroir et une unité de traitement.

Le mécanisme mettant en place l'arbre de diffusion est constitué de deux classes, `DiffusionTree` et `TreeUnit`. N'étant pas nécessairement spécifiques aux organisations de type Multicast applicatif, ces classes sont situées dans le *package* `orga.base.communities`. La classe `DiffusionTree` est chargée d'appliquer, sur des unités spécialisées, l'algorithme d'organisation interne en fonction des capacités présenté au chapitre trois (algorithme 3, page 83).

Chaque participant intégrant le système se voit inséré dans autant d'arbres de diffusion qu'il n'y a de sources, puis reçoit une requête dérivée lui permettant de procéder aux traitements qui l'intéressent à partir des différents flux bruts qu'il doit manipuler. Contrairement au précédent, ce *tracker* gère les modifications de capacités d'un participant, en déclenchant une réorganisation des différents arbres de diffusion concernée (l'algorithme étant conçu pour obtenir la meilleure latence en ne déplaçant que le moins de nœuds possibles).

Notre implémentation du Multicast applicatif accepte un paramètre depuis la ligne de commandes, `--no-sources`, qui indique de ne pas s'appuyer sur les capacités de diffusion des sources, si puissantes soient-elles. Cela permet, comme dans le cas du Multicast réseau, de limiter les envois effectués par la source à un seul exemplaire de chaque item produit. Sans ce paramètre, la source est pleinement intégrée à l'arbre de diffusion et atteint (sans jamais le dépasser) son maximum de connexions dès lors que le nombre d'utilisateurs est suffisamment grand.

3.2.4 SemPO et mise en cascade

L'implémentation suivante, fournie dans le *package* `orga.sempo` est celle de Semantic Peer-to-Peer Overlays for Publish/Subscribe Networks. Il s'agit ici de code produit par nos soins, suivant les indications fournies dans l'article d'origine[16], mais sans disposer de la production originale.

Le principe est ici de mettre en relation les nœuds en fonction des relations d'inclusion de requêtes. Le code permettant de rechercher la position d'un nœud donné, fortement inspiré de celui présenté au chapitre trois (algorithme 1), pouvant lui aussi être réutilisé dans d'autres circonstances, il est fourni dans la classe `ContainmentCascade` du *package* `orga.base.cascades`. Cette classe contient également une méthode permettant d'identifier les nœuds à déplacer en cas de réorganisation après insertion.

Comme dans la proposition d'origine, cette réorganisation est optionnelle : désactivée par défaut, elle peut être activée par l'option `--reorder`. Les autres options reconnues sont `--rewrite`, permettant d'utiliser une véritable réécriture (quoique toujours limitée à une seule vue au maximum) plutôt qu'un simple changement de flux entrant, et donc de s'abstenir de réappliquer des traitements inutiles, et `--limit-fanout`, qui permet de prendre en compte les limitations de capacités des nœuds, par une extension proposée par nous et basée sur l'algorithme d'organisation de l'arbre de diffusion. Sans cette option, le *tracker*, fidèle à la proposition originelle, place toutes les requêtes s'avérant équivalentes à une requête déjà insérée comme des successeurs directs de celle-ci, conduisant à surcharger certains nœuds.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

En cas de disparition d'un participant devant envoyer ses résultats à d'autres, hors cas de réorganisation pour des raisons de capacités, le *tracker* sélectionne le premier nœud équivalent à celui qui quitte le système pour le mettre à la place du précédent, selon le concept de « *leaf promotion* » proposé par les auteurs.

3.2.5 QTor, FQTor et communautés

Enfin, notre propre proposition repose des classes implémentant le concept de communautés. La classe abstraite `Community`, du *package* `orga.base.communities`, en fournit le fonctionnement de base : chaque communauté embarque un arbre de diffusion interne, permettant à l'ensemble de ses participants d'en recevoir les résultats comme dans un système à miroirs. Chaque classe héritant de *Community* n'a alors plus qu'à déterminer comment se produisent l'acquisition et le calcul.

L'implémentation actuellement fournie, `SimpleCommunity` correspond au cas, décrit au chapitre trois, dans lequel un participant donné est en charge de calculer les données, étant présent dans la communauté enfant et dans chacune des communautés parentes. Cette classe embarque le code de sélection permettant de sélectionner cette tête de communauté parmi les membres de toutes ces communautés (algorithme 2) ; qui n'est cependant jamais exécuté lorsque la communauté est constituée autour d'une source primaire de données. D'autres implémentations de communautés sont envisagées, notamment pour permettre la distribution des calculs[54].

Le *tracker* QTor, présent dans le *package* `orga.qtor`, repose donc sur une composition des différents éléments présentés : un objet de type cascade permet d'appliquer l'algorithme d'exploration du système, puis un objet de type communauté est créé pour chaque classe d'équivalence de requêtes identifiée. Dans une optique d'adaptabilité, le choix du type de cascade dépend des propriétés du langage : si un système de réécritures est fourni, la cascade utilisera les relations qui en découlent ; sinon, elle se basera si possible sur des relations d'inclusion de requêtes, comme dans le cas de SemPO. Si aucune propriété avancée n'est disponible, elle utilisera une « `NonCascade` », connectant chaque communauté aux communautés sources concernées.

Deux options sont ici disponibles : comme pour SemPO, `--reorder` permet d'activer les réorganisations après création d'une nouvelle communauté. Par ailleurs, `--checkup` permet de vérifier, lors de l'insertion d'un nouveau participant, si le système entier peut gagner à ce que ce participant envoie des ressources dans les communautés enfants de celle qu'il rejoint initialement (en l'absence de cette option, l'envoi de ressources d'un parent vers un enfant ne peut se faire que lors du rebranchement de la communauté enfant ; mais le nœud nouvellement inséré peut en revanche, selon ses capacités, envoyer des ressources vers ses propres communautés parentes).

Pour des raisons pratiques, un *tracker* QTor simplifié est proposé dans le *package* `orga.fqtor`. Celui-ci, qui ne reconnaît aucun paramètre particulier, utilise systématiquement une « `NonCascade` », quelles que soient les propriétés du langage. Cela permet de lancer simultanément au sein d'une même simulation ces deux comportements pour les comparer, attendu que les propriétés du langage sont communes à tous les *trackers* utilisés simultanément.

3.3 Échanges réseau réels ou simulés

Tout le code présenté dans les deux sections précédentes était conçu pour tourner au niveau du *tracker*. Afin d'obtenir un système fonctionnel, il est nécessaire d'y adjoindre

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

un fonctionnement prenant en compte les échanges réseau, réels ou simulés. C'est l'objet du *package* `netw`, qui, comme les deux autres, présente une partie de code commun, puis différentes implémentations. Quatre ont été terminées à ce jour, la dernière, correspondant à un véritable prototype réseau, étant encore en cours de finalisation.

3.3.1 Mécanismes généraux

La classe abstraite `SysNode` fournit le code dont tout nœud lambda du système doit pouvoir disposer, à savoir la possibilité d'émettre et de retirer une requête, de recevoir des instructions du *tracker* et d'envoyer des items aux autres nœuds, la manière réelle de communiquer devant être définie dans les classes en dérivant, selon l'interface suivante :

Interface 5 (*`netw.base.SysNode<S>`*)

- ***public boolean*** *wants(**Query** query);*
- ***public boolean*** *isme(**S** child);*
- ***protected void*** *send();*
- ***protected void*** *send(**String** id);*
- ***protected void*** *send(**String** expr, **boolean** source);*
- ***protected void*** *send(**Tuple** tuple, **String** provider, **List<S>** children);*

La première méthode, dépendant de la façon de communiquer avec l'utilisateur·trice, indique si une requête donnée, reçue du *tracker*, correspond à une demande ou non, et donc s'il faut mémoriser les résultats en plus de les envoyer ailleurs dans le système. Celle qui suit permet d'identifier si l'objet passé en paramètre correspond, ou non, au même participant physique, pour éviter les échanges externes dans ce cas. Les trois méthodes suivantes sont les envois d'informations vers le *tracker* pour signaler, respectivement, un changement de capacités, le retrait d'une requête dont l'identificateur est connu, ou l'expression d'une nouvelle requête par sa description textuelle. La dernière méthode, enfin, permet d'envoyer un tuple à une liste déterminée d'enfants, en précisant pour cela l'identifiant de la requête ayant produit le tuple en question (ce qui évite de recréer inutilement des objets `Tuple` lorsque les requêtes ne font qu'analyser ces derniers sans les modifier).

En supplément de cette méthode, il est nécessaire d'implémenter également l'interface `Remote`, utilisée par les unités pour transmettre les instructions du *tracker* et de ce fait présente dans le *package* `orga.base` :

Interface 6 (*`orga.base.Remote`*)

- ***public boolean*** *accept(**Query** task);*
- ***public void*** *addTask(**Query** task, **List<Participant>** children);*
- ***public void*** *addChild(**Query** task, **Participant** child);*
- ***public void*** *remChild(**Query** task, **Participant** child);*
- ***public void*** *remTask(**Query** task);*

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

La première de ces méthodes permet de savoir si un participant accepte de travailler pour une requête donnée qu'il n'a pas exprimée (dans le cadre d'une simulation, il est possible de supposer, sans échange réseau, que le résultat sera toujours le même). Les quatre autres servent à ajouter ou retirer une tâche donnée, et à indiquer d'autres participants à qui envoyer les résultats d'une requête.

Pour toutes les implémentations locales du système, destinées à être utilisées au sein d'une même instance de l'interpréteur Java, une classe abstraite `SimNode`, dérivée de `SysNone`, est capable de nommer directement les objets correspondant à son *tracker* et à ses descendants, ce qui permet d'automatiser, notamment, l'affichage graphique de l'état du réseau simulé. Une classe `NodeControl` est destinée à prendre en compte plusieurs instances de `SimNode` correspondant au même participant simulé, et de leur transmettre simultanément les mêmes instructions extérieures (exprimer ou retirer une requête, changer de capacité, publier un tuple inédit), ce qui permet de simuler un fonctionnement coordonné sur plusieurs *trackers* pour étudier leurs différentes de fonctionnement.

L'interface `NetworkControl` est alors chargée de faire le lien entre ces objets et les instructions automatisées :

Interface 7 (*`netw.base.NetworkControl`*)

- ***public void*** *launch(**Runnable** init, **int** actions);*
- ***public Object*** *calibrate(**String** node, **int** capacity);*
- ***public void*** *recalibrate(**Object** node, **int** capacity);*
- ***public void*** *subscribe(**Object** node, **String** expr, **boolean** source);*
- ***public void*** *publish(**Object** node, **String** source, **String** item);*
- ***public void*** *withdraw(**Object** node, **String** expr);*
- ***public void*** *control(**boolean** optimize, **boolean** print, **boolean** pause);*
- ***public boolean*** *acceptEvents();*

La première méthode sert à lancer l'instance automatisée qui génèrera les événements décrits ensuite (ce qui peut se faire directement ou dans un *thread* séparé). Le nombre d'événements qui seront générés est précisé à titre indicatif. La méthode suivante correspond à la création d'un nouvel objet *NodeControl*, ou équivalent selon l'implémentation considérée, à l'aide d'un identificateur (qui peut être `null` si le nœud n'est pas identifié) et de sa capacité initiale, et qui renvoie un objet dont le type n'est pas spécifié, qui servira par la suite à identifier ce nœud en particulier. Les quatre méthodes suivantes permettent de communiquer des instructions à ce nœud.

La méthode *control* permet de contrôler l'application elle-même : passer *optimize* à vrai déclenche l'application de la méthode *optimize()* sur tous les *trackers* concernés ; passer *print* à vrai demande aux générateurs de statistiques d'indiquer les résultats actuellement obtenus ; et passer *pause* à vrai suspend l'exécution, si toutefois l'application est destinée à pouvoir être contrôlée par l'interface graphique. La dernière méthode indique d'ailleurs si des événements issus d'une autre source que le `Runnable` précédent seront acceptés ou non (ce qui peut changer dynamiquement : les classes implémentant `NetworkControl` doivent également hériter de `Observable`).

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Comme pour les deux autres parties du simulateur, une classe `netw.X.NetLoader` implémentant la classe abstraite éponyme du *package* `netw.base` permet de charger dynamiquement l'implémentation réseau `X` et de lire ses paramètres depuis la ligne de commande (encore que les différentes implémentations proposées à ce jour n'aient aucun paramètre particulier à prendre en compte).

3.3.2 Communication directe entre objets Java

La première implémentation de « réseau » proposée, `direct`, correspond à une situation simple où la communication entre les différents nœuds se fait par l'appel direct des méthodes considérées, les objets cibles étant connus. Cette implémentation simple permet de réaliser des tests rapides, le comportement de la « couche réseau » étant entièrement déterministe.

Un inconvénient, toutefois, est que certaines opérations se passent dans un ordre peu réaliste : lors de l'envoi d'un item d'un nœud vers un autre, la méthode de réception du destinataire appelle ensuite automatiquement la méthode d'envoi vers les nœuds suivants, ce qui fait que la propagation des informations dans le graphe se fait en profondeur plutôt qu'en largeur.

Une seconde implémentation, `steps`, permet d'arranger ce souci : il s'agit d'une version dérivée de `direct`, dans laquelle seul l'envoi des données entre les participants a été modifié pour fonctionner par étapes successives de propagation. La limitation de capacité est prise en compte : un nœud surchargé est contraint d'envoyer ses données en plusieurs étapes, ce qui permet de visualiser la latence induite par la surcharge.

Cette implémentation n'est pas adaptée au lancement d'expérimentations rapides destinées à générer des statistiques ; mais elle est en revanche conçue pour faciliter la communication avec l'interface graphique, notamment en pouvant recevoir des événements arbitraires générés par l'utilisateur·trice de cette interface, en plus de ceux automatisés.

3.3.3 Simulateur basé sur PeerSim

Une troisième implémentation, basée sur le simulateur PeerSim[66], est en revanche dédiée au fait de lancer des expérimentations rapides. Elle n'est donc pas destinée à un affichage graphique, et n'en reçoit aucun événement (le mode de fonctionnement de PeerSim rendrait un contrôle tiers de ce type complexe à mettre en place).

L'implémentation choisie repose sur le fonctionnement *EventDriven* du simulateur, dans lequel chaque événement attendu se déroule dans une même unité de temps (tous les transferts d'informations entre nœuds, communications avec le *tracker* comprises, se fait avec un délai de 0). La phase d'initialisation se charge d'envoyer des messages de contrôle avec les délais requis pour que les différents événements considérés soient successifs.

Pour simplifier le fonctionnement, toutes les communications se font par le même protocole ; mais différents types de messages sont implémentés pour représenter les informations échangées (souscription, retrait d'une requête ou changement de capacité, envoyés d'un nœud vers le *tracker* ; ajout ou retrait d'une transformation, ou notification de nœud à servir, envoyés du *tracker* vers un nœud, et émission d'un item, envoyé d'un nœud vers un autre). L'ordre d'arrivée de messages envoyés au même moment n'étant pas déterministe, le *tracker*, s'il a besoin d'envoyer plusieurs instructions à un même nœud au cours d'un unique événement, attend un accusé-réception (*Acknowledgement*), afin de s'assurer qu'il n'y aura pas de conflits dus à l'ordre d'arrivée des instructions.

Notons que, quoiqu'ils connaissent l'objet Java destinataire, tous les nœuds (du point de vue de FleDDi) sont associés sur des nœuds (du point de vue de PeerSim) distincts,

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

et toutes les communications se font par la simulation du réseau. Virtuellement, donc, en dehors des méta-informations, les messages sont propagés en ne contenant que les informations utiles (description textuelle d'une requête, transformation à réaliser, item émis), comme dans une situation réelle.

3.3.4 Prototype pour déploiement réel

La dernière implémentation en date repose d'ailleurs sur un fonctionnement très nettement similaire à celui basé sur PeerSim, quoique ce fonctionnement soit ici fait pour être réellement distribué sur plusieurs machines. Cette approche est fournie dans le *package* `netw.deployed`, dans lequel le code destiné au *tracker* et celui destiné aux participants sont séparés en deux sous-*packages* distincts, exception faite d'une classe contenant le code commun d'envoi de messages réseau.

Ne relevant plus du domaine de la simulation, cette implémentation repose sur un mode de fonctionnement différent de celui des trois précédentes. Ainsi, pour les participants, la classe mère `SysNode` est dérivée directement, sans passer par la classe intermédiaire `SimNode`, et identifie les participants auxquels envoyer les données par la combinaison de leur adresse IP et du port de communication signalé, plutôt que par l'objet java gérant cet autre participant (qui n'est plus connu localement).

Les messages sont échangés sous forme d'objets Java sérialisés. Une précédente implémentation faisait échanger des messages textuels (requêtes par leur expression, et flux par des fichiers au format RSS), mais cette version, quoique présentant l'avantage d'avoir des participants plus autonomes, présentait quelques limites en ce qui concerne la généricité, et notamment empêchait l'implémentation future de *trackers* confiant des transformations arbitraires, ne pouvant être détaillée dans le langage de requêtes.

N'étant plus une simulation, cette implémentation n'est pas conçue pour être contrôlée par un fichier d'instruction comme les précédentes. De ce fait, si aucune interface de contrôle du *tracker* n'est prévue, en dehors de la possibilité de visualiser le graphe des participants (ou des unités) tel qu'il le connaît, les participants doivent pour leur part recevoir des instructions extérieures. Démarrés sans interface graphique particulière, ils attendent donc que ces instructions leur soient fournies par l'entrée standard. Un service web permettant de leur fournir des instructions de manière plus conviviale, de même qu'un service gérant la persistance vers une base de données, sont néanmoins fournis dans le *package* `inst.proto`.

3.4 Tests, classes principales et instrumentation

Les parties non-encore décrites du simulateur, et principalement le *package* `inst`, correspondent au code de contrôle de l'application (générateurs d'événements, lecture d'événements, gestion des logs et interface graphique). Contrairement aux précédentes, elles n'ont donc pas pour vocation à être étendues de manière générique (même s'il est évidemment possible d'ajouter ou de modifier des éléments).

3.4.1 Classe centrale du simulateur

La classe `Simulation` du *package* `netw.base` est une classe centrale dans nos expérimentations : c'est en effet elle qui gère la lecture des différents paramètres et le chargement automatique des classes concernées. Les paramètres reconnus sont les suivants :

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

- `--lang=X` permet d'activer le langage `X`. Le langage à base de filtre étant celui sur lequel reposent nos simulations, il est utilisé par défaut.
- `--network=X` permet de sélectionner l'implémentation réseau `X`. Par défaut, c'est l'implémentation basée sur PeerSim qui est sélectionnée.
- `--input=X` permet de sélectionner un fichier d'entrée, à partir duquel les différents événements seront lus. Cet argument est vraisemblablement nécessaire, à moins que la classe principale utilisée ne soit codée pour gérer elle-même une série d'événements.
- `--output=X` permet de sélectionner un fichier de sortie, dans lequel seront enregistrés tous les *logs* indiquant l'activité du système. Cet affichage est désactivé par défaut ; indiquer `/dev/stdout` permet de diriger cet affichage vers la sortie standard (conformément au mode de fonctionnement des systèmes de type UNIX). L'argument `--verbose` peut également être utilisé pour activer l'affichage vers la sortie standard.
- `--graphical` permet d'afficher l'interface graphique, masquée par défaut. Cette option peut être précisée (`--graphical=X`) pour demander un nombre `X` de vues simultanées ; toutefois, afficher plus de deux vues en même temps (ce qui est la valeur par défaut) est déconseillé.
- `--tracker=X` permet d'ajouter l'implémentation `X` à la liste des *trackers* chargés simultanément. L'expérimentation sera lancée de manière indépendante, mais simultanée, sur chacun des *trackers* chargés. Si aucune option de ce type n'est précisée, un *tracker* de type QTor est chargé seul.
- `--stats=X` permet d'indiquer le fichier dans lequel doivent être enregistrées les statistiques obtenues pendant l'exécution du dernier *tracker* indiqué.

Tous les autres arguments lus sont passés au langage, au réseau, puis au dernier *tracker* chargé. S'ils ne correspondent à aucun d'entre eux, un message d'avertissement est affiché sur la sortie d'erreur standard.

3.4.2 Points d'entrée pour l'interpréteur

Les deux classes fournies dans le *package* par défaut, `FleDDi` et `FleDDiDem`, qui proposent chacun une méthode *main(args)* servant de point d'entrée à l'interpréteur Java, se contentent donc de transmettre leurs arguments à cette classe `Simulation`, et de démarrer ensuite l'application à l'aide des instanciations réalisées. La différence entre ces deux classes est que `FleDDi` attend ses paramètres depuis la ligne de commandes, présentant le fonctionnement totalement générique du simulateur ; tandis que `FleDDiDem` génère elle-même les paramètres requis pour lancer un démonstrateur.

L'appel à cette seconde classe lancera donc, sans qu'il n'y ait besoin de préciser de paramètres particuliers, une application utilisant un langage de filtres booléens, reposant sur l'implémentation « réseau » `steps` avec une interface graphique, dans laquelle tourneront simultanément les *trackers* Unicast, Multicast, SemPO et QTor. Les événements seront lus depuis le fichier *example.tsv* présent à la racine du dépôt (et pouvant être embarqué dans une archive Jar).

Des points d'entrée supplémentaires pour l'interpréteur sont fournis dans les *packages* `test.filtering`, contenant les tests relatifs aux propriétés du langage de filtres, et `test.simulation`, contenant les tests de fonctionnement. Les premiers portent uniquement sur ce langage, sans recourir au reste de la simulation. Parmi les seconds (qui

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

correspondent essentiellement à des automatisations de paramètres, assortis de générations automatiques de jeux de données appropriés au test effectué), la classe `ResultTest` mérite d'être plus spécifiquement commentée.

Son rôle est de vérifier la conformité des résultats obtenus par les nœuds (ce qui, étant entendu que les méthodes du langage ont été vérifiées séparément, dépend essentiellement de la correction de l'organisation effectuée par le *tracker*), par comparaison aux résultats obtenus dans un cas d'Unicast classique. Elle lance donc systématiquement un *tracker* Unicast en plus de ceux demandés par les paramètres de la ligne de commande, génère des requêtes aléatoires sur un nombre réduit de mots-clefs, et demande à la source d'émettre un item correspondant à chaque combinaison possible de mots-clefs. Une fois la simulation terminée, les résultats obtenus par chaque nœud (assumant que l'implémentation réseau utilisée passe par des objets `NodeControl`) sont analysés pour vérifier qu'ils se correspondent bien dans chacun des *trackers* considérés.

3.4.3 Interface graphique

L'interface graphique fournie par le démonstrateur est entièrement écrite en Java/Swing, reposant sur la bibliothèque GraphStream[36] pour le rendu des graphes. Elle est normalement configurée pour adopter l'apparence native du système d'exploitation sur lequel le démonstrateur tourne, si celle-ci est disponible.

Cette interface présente une visualisation de deux graphes simultanés par défaut, permettant soit de visualiser le réseau du point de vue du *tracker* et du point de vue des nœuds, soit de comparer deux modes d'organisation lancés simultanément. Les tailles des nœuds et la façon de les positionner peuvent être modifiées par les menus. Un système de fenêtres internes peut ensuite être utilisé pour obtenir des informations générales, ainsi que le détail des nœuds, permettant d'interagir avec eux (ajouter ou retirer une requête, par exemple).

Pour l'instant, les seuls placements proposés sont le placement automatique géré par GraphStream sans contraintes particulières, et un placement en arbre, plaçant les sources en bas de la fenêtre et les nœuds par étages successifs. D'autres placements peuvent être obtenus en créant de nouvelles classes dans le *package* `inst.display.placements`, qui sera exploré automatiquement. Les classes en question n'ont pas d'interface particulière à implémenter, car elles ne seront pas instanciées par l'interface : tout se fait par un ensemble de méthodes statiques.

Interface 8 (*`inst.display.placements`*)

- ***public static Point*** *place(**Node** node);*
- ***public static boolean*** *useAutoLayout();*
- ***public static String*** *name();*

La première méthode est chargée de déterminer la position d'un nœud donné (au sens de GraphStream ; possédant un attribut « nodeview » permettant de retrouver le détail du nœud FleDDi), soit en renvoyant un objet de type `Point` fournissant les coordonnées (l'axe `z` de GraphStream n'est pour l'instant pas utilisé), soit en jouant sur ses liens avec les autres nœuds du système pour poser des contraintes à l'algorithme de placement de GraphStream. Celui-ci sera en effet activé si la seconde méthode renvoie vrai, et désactivé

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

sinon. La dernière méthode, pour sa part, indique le nom qui sera utilisé pour identifier le mode de placement dans le menu correspondant.

3.4.4 Gestion des logs

Plusieurs interfaces de logs sont présentées à divers endroits du système, en fonction des informations qui peuvent y être loguées. L'interface `netw.base.SimLogger` regroupe toutes les méthodes concernées, qui varient par les paramètres fournis. Trois implémentations principales en sont proposées :

- `inst.display.GUILogger` est utilisé pour communiquer à l'interface graphique les changements effectués dans le système, afin qu'elle se mette à jour.
- `inst.logs.StatsLogger` génère des statistiques sur le *tracker* pour lequel il est actif, et permet donc de chiffrer les expérimentations.
- `inst.logs.TraceLogger` permet d'obtenir un affichage (très) verbeux listant les opérations effectuées dans le système.

C'est la classe `Simulation` qui se charge d'instancier ces différents systèmes de logs et de les fournir aux objets concernés. Il est possible de créer de nouvelles implémentations de logs ; mais il faudra alors sans doute modifier cette classe pour ajouter les paramètres relatifs à leur instanciation et gérer leur répartition, ce qui s'automatise plus difficilement que pour les autres parties du système.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Annexe 4

Documentation fonctionnelle

Bootstrap

Au lancement sur une machine donnée, le logiciel crée un participant et propose à l'utilisateur de lancer une requête sur un système QTor existant (qui peut être soit connu via un tracker s'il est centralisé, soit découvert/construit au fur et à mesure s'il est distribué). L'algorithme « anchor » est alors lancé.

QTor

Cette entité regroupe tous les aspects communs à l'ensemble du système, comme la gestion des réécritures. Tout ce qui n'est pas spécifique à un participant ou à une communauté est regroupé ici.

Traitement des requêtes

$cost : \mathscr{Q} \rightarrow \mathbb{R}$

Cette fonction renvoie le coût associé à la requête.

Le modèle de coût actuellement utilisé associe comme partie entière l'estimation de charge de calcul (elle-même dépendant du nombre d'opérateurs à effectuer sur les données et du nombre de données attendues en entrée), ce qui est affiné par une estimation de la latence (qui dépend du nombre de hops, la durée de transfert étant supposée uniforme) utilisée comme partie décimale afin de ne pas être prioritaire.

$providers : \mathscr{Q} \rightarrow 2^{\mathscr{C}}$

Cette fonction renvoie les communautés auprès desquelles les données doivent être acquises. S'il s'agit d'une requête de base telle qu'exprimée par l'utilisateur, cela permet d'identifier les communautés correspondant aux sources concernées. S'il s'agit d'une requête réécrite, cela permet d'identifier les communautés parentes avec lesquelles il faudra se connecter.

Cela nécessite vraisemblablement le maintien d'un annuaire associant chaque requête à une communauté. Pour une recherche de réécriture en mode distribué, cet annuaire peut être construit au fur et à mesure par l'algorithme d'exploration du graphe, pour les communautés sources, une DHT peut éventuellement être utilisée pour donner un point de départ à l'utilisateur.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

relevant : $\mathscr{Q} \times \mathscr{Q} \rightarrow \mathbb{B}$

Cette fonction indique si la seconde requête est pertinente pour réécrire la première.

Elle vise à diminuer les coûts d'organisation en ne considérant dans les demandes de réécritures que les vues qui pourront potentiellement être utilisées ; et en évitant de recourir à des recherches de réécriture si les modifications sur la liste de vue n'ont aucune chance d'avoir un impact.

Au cas où il ne serait pas possible d'utiliser cette notion de pertinence (non-gérée par le langage), cette fonction est censée renvoyer toujours vrai, les faux positifs ne provoquant qu'un surcoût de traitement, tandis que les faux négatifs peuvent dégrader le fonctionnement.

rewrite : $\mathscr{Q} \times 2^{\mathscr{Q}} \rightarrow 2^{\mathscr{Q}}$

Cette fonction fait appel au système de réécriture (entité externe fournie par le langage de requête, et non spécifique à QTor) afin de générer des requêtes réécrites, qui pourront ensuite être sélectionnées par le modèle de coût. Une limite au nombre maximal de réécritures trouvées peut potentiellement être fixée (dans les attributs du système de réécriture) afin d'éviter les traitements trop coûteux.

Gestion du graphe

explore : $\mathscr{Q} \rightarrow 2^{\mathscr{C}} \times 2^{\mathscr{Q}}$

Cet algorithme ne déclenche lui-même aucune modification du graphe ; mais renvoie, d'une part, les équivalences trouvées, et d'autre part, la liste des vues pertinentes pour le cas où il faudrait créer une nouvelle communauté et donc lui générer une nouvelle réécriture.

Ne nécessitant que des informations pouvant être obtenues en contactant directement les communautés, cet algorithme peut être appliqué soit par un tracker centralisé, soit directement par un participant ne connaissant pas encore le graphe (à partir du moment où il est capable d'identifier les communautés sources). Dans ce dernier cas, l'annuaire des associations entre requêtes et communautés doit être constitué en cours de route.

Cette version étendue de l'algorithme tient compte de la possibilité que l'équivalence des requêtes ne soit pas entièrement déterminable : elle renvoie donc une liste de communautés qui ont été démontrées comme équivalentes, pouvant potentiellement comporter plusieurs éléments. Le système devra ensuite se charger de les fusionner le cas échéant.

Algorithme 1 (*QTor: explore*)

1: **inputs**: $q \in \mathscr{Q}$
2: **outputs**: $C_E \subset \mathscr{C}$, $Q \subset \mathscr{Q}$
3: **variables**: $C_S \subset \mathscr{C}$, $c_x \in \mathscr{C}$
4:
5: $C_E, Q \leftarrow \emptyset, \emptyset$
6: ***if*** $\neg q.isSource$ ***then***
7: $\quad C_S \leftarrow providers(q)$
8: $\quad$ ***while*** $C_S \neq \emptyset$ ***do***
9: $\quad\quad$ **remove** *a community* c_x **from** C_S
10: $\quad\quad$ ***if*** $relevant(q, c_x.q) \wedge c_x.q \notin Q$ ***then***
11: $\quad\quad\quad$ *reference the association between* $c_x.q$ *and* c_x
12: $\quad\quad\quad$ ***if*** $c_x.q \equiv q$ ***then***

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

```
13:         insert c_x into C_E
14:       else
15:         insert c_x.q into Q
16:         insert all c_x.successors() into C_S
17:       end if
18:     end if
19:   end while
20: end if
21: return ⟨C_E, Q⟩
```

$createCommunity : \mathscr{U} \times 2^{\mathscr{Q}} \to \mathscr{C}$

Cet algorithme génère une nouvelle communauté pour l'unité passée en paramètre, détermine la liste des réécritures possibles, sélectionne le placement le plus approprié, et provoque le raccordement de la nouvelle communauté à l'hypergraphe existant. Il peut être géré soit de manière centralisée par un tracker, soit directement par le premier participant qui rejoindra la nouvelle communauté.

Un système Pub/Sub distinct du système général gère le signalement d'apparition ou de disparition de communautés. Chaque communauté doit donc y être inscrite (l'inscription étant éventuellement paramétrée par les sources concernées). Cet algorithme gère également l'inscription de la nouvelle communauté et la publication de l'information concernant son apparition.

Algorithme 2 (*QTor: createCommunity*)

```
 1: inputs: u ∈ 𝒰, Q ⊂ 𝒬
 2: outputs: c ∈ 𝒞
 3: variables: r ∈ 𝒬, c_x ∈ 𝒞
 4:
 5: c.q ← u.q
 6: c.U ← {u}
 7: if Q ≠ ∅ then
 8:   select from rewrite(c.q, Q) the rewriting r having the minimal cost(r)
 9:   c.move(r)
10: end if
11: register c to the community publish/subscribe system
12: publish the creation of c over it
13: return c
```

$fusion : 2^{\mathscr{C}} \times \mathscr{U} \to \mathscr{C}$

Cet algorithme est appelé lorsque l'on se rend compte que plusieurs communautés jusque là considérées comme distinctes traitent en fait la même communauté et doivent être fusionnées. Il a deux cas de déclenchements : soit suite à publication de la création d'une nouvelle communauté, soit lors de l'insertion d'un nouvel utilisateur. Dans ce dernier cas, l'unité correspondante est intégrée immédiatement, pour éviter une réorganisation supplémentaire ensuite.

Il y a un risque d'attaque à ce niveau : un participant peut signaler que deux communautés traitent une requête équivalente alors que ce n'est pas le cas. Il convient de

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

faire les vérifications nécessaires, mais cette partie n'est pas intégrée dans l'algorithme ci-dessous pour simplifier (l'équivalence pouvant ne pas avoir été fournie plus tôt parce que difficile à détecter, des informations complémentaires de la part du participant peuvent être requises).

Algorithme 3 (*QTor: fusion*)

1: **inputs**: $C \subset \mathscr{C}$, $u \in \mathscr{U}$
2: **outputs**: $c \in \mathscr{C}$
3: **variables**: $R \subset \mathscr{Q}$, $r \in \mathscr{Q}$, $c_x \in \mathscr{C}$
4:
5: *use the normalized query as c.q*
6: $c.U$, $R \leftarrow \emptyset$, $\emptyset$
7: ***for all*** $c_x \in C$ **successively** ***do***
8:     **insert** *all* $c_x.U$ **into** $c.U$
9:     **insert** $c_x.r$ **into** R
10:     ***if*** $\neg c_x.h.isAnchored$ ***then***
11:         $c_x.h.p.freeUnit(c_x.h)$
12:     ***end if***
13: ***end for***
14: ***if*** $u \neq null$ ***then***
15:     **insert** u **into** $c.U$
16: ***end if***
17: **select from** R *the rewriting* r **having** *the minimal* $cost(r)$
18: $c.move(r)$
19: *register c to the community publish/subscribe system*
20: *publish the creation of c over it*
21: ***return*** c

Entité 1 (*QTor*)

- *cost* : $\mathscr{Q} \rightarrow \mathbb{R}$ *Détermine et renvoie le coût associé à une requête, afin de guider les choix de placement.*
- *providers* : $\mathscr{Q} \rightarrow 2^{\mathscr{C}}$ *Détermine et renvoie l'ensemble des communautés auprès desquelles les données doivent être acquises pour traiter la requête.*
- *relevant* : $\mathscr{Q} \times \mathscr{Q} \rightarrow \mathbb{B}$ *Détermine et renvoie si la seconde requête fournie en paramètre est pertinente ou non pour réécrire la première.*
- *rewrite* : $\mathscr{Q} \times 2^{\mathscr{Q}} \rightarrow 2^{\mathscr{Q}}$ *Génère autant de réécritures que possible de la requête en utilisant les vues fournies.*
- *explore* : $\mathscr{Q} \rightarrow 2^{\mathscr{C}} \times 2^{\mathscr{Q}}$ *Explore l'hypergraphe des communautés à la recherche d'un placement.*
- *createCommunity* : $\mathscr{U} \times 2^{\mathscr{Q}} \rightarrow \mathscr{C}$ *Crée d'une nouvelle communauté.*
- *fusion* : $2^{\mathscr{C}} \times \mathscr{U} \rightarrow \mathscr{C}$ *Fusionne plusieurs communautés traitant la même requête.*

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Query

$isSource \in \mathbb{B}$

Cette propriété indique s'il s'agit d'une véritable requête utilisateur (elle vaut faux dans ce cas), ou bien d'une source de donnée que l'on représente par une expression du langage de requêtes pour uniformiser (elle vaut alors vrai).

Entité 2 (*Query* $\in \mathscr{Q}$)

- $isSource \in \mathbb{B}$ *Indique s'il s'agit d'une requête source.*

Participant

Réprésente un participant du système : une entité de ce type est générée chez chaque utilisateur rejoignant le système QTor. Elle sert à référencer les requêtes lancées et à communiquer avec le système général.

Gestion des unités

$capa \in \mathbb{N}$

Afin de classer plus simplement les participants sans avoir besoin de coordination entre les communautés, chaque participant indique au système ses capacités totales, indépendamment de la proportion de ressources allouées à chaque unité. En cas de modification, le participant peut relancer une réorganisation de ses différentes communautés, voire quitter les communautés qui ne lui sont pas utiles et consomment trop de ressources.

$Q \subset \mathscr{Q}$

Chaque participant au système est caractérisé par les requêtes qu'il traite. Celles-ci peuvent correspondre soit à des requêtes directement exprimées par l'utilisateur, soit à des pseudo-requêtes générées pour caractériser un flux source initial émis par le participant (afin d'uniformiser le traitement des sources et des requêtes), soit à des requêtes ne correspondant pas aux demandes initiales de l'utilisateur, mais que le participant est amené à traiter pour aider au bon fonctionnement du système.

$U \subset \mathscr{U}$

Dans le modèle de fonctionnement de QTor, un participant contribue par un ensemble d'unités de calcul, correspondant à un ensemble de ressources de traitement et de diffusion, chacune dédiée à une requête en particulier. Chaque participant doit donc être en mesure d'indiquer l'unité associée à chacune des requêtes qu'il traite : il existe une bijection entre les éléments de U et ceux de Q.

$createUnit : \mathscr{Q} \times \mathbb{B} \rightarrow \mathscr{U}$

Un participant peut être amené à créer de nouvelles unités pour enrichir sa participation au système. Les unités sont ancrées si elles correspondent à une demande utilisateur,

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

et ne le sont pas s'il s'agit d'une requête traitée pour des raisons fonctionnelles, sans être demandée par l'utilisateur.

$freeUnit : \mathscr{U} \rightarrow \emptyset$

Les ressources utilisées sans correspondre à une demande utilisateur peuvent être libérées à tout moment, si elles ne sont plus requises au bon fonctionnement du système. Cette fonction n'est normalement appelée que sur les unités qui ne sont pas ancrées.

Communication avec le système

QTor

Un participant est toujours en mesure de contacter le système dont il fait partie.

$accept : \mathscr{Q} \rightarrow \mathbb{B}$

Un système QTor n'affectant pas de tâches à un participant de manière arbitraire, un contrôle est requis lorsqu'un participant peut être amené à travailler sur une requête (en dehors du cas de l'acquisition auprès d'une communauté parente, qui, si elle doit avoir lieu, est une nécessité technique).

L'utilisateur doit donc être en mesure de paramétrer ses propres choix de participation (en fonction du nombre d'unités présentes dans le système, de la quantité de ressources disponibles et demandées, de la nature des requêtes…). Cette fonction indique si la requête est acceptable ou non, d'après la politique de participation définie par l'utilisateur.

$anchor : \mathscr{Q} \rightarrow \emptyset$

Cet algorithme permet au participant de lancer une nouvelle requête, suite à une demande utilisateur.

Si une unité correspondante existe déjà, celle-ci est « ancrée » ; dans le cas contraire, une exploration du graphe est lancée pour rejoindre une communauté existante ou en créer une à la bonne place.

Algorithme 4 (*Participant: anchor*)

1: **inputs**: $q \in \mathscr{Q}$
2: **outputs**: *nothing*
3: **variables**: $u \in \mathscr{U}$, $C_E \subset \mathscr{C}$, $Q \subset \mathscr{Q}$, $c \in \mathscr{C}$
4:
5: ***if*** $\exists u \in U,\ u.q \equiv q$ ***then***
6: $u.isAnchored \leftarrow true$
7: ***else***
8: $u \leftarrow createUnit(q,\ true)$
9: $C_E, Q \leftarrow QTor.explore(q)$
10: ***if*** $C_E.length > 0$ ***then***
11: ***if*** $C_E.length > 1$ ***then***
12: $QTor.fusion(C_E,\ u)$
13: ***else***
14: *c is the unique element of* C_E
15: $c.join(u)$
16: ***end if***

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

17: **else**
18: $QTor.createCommunity(u, Q)$
19: **end if**
20: **end if**

Entité 3 (*Participant* $\in \mathscr{P}$)

- *capa* $\in \mathbb{N}$ *Indique les capacités totales du participant.*
- $Q \subset \mathscr{Q}$ *Ensemble des requêtes traitées par le participant (exprimées ou non).*
- $U \subset \mathscr{U}$ *Ensemble des unités affectées par le participant.*
- *QTor Référence au système général*
- *createUnit* : $\mathscr{Q} \times \mathbb{B} \rightarrow \mathscr{U}$ *Crée une unité (ancrée ou non) pour traiter la requête demandée.*
- *freeUnit* : $\mathscr{U} \rightarrow \emptyset$ *Libère les ressources d'une unité superflue.*
- *accept* : $\mathscr{Q} \rightarrow \mathbb{B}$ *Indique si le participant accepte de traiter une nouvelle requête.*
- *anchor* : $\mathscr{Q} \rightarrow \emptyset$ *Algorithme appelé lorsque le participant souhaite effectuer une nouvelle requête.*

Unit

Représente les ressources allouées par un participant au calcul d'une requête donnée et à la diffusion des résultats correspondants.

Informations sur les fonctions de l'unité

$q \in \mathscr{Q}$

Chaque unité correspond à une requête unique dans le système, et doit donc pouvoir indiquer sur quelle requête elle travaille. Une unité peut cependant se voir affecter un traitement à utiliser différent de cette requête (soit une requête réécrite entière, si elle est seule en charge du traitement de la requête, soit un ensemble d'opérations permettant d'obtenir une partie des résultats si le travail se fait de manière distribuée).

$isAnchored \in \mathbb{B}$

Des unités pouvant être créées pour des raisons fonctionnelles, il convient de les distinguer de celles créées pour répondre à une demande utilisateur (les unités ancrées n'ont aucune raison de disparaître tant que l'utilisateur n'en a pas fait la demande ; tandis que les unités non-ancrées peuvent être libérées si les ressources qu'elles consomment ne sont plus nécessaires).

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Informations de mise en relation

$c \in \mathscr{C}$

Dans un système QTor, chaque unité a pour vocation de rejoindre une communauté, constituée de l'ensemble des unités travaillant sur des requêtes équivalentes à la sienne. Chaque unité doit être en mesure, en permanence, de communiquer avec le reste de sa communauté.

$p \in \mathscr{P}$

Chaque unité est gérée par un participant donné ; la communauté, connaissant l'ensemble de ses unités, peut contacter le participant concerné par leur intermédiaire.

Informations sur les ressources disponibles

$fanout : \emptyset \to \mathbb{N}$

L'envoi de données aux autres participants consommant des ressources, chaque unité est associée à un degré sortant maximal, qui ne doit pas être dépassé par le système. Celui-ci peut être fonction de la quantité de données attendues en sortie de la requête. À moins d'un mécanisme d'allocation fixe, les capacités d'une unité donnée sont considérées comme égales aux capacités totales du participant moins les capacités actuellement utilisées par ses autres unités.

$performable : \mathscr{Q} \to \mathbb{B}$

Tant que les calculs ne sont pas distribués, il est nécessaire qu'une unité de la communauté ait les ressources de calcul suffisantes pour procéder au traitement de la requête. Le modèle de coût permet de sélectionner la réécriture la moins coûteuse, mais ceci peut toutefois s'avérer encore hors de portée pour certaines unités limitées. Cette fonction permet de vérifier si une unité est en mesure de traiter seule la réécriture qu'on lui propose.

Entité 4 (*Unit* $\in \mathscr{U}$)

- $q \in \mathscr{Q}$ *Requête traitée par l'unité.*
- $isAnchored \in \mathbb{B}$ *Indique si l'unité est ancrée, c'est-à-dire correspond à une demande utilisateur.*
- $c \in \mathscr{C}$ *Communauté à laquelle appartient l'unité.*
- $p \in \mathscr{P}$ *Référence au participant gérant l'unité.*
- $fanout : \emptyset \to \mathbb{N}$ *Indique la capacité maximale de diffusion de l'unité.*
- $performable : \mathscr{Q} \to \mathbb{B}$ *Indique si l'unité est capable de traiter la réécriture.*

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Community

Informations publiques sur la communauté

$q \in \mathscr{Q}$

Chaque communauté est dédiée au traitement d'une requête en particulier, et peut donc renseigner sur la requête sur laquelle elle travaille.

$successors : \emptyset \to 2^{\mathscr{C}}$

Chaque communauté peut renseigner (parce que cette information est mise en cache, ou bien par une demande à l'ensemble de ses participants) sur l'ensemble des communautés qui récupèrent des informations auprès d'elle (ses successeurs dans le graphe logique).

$isActive \in \mathbb{B}$

Une communauté peut éventuellement ne contenir aucune unité suffisamment puissante pour traiter sa requête, même si l'usage des réécritures permet de réduire les coûts. Dans ce cas, elle est placée dans le graphe et l'information de son arrivée correspond également à une demande de ressources : les autres communautés du système sont susceptibles de lui envoyer une unité suffisamment puissante pour qu'elle puisse fonctionner convenablement.

Communication avec le système

$QTor$

Une communauté est toujours en mesure de contacter le système dont elle fait partie.

$registerView : \mathscr{C} \to \emptyset$

Cet algorithme est appelé lorsque la communauté est informée, par l'intermédiaire d'une publication dans le système Pub/Sub des communautés, de l'apparition d'une nouvelle vue pouvant potentiellement l'intéresser.

Algorithme 5 (*Community: registerView*)

```
1:  inputs: c_x ∈ 𝒞
2:  outputs: nothing
3:  variables: u ∈ 𝒰 , c_e ∈ 𝒞
4:
5:  if c_x ≠ this then
6:      if c_x.q ≡ this.q then
7:          QTor.fusion({this, c_x}, null)
8:      else
9:          if ¬c_x.isActive then
10:             u ← this.askHelp(c_x.r)
11:             if u ≠ null then
12:                 c_x.join(u)
13:             end if
14:         end if
15:         if QTor.relevant(q, c_x.q) then
16:             insert c_x.q into this.Q
```


Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

17: *reevaluatePlacement()*
18: ***end if***
19: ***end if***
20: ***end if***

$unregisterView : \mathscr{C} \rightarrow \emptyset$

Symétrique du précédent, cet algorithme est appelé lorsque la communauté est informée, par l'intermédiaire d'une publication dans le système Pub/Sub des communautés, de la disparition d'une vue qu'elle pouvait potentiellement utiliser.

Algorithme 6 (*Community: unregisterView*)

1: **inputs**: $c_x \in \mathscr{C}$
2: **outputs**: *nothing*
3: **variables**: *nothing*
4:
5: ***if*** $c_x.q \in this.Q$ ***then***
6: **remove** $c_x.q$ **from** $this.Q$
7: ***if*** $c_x \in QTor.providers(this.r)$ ***then***
8: $c_x.r \leftarrow null$
9: *reevaluatePlacement()*
10: ***end if***
11: ***end if***

$askHelp : \mathscr{Q} \rightarrow \mathscr{U}$

Cet algorithme est appelé lorsqu'une autre communauté a besoin d'aide pour fonctionner. Cela peut survenir soit lors d'un branchement d'une communauté enfant, soit lors de l'apparition d'une nouvelle communauté (dans le cas décrit par l'algorithme précédent). Le principe est ici d'interroger tous les membres de la communauté afin de proposer la meilleure aide possible.

Algorithme 7 (*Community: askHelp*)

1: **inputs**: $r \in \mathscr{Q}$
2: **outputs**: $u \in \mathscr{U}$
3: **variables**: $U \subset \mathscr{U}$
4:
5: $U \leftarrow \emptyset$
6: ***for all*** $u \in this.U$ **concurrently** ***do***
7: ***if*** $u.p.accept(r)$ ***then***
8: **insert** $u.p.createUnit(r, false)$ **into** U
9: ***end if***
10: ***end for***
11: ***if*** $U = \emptyset$ ***then***
12: $u \leftarrow null$
13: ***else***
14: $u \leftarrow choice(U, r)$
15: *free the other units of U*
16: ***end if***

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

17: ***return*** u

Gestion du placement dans le graphe

$r \in \mathscr{Q}$

À chaque communauté est affectée une réécriture, qui correspond à sa position dans l'hypergraphe logique (les bases de l'hyperarc entrant correspondent aux vues utilisées).

$Q \subset \mathscr{Q}$

Afin d'éviter de coûteuses explorations successives du graphe, chaque communauté maintient un ensemble des vues précédemment identifiées comme pertinentes, mis à jour en fonction des informations publiées dans le système Pub/Sub des communautés.

$choice : 2^{\mathscr{U}} \times \mathscr{Q} \to \mathscr{U}$

Cette fonction choisit, parmi un ensemble d'unités candidates, la plus appropriée pour servir de racine pour une communauté. La même fonction de choix est normalement utilisée qu'il s'agisse de choisir une racine pour la communauté elle-même, ou pour envoyer des ressources vers une autre communauté ; mais des différences de traitement peuvent être envisagées. Cela tient en tout cas compte des capacités des différentes unités, ainsi que du taux de présence dans les communautés auprès desquelles la réécriture fournie va chercher ses données.

$h \in \mathscr{U}$

Tant que le calcul n'est pas partagé, une unique unité est en charge d'acquérir les données auprès des communautés parentes, de procéder au calcul de la requête et de servir de racine à l'arbre de diffusion interne. Cette unité particulière, désignée comme étant la tête de communauté, peut être soit une unité originaire de la communauté, soit une ressource envoyée par l'une des communautés parentes.

$reevaluatePlacement : \emptyset \to \emptyset$

Cet algorithme est appelé suite aux apparitions ou disparitions de vues, ou lorsqu'un changement de volumétrie dans les données entrantes conduit la communauté à s'interroger sur la pertinence de changer de réécriture. Il s'agit d'un élément critique, nécessitant un mécanisme d'exclusion mutuelle, afin d'empêcher deux communautés de se déplacer simultanément, car il s'agirait du seul cas pour lequel des cycles risqueraient d'apparaître dans le graphe.

Algorithme 8 (*Community: reevaluatePlacement*)

1: **inputs**: *nothing*
2: **outputs**: *nothing*
3: **variables**: $Q \subset \mathscr{Q}$, $r \in \mathscr{Q}$
4:
5: *build* Q *by selecting all queries from* $this.Q$ *that will not cause loops*
6: **select from** $QTor.rewrite(q, Q)$ *the rewriting* r **having** *the minimal* $QTor.cost(r)$
7: ***if*** $r \neq this.r$ ***then***
8: $\quad this.move(r)$

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

9: ***end if***

$move : \mathscr{Q} \to \emptyset$

Cette fonction prend en charge le branchement effectif d'une communauté dans l'hypergraphe, en sélectionnant une tête de communauté pour gérer la réécriture fournie.

Algorithme 9 (*Community: move*)

1: **inputs**: $r \in \mathscr{Q}$
2: **outputs**: *nothing*
3: **variables**: $U \subset \mathscr{U}$, $c_p \in \mathscr{C}$, $u \in \mathscr{U}$
4:
5: $U \leftarrow \emptyset$
6: ***for all*** $c_p \in QTor.providers(r)$ **concurrently** ***do***
7:    ***if*** $c_p.askHelp(r) \neq null$ ***then***
8:       *add this unit into* U
9:    ***end if***
10: ***end for***
11: **synchronize**
12: ***if*** $U \neq \emptyset$ ***then***
13:    $this.isActive \leftarrow true$
14:    $this.h \leftarrow this.choice(U, r)$
15:    **insert** $this.h$ **into** $this.U$
16:    ***for all*** $u \in U$ **concurrently** ***do***
17:       ***if*** $u \neq this.h$ ***then***
18:          $u.p.freeUnit(u)$
19:       ***end if***
20:    ***end for***
21: ***else***
22:    ***for all*** $u \in this.U$ **concurrently** ***do***
23:       ***if*** $u.performable(r)$ ***then***
24:          **insert** u **into** U
25:       ***end if***
26:    ***end for***
27:    ***if*** $U = \emptyset$ ***then***
28:       $this.isActive \leftarrow false$
29:       *temporarily select the first unit of* $this.U$ *for* $this.h$
30:    ***else***
31:       $this.isActive \leftarrow true$
32:       $this.h \leftarrow this.choice(U, r)$
33:    ***end if***
34: ***end if***
35: ***for all*** $c_p \in QTor.providers(r)$ **concurrently** ***do***
36:    ***if*** $c_p.q \notin this.h.p.Q$ ***then***
37:       $u \leftarrow this.h.p.createUnit(c_p.q, false)$
38:       $c_p.join(u)$
39:    ***end if***
40: ***end for***
41: *reorganization()*

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Organisation interne

$U \subset \mathscr{U}$

Cette liste, utilisée notamment pour l'organisation interne, regroupe l'ensemble des participants de la communauté. Elle peut être directement maintenue ordonnée, ou ne l'être qu'au moment du déploiement de l'arbre de diffusion (solution préférée ici pour demeurer explicite).

$join : \mathscr{U} \rightarrow \emptyset$

Cet algorithme est appelé lors de l'insertion d'un nouvel utilisateur dans une communauté. Il vérifie s'il doit y avoir remise en cause ou non de la tête de communauté (qui, hormis ce cas, n'est modifiée que lors d'un appel à l'algorithme « move »), et déclenche une réorganisation de la communauté.

Algorithme 10 (*Community: join*)

```
1: inputs: u ∈ 𝒰
2: outputs: nothing
3: variables: c_p ∈ 𝒞, u_p ∈ 𝒰
4:
5: u.c ← this
6: insert u into this.U
7: if u starts this.U then
8:    if this.h.isAnchored ∧ u.performable(this.r) then
9:       free the related unanchored units of this.h.p
10:      this.h, this.isActive ← u, true
11:      for all c_p ∈ QTor.providers(r) concurrently do
12:         if c_p.q ∉ u.p.Q then
13:            u_p ← u.p.createUnit(c_p.q, false)
14:            c_p.join(u_p)
15:         end if
16:      end for
17:   end if
18: end if
19: this.reorganization()
```

La condition « this.h.isAnchored » au début du second if a pour but d'éviter de remettre en cause une tête de communauté lorsque celle-ci était envoyée par une communauté parente, considérant qu'elle est alors mieux intégrée au reste du graphe. Ce choix est discutable et peut, selon le cas, être remis en cause ; mais il faut alors veiller à libérer les ressources le cas échéant.

Par ailleurs, il est possible, avant l'appel à la fonction de réorganisation (ligne 19) de pratiquer certaines optimisations de latence en demandant au nouveau venu de prendre en charge l'envoi de ressources vers des communautés enfants jusque là gérées par des participants moins puissants.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

leave : $\mathscr{U} \to \emptyset$

Symétriquement, cet algorithme est appelé lors de la disparition d'une unité autrefois membre de la communauté. Étant donné le mécanisme d'échange de ressources, la communauté ne peut être menacée de disparition que si elle est une feuille, ou bien un nœud intermédiaire ne correspondant qu'au traitement interne d'un participant.

Algorithme 11 (*Community: leave*)

1: **inputs**: $u \in \mathscr{U}$
2: **outputs**: *nothing*
3: **variables**: $del \in \mathbb{B}$, $u_o \in \mathscr{U}$
4:
5: **remove** u **from** $this.U$
6: ***if*** $\exists u_o \in this.U$, $u_o.isAnchored$ ***then***
7:     $del \leftarrow false$
8: ***else***
9:     ***if*** $this.U.length = 0$ ***then***
10:         $del \leftarrow true$
11:     ***end if***
12:     ***if*** $this.successors() = \emptyset$ ***then***
13:         *free the remaining units*
14:         $del \leftarrow true$
15:     ***end if***
16:     ***if*** $this.U.length = 1$ ***then***
17:         *inform the participant it can free this if needed*
18:     ***end if***
19: ***end if***
20: ***if*** del ***then***
21:     *publish the disparition of this in the community publish/subscribe system*
22: ***else***
23:     ***if*** $u = this.h$ ***then***
24:         $this.move(this.r)$
25:     ***else***
26:         $this.reorganization()$
27:     ***end if***
28: ***end if***

reorganization : $\emptyset \to \emptyset$

Cet algorithme gère les réorganisations de l'arbre de diffusion au sein de la communauté, après une modification de la liste des unités présentes. Afin de ne pas consommer trop de liens réseaux, il fonctionne étage par étage, et ne remet en cause les branchements existants que lorsque la position relative de l'unité parente et de l'unité enfant évolue (étant entendu que l'identité de l'unité parente d'une autre importe peu tant que la distance entre cette unité et la tête de communauté reste fixe).

Algorithme 12 (*Community: reorganization*)

1: **inputs**: *nothing*
2: **outputs**: *nothing*

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

3: **variables**: $U \subset \mathscr{U}$, $P \subset \mathscr{U}$, $N \subset \mathscr{U}$, $u \in \mathscr{U}$, $n \in \mathbb{N}$
4:
5: $U \leftarrow this.U$
6: **remove** *this.h* **from** U
7: *order if needed* U *regarding to the participants' capacities*
8: $P \leftarrow \{this.h\}$
9: ***while*** $U \neq \emptyset$ ***do***
10: $N \leftarrow \emptyset$
11: $n \leftarrow \Sigma(u.fanout \ \forall u \in P)$
12: *remove the* n *first units of* U *and insert them into* N
13: ***for all*** $u \in P$ **successively** ***do***
14: *unlink all children of* u *that are not in* N
15: ***end for***
16: ***for all*** $u \in N$ **successively** ***do***
17: *unlink* u *from its parent if this parent is not in* P
18: *link* u *from the first available parent in* P *if* u *has no parent*
19: ***end for***
20: $P \leftarrow N$
21: ***end while***

Il s'est avéré qu'ordonner les unités en fonction des capacités totales des participants (et non du fanout spécifique de ces unités) était le plus intéressant pour diminuer la latence globale sans provoquer de conflit entre plusieurs variables. Cela conduit par ailleurs à ordonner les participants de la même manière dans les différences communautés (hors choix de la racine), ce qui évite les transferts croisés.

Entité 5 (*Community* $\in \mathscr{C}$)

- $q \in \mathscr{Q}$ *Requête caractérisant la communauté.*
- $isActive \in \mathbb{B}$ *Indique si la communauté est active (ressources suffisantes).*
- $QTor$ *Référence au système général.*
- $r \in \mathscr{Q}$ *Réécriture actuellement traitée.*
- $Q \subset \mathscr{Q}$ *Liste des requêtes mémorisées pour d'éventuelles réécritures.*
- $h \in \mathscr{U}$ *Tête de communauté : unité chargée d'acquérir les données et de traiter la requête*
- $U \subset \mathscr{U}$ *Liste des unités présentes dans la communauté.*
- $successors : \emptyset \rightarrow 2^{\mathscr{C}}$ *Renvoie la liste des communautés enfant de cette communauté.*
- $registerView : \mathscr{C} \rightarrow \emptyset$ *Prévient la communauté de l'apparition d'une nouvelle vue dans le graphe.*
- $unregisterView : \mathscr{C} \rightarrow \emptyset$ *Prévient la communauté de l'apparition d'une nouvelle vue dans le graphe.*

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

- *askHelp* : $\mathscr{Q} \rightarrow \mathscr{U}$ *Demande d'aide de la part d'une autre communauté.*
- *choice* : $2^{\mathscr{U}} \times \mathscr{Q} \rightarrow \mathscr{U}$ *Sélectionne une unité pour servir de racine à une communauté (celle-ci ou une autre)*
- *reevaluatePlacement* : $\emptyset \rightarrow \emptyset$ *Réévalue le placement de la communauté.*
- *move* : $\mathscr{Q} \rightarrow \emptyset$ *Connecte la communauté à aux parents indiqués par la réécriture fournie.*
- *join* : $\mathscr{U} \rightarrow \emptyset$ *Insertion d'une nouvelle unité dans la communauté.*
- *leave* : $\mathscr{U} \rightarrow \emptyset$ *Départ d'une unité de la communauté.*
- *reorganization* : $\emptyset \rightarrow \emptyset$ *Réorganisation totale de l'arbre de diffusion de la communauté.*

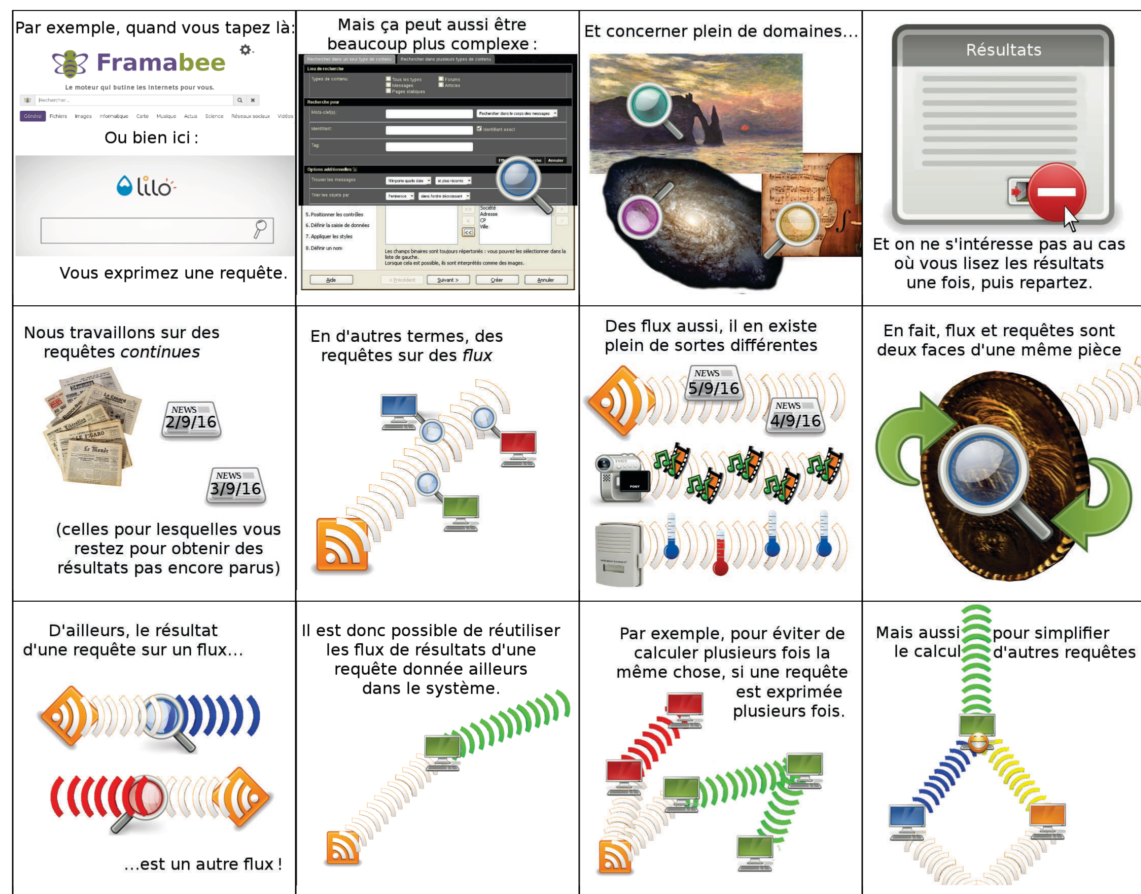
Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Annexe 5

Ma thèse... en BD !

Le but de cette dernière annexe est de présenter notre proposition de manière aussi accessible que possible, en reprenant notamment une grande partie des schémas présentés dans les autres chapitres. L'idée d'un format de bandes dessinées vient du nom de l'équipe dans laquelle nous travaillons au sein du LIRIS : l'équipe BD, pour « bases de données ».

Commençons par le début. La notion de base, pour comprendre ce sur quoi nous travaillons, c'est la notion de *requête*. C'est ce que vous exprimez pour obtenir les informations qui vous intéressent.



Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Les requêtes étant exprimées par un grand nombre de *participants*, il faut faire en sorte que chacun puisse obtenir les résultats attendus dans de bonnes conditions, ce qui peut s'avérer assez complexe.

Notre problème, c'est d'organiser le système.

Donc de mettre les participants en relation entre eux

...mais ça ne se fait pas n'importe comment non plus.

Il y a des *contraintes* à respecter :

20

Par exemple, il ne faut pas créer de cycles :

???

sinon, les flux n'arrivent plus...

Il faut que chaque participant reçoive les infos dont il a besoin

≠ !

Et les capacités sont limitées, il faut éviter de les dépasser

Et éviter les trop longues attentes

Tout ça fait que l'organisation peut être très compliquée

Ce qui peut s'avérer assez gênant aussi...

Plus on peut déterminer comment doit marcher le système facilement,

mieux c'est.

(Et si on peut, au passage, avoir un système stable,

???

ce n'est pas mal non plus)

Pour s'y retrouver, schématisons un peu tout ça.

⇒

⇒

On représente chaque participant par une forme...

...et chaque requête par une couleur

Un participant qui fait plusieurs requêtes sera donc présenté comme ça :

Ou encore comme ça :

Chaque participant est caractérisé par :

- ses requêtes

- ses capacités

On représente donc le système comme un graphe :

Les relations entre participants indiquent qui fait quoi dans le système

Récupérer flux jaune chez rond
Diffuser flux jaune à triangle
Récupérer flux bleu chez carré
Créer flux vert par mélange
Diffuser flux vert à losange

Vu par notre démonstrateur, ça va ressembler à ça :

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Plusieurs façons de répondre à des problèmes du même genre ont été proposées. Certaines sont assez proches de la nôtre, d'autres en sont plutôt très éloignées. Aucune ne nous satisfait complètement.

On peut demander à la source de faire tout le travail...

Ça peut simplifier les calculs, mais elle est vite surchargée

S'il y a trop de monde, on peut mettre en place d'autres machines...

Mais ça peut coûter cher, et on ne leur fait pas forcément confiance...

On peut s'entre-aider sur la diffusion du flux de départ...

Mais ça empêche de s'entre-aider pour les calculs

...et si on choisit plutôt de s'entre-aider sur les flux de résultats...

C'est bien... ...sauf si la source doit encore faire tous les calculs

Il y a des systèmes complexes où chacun bosse sur un morceau du travail commun

Mais c'est dur de s'y retrouver, et chacun peut être amené à bosser sur des requêtes sans rapport avec la sienne

Dans SemPO[16], on dit aux participants de récupérer les résultats qui contiennent le moins de données en trop

C'est pratique, parce que si un autre fait la même requête qu'eux, les bons résultats tombent tout prêts

Le hic, c'est que ça ne tient pas compte des limites de capacités...

Dans Delta[53], on utilise une organisation globale de l'ensemble des calculs

On commence par calculer toutes les relations entre toutes les requêtes de tout le monde...

En supprimant les cycles au passage pour éviter les risques...

Puis on appelle un ILP (un truc qui fait du sudoku) pour que chacun récupère les bons flux sans surcharge

Et enfin, un algo vient replacer les participants pour diminuer le temps d'attente des données

C'est pas mal, mais un peu long et compliqué à organiser...

Et plus les participants s'intéressent aux mêmes résultats... ...plus c'est dur.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Notre idée, c'est de nous baser sur les similitudes entre requêtes. En particulier, de s'appuyer sur le fait que certains participants vont rechercher les mêmes résultats pour simplifier l'organisation.

Voici un système. Il y a plein de relations entre les requêtes, donc c'est compliqué.

Si on réunit entre eux tous les participants dont les requêtes sont *équivalentes*, ça devient déjà beaucoup plus simple

On peut maintenant choisir les relations qu'on va utiliser entre les requêtes sans trop se casser la tête...

Et il n'y a ensuite plus qu'à relier les différents participants entre eux pour de vrai.

Un groupe de participants dont les requêtes sont équivalentes, on appelle ça une *communauté*

C'est le concept principal qu'on propose dans notre approche

Les communautés forment une *abstraction* pour guider l'organisation

⇒

On peut les utiliser sans devoir prendre en compte le détail des participants à l'intérieur.

Concrètement, imaginons un participant qui veut s'insérer dans un système existant.

?

Tout ce qu'il connaît, au début, c'est la source sur laquelle est exprimée sa requête.

Il va demander à cette source à quelles communautés elle envoie ses flux. C'est l'étape 1.

?

Certaines communautés ont des requêtes qui l'intéressent, d'autres pas.

?

On laisse tomber celles qui ne l'intéressent pas.

Mais on continue d'explorer à partir des autres.

?

Si une communauté travaille sur une requête équivalente, on la trouve comme ça.

Sinon, il va falloir en créer une nouvelle.

Pour savoir où la brancher, on regarde les communautés qui avaient été intéressantes :

Ne pas explorer tout n'est pas grave: celles qu'on ne visite pas ne vont pas être utiles

?

Une fois la nouvelle communauté branchée, on informe le reste du système de son arrivée

En effet, elle peut offrir de nouvelles possibilités pour faciliter le travail des autres

?

?

?

On va donc réorganiser une partie du graphe, et ainsi améliorer les coûts

C'est ici qu'on risque de créer des cycles, alors il faut faire attention

On obtient donc un système dans lequel les communautés s'échangent des flux en cascade (ou en *torrent*)

Q.Tor

D'où le nom de « torrent de requêtes »

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Maintenant, il faut *concrétiser* tout ça, en faisant en sorte de relier les participants entre eux dans les et entre les communautés. Pour chaque communauté, il y a trois choses à faire.

D'abord, il faut *acquérir* les flux dont on a besoin pour la requête

L'acquisition est une étape importante.

Si une communauté doit envoyer son flux à une autre, il faut que les participants se rencontrent.

On peut faire ça en demandant à un membre de la communauté enfant d'envoyer d'intégrer la commauté parente, pour « rembourser » les ressources qu'il consomme

Mais si la communauté parente dispose de suffisamment de ressources, on peut aussi faire l'échange dans l'autre sens...

(À condition que les participants à l'intérieur soient d'accord, bien sûr)

Les deux ont leurs avantages

On choisit entre les deux en fonction de la situation

Dans les deux cas, ça permet d'organiser les deux communautés de façon indépendante.

Ensuite, il faut *calculer* les résultats de celle-ci à partir de ces flux

Répartir le calcul est complexe. Une autre thèse[54] est en cours en ce moment sur ce sujet.

Pour l'instant, on considère qu'un seul participant se charge de calculer les résultats.

Et enfin, il faut *diffuser* ces résultats aux membres de la communautés

L'organisation de la diffusion a pour but de relier tout le monde au plus près possible du participant qui calcule...

Mais, bien sûr, en tenant compte des limitations de capacités !

Par exemple, voyons ce qui se passe quand un nouveau participant doit être inséré...

On commence par ordonner tout le monde en fonction des capacités, du plus au moins puissant.

(En mettant celui qui calcule en premier, évidemment)

On met celui qui calcule tout seul en bas (c'est de lui que tout part, vu qu'il fabrique les résultats)

On prend autant de participants qu'il n'y a de places libres à l'étage suivant de l'arbre, et on les connecte.

Et on recommence tant qu'il reste du monde à connecter.

Sauf que là, on vient de déconnecter celui-ci de celui à qui il était branché, en le laissant à la même distance...

lui, là.

Ça coûte cher pour pas grand chose, ce genre d'opérations.

Donc on vérifie quand même ce genre de choses, pour rendre l'orga plus efficace.

S'il n'y a pas de différence relative entre le participant et son ancien parent, on laisse.

Et en organisant comme ça chaque communauté une par une...

...on finit par avoir organisé facilement le système entier !

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Un aspect intéressant de notre approche est qu'elle permet au système de s'*adapter* assez efficacement aux différentes situations que l'on peut rencontrer dans la vie du système.

La *popularité* des requêtes est facile à prendre en compte

Ici, on voit une requête clairement plus populaire que les autres (la verte)

On aura donc une grosse communauté, et les autres beaucoup plus petites.

Mais on garde quand même une communauté par requête, donc ça simplifie l'organisation au niveau des requêtes.

Et à l'intérieur de cette grosse communauté, il y a seulement à connecter les participants selon leurs capacités : c'est beaucoup plus facile.

Mais que se passe-t-il s'il n'y a aucune requête populaire ?

(Bon, là ça fait beaucoup de requêtes, ce sera plus lisible si on en retire un peu)

On va organiser le système de la même façon au niveau des communautés.

(Même si ça fait un peu beaucoup de communautés)

Aucune communauté ne contiendra beaucoup de monde…

Mais il y aura quand même les échanges de ressources.

Donc chacun pourra quand même recevoir de l'aide.

Et l'organisation interne sera super simple à faire.

Et si la popularité d'une des requêtes se met à beaucoup augmenter en cours de route…

…ça ne rendra pas les choses plus compliquées ☺

Et que se passe-t-il si on ne peut pas mettre les communautés en relation ?

On s'arrangera quand même, avec chaque requête branchée directement sur la source…

Mais ce sera quand même possible de s'entre-aider

Pas de surcharge

(Et chaque communauté traitera sa propre requête)

Concernant les capacités, un aspect intéressant est que l'échange de ressources entre communautés peut se faire sur plusieurs étages

Ce qui donne une grande souplesse.

Par exemple, voici un système.

Il a cinq communautés, et des participants de capacités variables.

Si la source est assez puissante, c'est elle qui va monter calculer pour tout le monde.

On va donc obtenir un de ces systèmes où la source fait tout… mais parce qu'on est sûrs qu'elle peut le faire.

Au contraire, si tous les participants sont très peu puissants, chacun descend chercher seulement ce dont il a besoin.

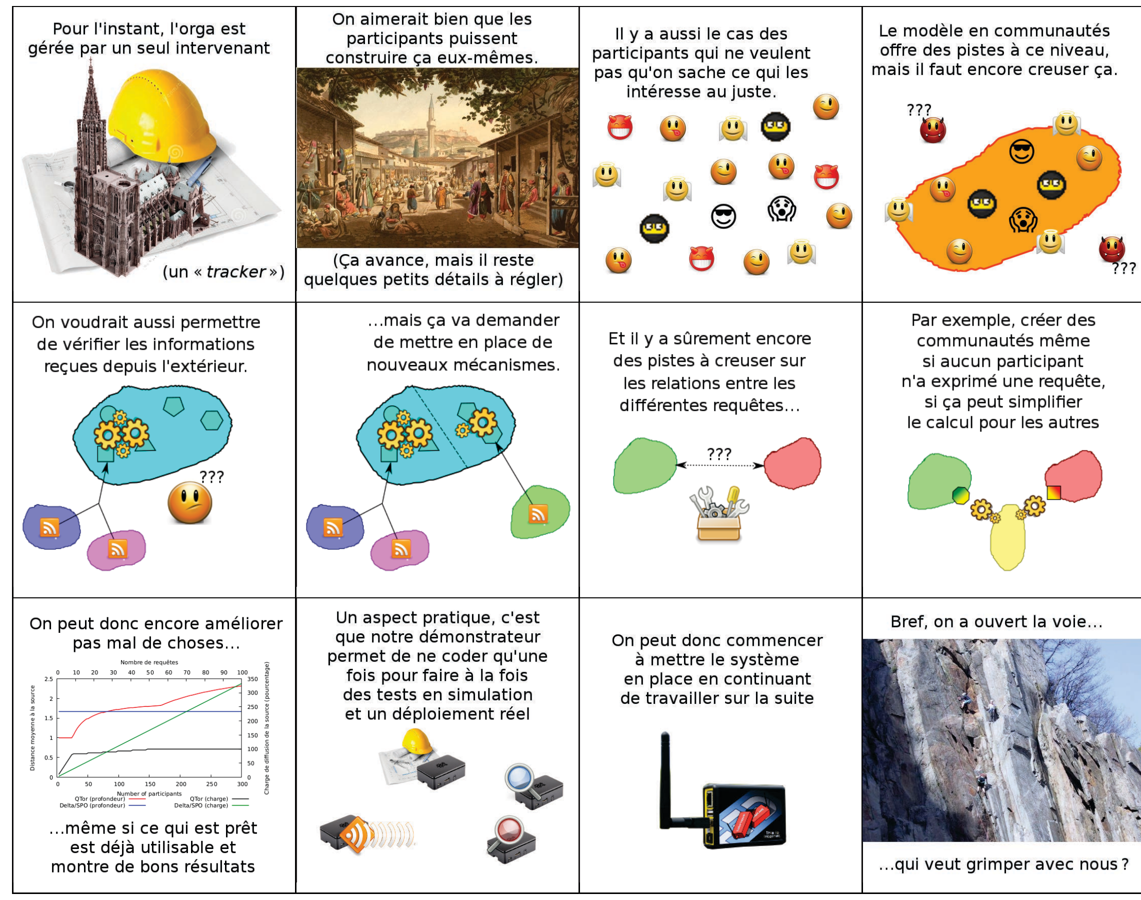
Ça donne un système très distribué, où chacun ne travaille que pour ses besoins à soi.

Si jamais un des participants est très puissant et altruiste, rien ne l'empêche de venir donner des coups de main là où il y en a besoin.

…et le système fonctionne « de chacun, selon ses moyens, à chacun, selon ses besoins. »

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Nous avons donc mis en place quelque chose de très intéressant. Un prototype a été conçu pour faire tourner ça réellement, et pour l'instant, tout va bien... mais il reste encore des choses à faire.



Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

Références bibliographiques

[1] Spotify starts shutting down its massive p2p network, 2014. URL: https://torrentfreak.com/spotify-starts-shutting-down-its-massive-p2p-network-140416/.

[2] Daniel J. Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Çetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag Maskey, Alex Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, and Stanley B. Zdonik. The design of the borealis stream processing engine. In *CIDR*, pages 277–289, 2005.

[3] Serge Abiteboul and Oliver M. Duschka. Complexity of answering queries using materialized views. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '98, pages 254–263, New York, NY, USA, 1998. ACM.

[4] Lada A Adamic and Bernardo A Huberman. Zipf's law and the internet. *Glottometrics*, 3(1):143–150, 2002.

[5] Yanif Ahmad and Uğur Çetintemel. Network-aware query processing for stream-based applications. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pages 456–467. VLDB Endowment, 2004.

[6] E. Anceaume, M. Gradinariu, A.K. Datta, G. Simon, and A. Virgillito. A semantic overlay for self-* peer-to-peer publish/subscribe. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, pages 22–22, 2006.

[7] D.P. Anderson. BOINC: a system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 4 – 10, nov. 2004.

[8] Arvind Arasu, Shivnath Babu, and Jennifer Widom. Cql: A language for continuous queries over streams and relations. In Georg Lausen and Dan Suciu, editors, *Database Programming Languages*, volume 2921 of *Lecture Notes in Computer Science*, pages 1–19. Springer Berlin Heidelberg, 2004.

[9] Demonstration Proposal Arvind, Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Keith Ito, Itaru Nishizawa, Justin Rosenstein, and Jennifer Widom. STREAM: The stanford stream data manager. *IEEE Data Engineering Bulletin*, 26:2003, 2003.

[10] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16. ACM, 2002.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

[11] Scott Boag, Don Chamberlin, Mary F Fernández, Daniela Florescu, Jonathan Robie, Jérôme Siméon, and Mugur Stefanescu. Xquery 1.0: An xml query language, 2002.

[12] Stéphane Bortzmeyer. Centralisé, décentralisé, pair à pair, quels mots pour l'architecture des systèmes répartis ? *JRES*, 2015.

[13] Bengt Carlsson and Rune Gustavsson. The rise and fall of napster–an evolutionary approach.

[14] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. Splitstream: High-bandwidth multicast in cooperative. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. ACM Request Permissions, December 2003.

[15] Stefano Ceri, Georg Gottlob, and Letizia Tanca. What you always wanted to know about datalog (and never dared to ask). *Knowledge and Data Engineering, IEEE Transactions on*, 1(1):146–166, 1989.

[16] Raphaël Chand and Pascal Felber. Semantic peer-to-peer overlays for publish/subscribe networks. In JoséC. Cunha and PedroD. Medeiros, editors, *Euro-Par 2005 Parallel Processing*, volume 3648 of *Lecture Notes in Computer Science*, pages 1194–1204. Springer Berlin Heidelberg, 2005.

[17] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel Madden, Vijayshankar Raman, Frederick Reiss, and Mehul A. Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[18] Jianjun Chen, David J. DeWitt, Feng Tian, and Yuan Wang. NiagaraCQ: a scalable continuous query system for internet databases. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, SIGMOD '00, pages 379–390, New York, NY, USA, 2000. ACM.

[19] Mitch Cherniack, Hari Balakrishnan, Magdalena Balazinska, Donald Carney, Ugur Çetintemel, Ying Xing, and Stanley B. Zdonik. Scalable distributed stream processing. In *CIDR*, 2003.

[20] Paul-Alexandru Chirita, Stratos Idreos, Manolis Koubarakis, and Wolfgang Nejdl. Publish/subscribe for rdf-based p2p networks. In ChristophJ. Bussler, John Davies, Dieter Fensel, and Rudi Studer, editors, *The Semantic Web: Research and Applications*, volume 3053 of *Lecture Notes in Computer Science*, pages 182–197. Springer Berlin Heidelberg, 2004.

[21] Rada Chirkova, Chen Li, and Jia Li. Answering queries using materialized views with minimum size. *VLDB J*, 15(3):191–210, 2006.

[22] Gregory Chockler, Roie Melamed, Yoav Tock, and Roman Vitenberg. Constructing scalable overlays for pub-sub with many topics. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 109–118. ACM, 2007.

[23] James Roberts Christine Fricker, Philippe Robert and Nada Sbihi. Impact of traffic mix on caching performance in a content-centric network. February 2012.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

[24] D. Clark. The design philosophy of the darpa internet protocols. In *Symposium proceedings on Communications architectures and protocols*, SIGCOMM '88, pages 106–114, New York, NY, USA, 1988. ACM.

[25] James Clark, Steve DeRose, et al. Xml path language (xpath) version 1.0, 1999.

[26] Bram Cohen. Incentives build robustness in bittorrent. In *Proceedings of the Workshop on Economics of Peer-to-Peer Systems*, 2003.

[27] PhD Comics. What is openaccess? *URL: http://www.phdcomics.com/comics.php?f=1533*, 2012.

[28] Thomas H Cormen, Charles E Leiserson, and Ronald L Rivest. Algorithms, 1990.

[29] Arturo Crespo and Hector Garcia-Molina. Semantic overlay networks for P2P systems. In *AP2PC*. Springer, 2004.

[30] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[31] S.E. Deering. Host extensions for IP multicasting, RFC1112, August 1989.

[32] Manal El Dick. *P2P Infrastructure for Content Distribution*. PhD thesis, January 2009.

[33] Sébastien Dufromentel, Sylvie Cazalens, François Lesueur, and Philippe Lamarre. Qtor: A flexible publish/subscribe peer-to-peer organization based on query rewriting. In *Database and Expert Systems Applications*, pages 507–519. Springer, 2015.

[34] Sébastien Dufromentel, Sylvie Cazalens, and François Lesueur Philippe Lamarre. FleDDI: Highlighting the Flexibility of Data Dissemination Systems. 2016. Bases de Données Avancées.

[35] Sébastien Dufromentel, François Lesueur, and Philippe Lamarre. QTor : une organisation basée sur les requêtes pour les systèmes distribués de gestion de flux, October 2013. Bases de Données Avancées.

[36] Antoine Dutot, Frédéric Guinand, Damien Olivier, and Yoann Pigné. Graphstream: A tool for bridging the gap between complex systems and dynamic graphs. In *Emergent Properties in Natural and Artificial Complex Systems. Satellite Conference within the 4th European Conference on Complex Systems (ECCS'2007)*, 2007.

[37] Peter Eades, Xuemin Lin, and W.F. Smyth. A fast and effective heuristic for the feedback arc set problem. *Information Processing Letters*, 47(6):319 – 323, 1993.

[38] Manal El Dick, Esther Pacitti, and Bettina Kemme. Flower-CDN: a hybrid P2P overlay for efficient query processing in CDN. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '09, pages 427–438, New York, NY, USA, 2009. ACM.

[39] Igor Epimakhov, Abdelkader Hameurlain, Tharam Dillon, and Franck Morvan. Resource scheduling methods for query optimization in data grid systems. In Johann Eder, Maria Bielikova, and AMin Tjoa, editors, *Advances in Databases and Information Systems*, volume 6909 of *Lecture Notes in Computer Science*, pages 185–199. Springer Berlin Heidelberg, 2011.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

[40] Jenny Rose Finkel and Christopher D. Manning. Nested named entity recognition. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1*, EMNLP '09, pages 141–150, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[41] Bryan Ford, Pyda Srisuresh, and Dan Kegel. Peer-to-peer communication across network address translators. In *USENIX Annual Technical Conference, General Track*, pages 179–192, 2005.

[42] Framasoft. Dégooglisons Internet. *URL: http://degooglisons-internet.org/*, 2012.

[43] Bugra Gedik, Henrique Andrade, Kun-Lung Wu, Philip S. Yu, and Myungcheol Doo. SPADE: the system s declarative stream processing engine. In Jason Tsong-Li Wang, editor, *SIGMOD Conference*, pages 1123–1134. ACM, 2008.

[44] A. Gounaris, N.W. Paton, R. Sakellariou, A.A.A. Fernandes, J. Smith, and P. Watson. Practical adaptation to changing resources in grid query processing. In *Data Engineering, 2006. ICDE '06. Proceedings of the 22nd International Conference on*, pages 165–165, 2006.

[45] James R Groff and PN Weinberg. The complete reference sql, osborne, 1999.

[46] Abhishek Gupta, Ozgur D. Sahin, Divyakant Agrawal, and Amr El Abbadi. Meghdoot: Content-based publish/subscribe over p2p networks. In *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '04, pages 254–273, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[47] Alon Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, December 2001.

[48] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[49] Zeinab Hmedeh, Nicolas Travers, Nelly Vouzoukidou, Vassilis Christophides, Cédric Du Mouza, and Michel Scholl. Everything you would like to know about RSS feeds and you are afraid to ask. In *BDA'11, Base de Données Avancées*, Maroc, October 2011.

[50] Mojtaba Hosseini, Dewan Tanvir Ahmed, Shervin Shirmohammadi, and Nicolas D Georganas. A survey of application-layer multicast protocols. *IEEE Communications Surveys & Tutorials*, 9(3):58–74, 2007.

[51] François Huguet. Yacy, la recherche web décentralisée. 2011.

[52] Marie Jacob and Zachary Ives. Sharing work in keyword search over databases. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, SIGMOD '11, pages 577–588, New York, NY, USA, 2011. ACM.

[53] Konstantinos Karanasos, Asterios Katsifodimos, and Ioana Manolescu. Delta: Scalable data dissemination under capacity constraints. *PVLDB*, 7(4):217–228, 2013.

[54] Roland Kotto-Kombi, Nicolas Lumineau, Philippe Lamarre, and Yves Caniou. Parallel and Distributed Stream Processing: Systems Classification and Specific Issues. working paper or preprint, October 2015.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

[55] Gunnar Kreitz and Fredrik Niemelä. Spotify–large scale, low latency, p2p music-on-demand streaming. In *Peer-to-Peer Computing*, pages 1–10, 2010.

[56] Yonne Lautre. La brique internet : nouvelle offensive des fai indépendants ? 2015.

[57] François Lesueur. Sécurité des réseaux pair-à-pair. November 2009.

[58] Alon Levy, Anand Rajaraman, and Joann Ordille. Querying heterogeneous information sources using source descriptions. 1996.

[59] A.Y. Levy, A.O. Mendelzon, and Y. Sagiv. Answering queries using views. In *Proceedings of the fourteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 95–104. ACM, 1995.

[60] Jian Liang, Rakesh Kumar, and Keith W Ross. The fasttrack overlay: A measurement study. *Computer Networks*, 50(6):842–858, 2006.

[61] Imene Mami, Zohra Bellahsene, and Remi Coletta. A constraint satisfaction based approach to view selection in a distributed context. In *BDA'2012: 28e journées Bases de Données Avancées*, page 10, 2012.

[62] I. Manolescu, K. Karanasos, V. Vassalos, and S. Zoupanos. Efficient xquery rewriting using multiple views. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 972–983, 2011.

[63] Andreu Mas-Colell, Michael Dennis Whinston, Jerry R Green, et al. *Microeconomic theory*, volume 1. Oxford university press New York, 1995.

[64] Iris Miliaraki and Manolis Koubarakis. Foxtrot: Distributed structural and value xml filtering. *ACM Trans. Web*, 6(3):12:1–12:34, October 2012.

[65] Marcelo A Montemurro. Beyond the zipf–mandelbrot law in quantitative linguistics. *Physica A: Statistical Mechanics and its Applications*, 300(3):567–578, 2001.

[66] Alberto Montresor and Márk Jelasity. PeerSim: A scalable P2P simulator. In *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, pages 99–100, Seattle, WA, September 2009.

[67] Matteo Mordacchini, Ranieri Baraglia, Patrizio Dazzi, and Laura Ricci. A p2p recommender system based on gossip overlays (prego). In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 83–90. IEEE, 2010.

[68] Meredith Ringel Morris. Collaborative search revisited. In *Proceedings of the 2013 conference on Computer supported cooperative work*, CSCW '13, pages 1181–1192, New York, NY, USA, 2013. ACM.

[69] Kamesh Munagala, Utkarsh Srivastava, and Jennifer Widom. Optimization of continuous queries with shared expensive filters. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '07, pages 215–224, New York, NY, USA, 2007. ACM.

[70] Leonardo Neumeyer, Bruce Robbins, Anish Nair, and Anand Kesari. S4: Distributed stream computing platform. In Wei Fan, Wynne Hsu, Geoffrey I. Webb, Bing Liu 0001, Chengqi Zhang, Dimitrios Gunopulos, and Xindong Wu, editors, *ICDM Workshops*, pages 170–177. IEEE Computer Society, 2010.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

[71] Jakob Nielsen. Nielsen's law of internet bandwidth. *Online at http://www.useit.com/alertbox/980405.html*, 1998.

[72] Brice Nédelec, Pascal Molli, and Achour Mostéfaoui. CRATE: Writing stories together with our browsers. World Wide Web Conference, 2016.

[73] Melih Onus and Andréa W Richa. Minimum maximum-degree publish-subscribe overlay network design. *IEEE/ACM Transactions on Networking (TON)*, 19(5):1331–1343, 2011.

[74] Gurobi Optimization et al. Gurobi optimizer reference manual. *URL: http://www. gurobi. com*, 2012.

[75] O. Papaemmanouil and U. Cetintemel. Semcast: Semantic multicast for content-based data dissemination. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pages 242–253, 2005.

[76] Jay A Patel, Étienne Rivière, Indranil Gupta, and Anne-Marie Kermarrec. Rappel: Exploiting interest and network locality to improve fairness in publish-subscribe systems. *Computer Networks*, 53(13):2304–2320, 2009.

[77] Loïc Petit, Cyril Labbé, and Claudia Lucia Roncancio. An algebric window model for data stream management. In *Proceedings of the Ninth ACM International Workshop on Data Engineering for Wireless and Mobile Access*, MobiDE '10, pages 17–24, New York, NY, USA, 2010. ACM.

[78] Guillaume Pierre and Maarten Van Steen. Globule: a collaborative content delivery network. *IEEE Communications Magazine*, 44(8):127–133, 2006.

[79] Peter R. Pietzuch, Jonathan Ledlie, Jeffrey Shneidman, Mema Roussopoulos, Matt Welsh, and Margo I. Seltzer. Network-aware operator placement for stream-processing systems. In Ling Liu, Andreas Reuter, Kyu-Young Whang, and Jianjun Zhang, editors, *ICDE*, page 49. IEEE Computer Society, 2006.

[80] Rachel Pottinger and Alon Halevy. Minicon: A scalable algorithm for answering queries using views. *The VLDB Journal*, 10(2-3):182–198, September 2001.

[81] Eric Prud'Hommeaux, Andy Seaborne, et al. Sparql query language for rdf. *W3C recommendation*, 15, 2008.

[82] Konstantin Pussep, Matthias Weinert, Nicolas Liebau, and Ralf Steinmetz. Flexible framework for nat traversal in peer-to-peer applications. Technical Report KOM-TR-2007-06, KOM - Multimedia Communications Lab, Technische Universität Darmstadt, Merckstraße 25, 64283 Darmstadt, Germany, Nov 2007.

[83] Antony I. T. Rowstron and Peter Druschel. Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350. Springer, 2001.

[84] Stefan Saroiu, Krishna P Gummadi, Richard J Dunn, Steven D Gribble, and Henry M Levy. An analysis of internet content delivery systems.

[85] Michel Serres. *Petite poucette.* Le pommier Paris, France, 2012.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

[86] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE, 2010.

[87] Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, volume 31, 4 of *Computer Communication Review*, pages 149–160, San Diego, California, USA, 2001. ACM Press.

[88] Iradj Ouveysib Tolga Bektasa, Osman Oguza. Designing cost-effective content distribution networks. In *Computers and Operations Research*, October 2005.

[89] Jordi Creus Tomàs, Bernd Amann, Nicolas Travers, and Dan Vodislav. RoSeS: a continuous content-based query engine for RSS feeds. In *DEXA'11: Proceedings of the 22nd international conference on Database and expert systems applications*. Springer-Verlag, August 2011.

[90] Christos Tryfonopoulos, Stratos Idreos, Manolis Koubarakis, and Paraskevi Raftopoulou. Distributed large-scale information filtering. In Abdelkader Hameurlain, Josef Küng, and Roland Wagner, editors, *Transactions on Large-Scale Data- and Knowledge-Centered Systems XIII*, volume 8420 of *Lecture Notes in Computer Science*, pages 91–122. Springer Berlin Heidelberg, 2014.

[91] Rob V. van Nieuwpoort, Jason Maassen, Rutger Hofman, Thilo Kielmann, and Henri E. Bal. Ibis: an efficient Java-based grid programming environment. In *Joint ACM Java Grande - ISCOPE 2002 Conference*, pages 18–27, Seattle, Washington, USA, November 2002.

[92] Joel Wolf, Nikhil Bansal, Kirsten Hildrum, Sujay Parekh, Deepak Rajan, Rohit Wagle, Kun-Lung Wu, and Lisa Fleischer. SODA: an optimizing scheduler for large-scale stream-based distributed computer systems. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '08, pages 306–325, New York, NY, USA, 2008. Springer-Verlag New York, Inc.

[93] FFDN / YunoHost. La Brique Internet. *URL: https://labriqueinter.net*, 2015.

[94] Yuqing Zhu, Jianmin Wang, and Chaokun Wang. Ripple: A publish/subscribe service for multidata item updates propagation in the cloud. *Journal of Network and Computer Applications*, 34(4):1054 – 1067, 2011. Advanced Topics in Cloud Computing.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés

FOLIO ADMINISTRATIF

THESE DE L'UNIVERSITE DE LYON OPEREE AU SEIN DE L'INSA LYON

Nom : FOUGERIT
Nom d'usage : DUFROMENTEL

Prénoms : Sébastien, Julien, Yvan

DATE de SOUTENANCE : 9 décembre 2017

TITRE : QTor : une approche communautaire pour l'évaluation de requêtes continues

NATURE : Doctorat

Numéro d'ordre : 2016LYSEI132

Ecole doctorale : Infomaths

Spécialité :

RESUME :

Cette thèse porte sur la mise en place d'un système de requêtage sur des flux sous contraintes de capacités. Ce système est porté par ses utilisateurs·trices et basé sur les similitudes entre requêtes.

Les relations d'équivalences entre les différentes requêtes permettent de réunir les participants au sein de communautés d'intérêt. Celles-ci forment alors une abstraction permettant de séparer le problème d'organisation du système en plusieurs sous-problèmes plus simples et de taille réduite.

Afin de garantir une généricité vis-à-vis du langage, l'organisation repose sur une API simple et modulable. Nous avons ainsi recours au mécanisme de réécritures de requêtes utilisant des vues matérialisées, connu en bases de données, pour déterminer les relations possibles entre les communautés. Le choix entre ces différentes possibilités est ensuite effectué à l'aide d'un modèle de coût paramétrable.

Les relations entre communautés sont concrétisées par un échange de ressources entre elles, un participant de l'une venant contribuer à l'autre. Cela permet de s'affranchir des limitations de capacités au niveau abstrait, tout en en tenant hautement compte pour la mise en relation effective des participants.

Au sein des communautés, un arbre de diffusion permet à l'ensemble des participants de récupérer les résultats requis.

L'approche, mise en œuvre de manière incrémentale, permet une réduction efficace des coûts de calcul et de diffusion (l'optimalité est atteinte, notamment, dans le cas de l'inclusion de requête) pour un coût d'organisation limité et une latence raisonnable. Les expérimentations réalisées ont montré une grande adaptabilité aux variations concernant les requêtes exprimées et les capacités des participants.

Le démonstrateur mis en place peut être utilisé à la fois pour des simulations (automatiques ou interactives) et pour un déploiement réel, par une implémentation commune générique vis-à-vis du langage.

MOTS-CLÉS : requêtes continues sur flux, communautés d'intérêts, système pair-à-pair, réécritures de requêtes

Laboratoire (s) de recherche : LIRIS

Directeur de thèse : Philippe Lamarre
Co-Directeur de thèse : François Lesueur

Président de jury : Composition du jury : HAMEURLIN AbdelKader, DEFUDE Bruno, SKAF Hala, RONCANCIO Claudia,

LAMARRE Philippe, LESUEUR François

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2016LYSEI132/these.pdf
© [S. Dufromentel], [2016], INSA Lyon, tous droits réservés