Optimisation de requêtes en environnements multi-clouds

Damien T Wojtowicz

THÈSE

En vue de l'obtention du
DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Université Toulouse 3 - Paul Sabatier

Présentée et soutenue par

Damien WOJTOWICZ

Le 26 avril 2023

Optimisation de requêtes en environnements multi-clouds

Ecole doctorale : **EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse**

Spécialité : **Informatique et Télécommunications**

Unité de recherche :
IRIT : Institut de Recherche en Informatique de Toulouse

Thèse dirigée par
Franck MORVAN et Abdelkader HAMEURLAIN

Jury

Mme Marinette SAVONNET, Rapporteure
M. Djamal BENSLIMANE, Rapporteur
M. Philippe PUCHERAL, Examinateur
M. Franck MORVAN, Directeur de thèse
M. Abdelkader HAMEURLAIN, Co-directeur de thèse

OPTIMISATION DE REQUÊTES EN ENVIRONNEMENTS MULTI-CLOUDS

DAMIEN WOJTOWICZ

Manuscrit de thèse

Université Toulouse III – Paul Sabatier
Institut de Recherche en Informatique de Toulouse

Directeur de thèse : *Pr Franck MORVAN*
Co-directeur : *Pr Abdelkader HAMEURLAIN*
Co-encadrante : *Dr Shaoyi YIN*

Remerciements

Je tiens à adresser toute ma gratitude à Franck MORVAN, Abdelkader HAMEURLAIN et Shaoyi YIN pour leur confiance, leurs conseils, leur support et leur bienveillance, qu'ils me témoignent depuis plusieurs années déjà. C'est grâce à eux que j'ai pu mener à bien le doctorat et grandir scientifiquement. Je leur en suis très reconnaissant.

Je remercie ensuite les membres du jury, Djamal BENSLIMANE, Phillipe PUCHERAL et Marinette SAVONNET, pour le temps qu'ils ont passé à évaluer mes travaux, leur bienveillance et leurs commentaires constructifs.

Merci également à Kaiyin, qui m'a soutenu pendant cette thèse et sans qui je ne serai certainement pas arrivé jusqu'au bout. Je remercie aussi mes parents pour leur soutien pendant mes études et pour leur éducation qui m'a donné la curiosité d'apprendre. Merci également à tous mes proches pour leur présence.

Enfin, je remercie les collègues qui partagent mon quotidien au laboratoire pour leur présence, les bons moments que nous passons ensemble, pour leur inspiration, pour leur implication dans les différents projets que nous avons entrepris à la commission des doctorants ainsi que leur aide pour les enseignements. Merci donc à Riad, à Malik, à Antoine, à Mira, à Max, à Alexis, à Lila, à Olivier, à Maël, à Rafik, à Luis, à Paul, à Nicolas, à Raphaël, à Gilles, à Karen, à José, à Hugo et à bien d'autres.

Résumé

La massification des données publiques pousse leurs producteurs à sous-traiter leur diffusion auprès de fournisseurs cloud, parfois sous la forme de bases de données relationnelles hébergées sous des offres de type *Database-as-a-Service* (DBaaS). L'étude de ces jeux de données peut passer par leur analyse croisée, qui peut être effectuée à l'aide de requêtes multi-clouds lorsque les relations sur lesquelles elles portent sont hébergées par des fournisseurs cloud différents. Dans cette perspective, un middleware gérant l'orchestration de la sous-traitance des requêtes multi-clouds auprès des fournisseurs qu'elles impliquent a été proposé. Il calcule des devis pour ses utilisateurs, afin de les informer des performances et du coût monétaire de leurs requêtes. Ces devis sont dérivés de plans d'exécution multi-clouds, produits par un optimiseur s'appuyant sur estimations sur les résultats intermédiaires des sous-requêtes.

Deux stratégies de recherche ont été proposées. La première, exhaustive, permet de trouver de bons plans d'exécution pour des requêtes impliquant peu de fournisseurs. Cependant, sa complexité factorielle a conduit au développement d'une seconde stratégie, aléatoire et itérative. Celle-ci est conçue pour explorer plus rapidement une grande variété de plans d'exécution tout en ne produisant pas d'explosion combinatoire. Les estimations peuvent être erronées, diminuant ainsi la précision des devis et conduisant à l'exécution de plans sous-optimaux. Afin de protéger le middleware et ses utilisateurs des conséquences de ces erreurs, un modèle de coûts multi-cloud et une méthode d'optimisation dynamique ont été proposés. Le premier corrige les estimations fournies à l'optimiseur à l'aide de modèles d'apprentissage automatique en ligne. La seconde réoptimise les plans d'exécution multi-clouds à l'aune des valeurs constatées sur les résultats intermédiaires grâce à un système multi-agent.

Le volet expérimental de cette thèse a montré que la sous-traitance des requêtes multi-clouds étaient financièrement pertinentes par rapport à un téléchargement des données qu'elles manipulent suivi d'une exécution chez un seul fournisseur cloud. Les apports de chacune des stratégies de recherche, du modèle de coûts multi-cloud et de la méthode d'optimisation dynamique, ont également été évalués.

Mots-clés : Multi-cloud, Optimisation dynamique de requêtes, Stratégie de recherche, Modèles de coûts, Apprentissage automatique en ligne.

Abstract

Query Optimisation within Multi-cloud Environments

The increasing volume of public data is leading their producers to outsource their dissemination to cloud providers, sometimes as relational databases hosted on Database-as-a-Service (DBaaS) offers. Studying these datasets may involve cross-analysis which can be achieved using multi-cloud queries if the source relations are hosted by several cloud providers. With this in mind, a middleware managing the orchestration of the outsourcing of the multi-cloud queries to the providers they involve was proposed. It calculates quotations for its users, in order to inform them of the performance and monetary cost of their queries, on the basis of multi-cloud execution plans. The latter are produced by an optimiser taking as an input estimates on the sub-queries' intermediate results.

Two search strategies was proposed. The first one is exhaustive and can find good execution plans for queries involving few providers. However, its factorial complexity led to the development of a second strategy that is random and iterative. The latter is designed to explore more quickly a wide variety of execution plans while not entailing a combinatorial explosion. The estimates can be erroneous, hence reducing the accuracy of the quotations and ultimately leading to the execution of sub-optimal plans. In order to protect both the middleware and its users from the consequences of these errors, a multi-cloud cost model and a dynamic optimisation method was proposed. The former corrects the estimates provided to the optimiser using online machine learning models. The latter re-optimises the multi-cloud execution plans according to the actual intermediate results using a multi-agent system.

Experiments during this thesis showed that multi-cloud queries were financially efficient when compared to downloading the data then need and executing them on a single cloud provider. The benefits of each of the search strategies, the multi-cloud cost model and the dynamic optimisation method were also evaluated.

Keywords: Multi-cloud, Dynamic Query Optimisation, Search Strategy, Cost Model, Online Machine Learning

Table des matières

Table des illustrations

Liste des figures

Liste des tableaux

Liste des algorithmes

1

Introduction générale

1 *L'ère du* « Big Data » *et l'avènement du multi-cloud*

La production et l'accumulation de quantités de plus en plus massives de données dans tous les domaines par une grande variété d'acteurs est un phénomène qui va en s'accélérant. Les exemples sont pléthoriques [1] : les réseaux sociaux gagnent quotidiennement de nouveaux utilisateurs dont le comportement est systématiquement archivé afin d'être méticuleusement scruté ; l'instrumentation jouit d'une précision, d'une résolution et d'une couverture sans précédente [2] ; le génome d'un nombre croissant d'individus et d'espèces est séquencé quotidiennement. Ces données massives, qualifiées de « *Big Data* », sont caractérisées par les « quatre V » évoquant à la fois leurs caractéristiques intrinsèques, c'est-à-dire leur volume et leur variété, ainsi que les implications de ces dernières sur la conception des systèmes les exploitant en matière de vélocité et de véracité [3, ch. 10, p. 449].

Elles peuvent devenir encombrantes, au point que leur stockage et leur analyse dépassent parfois les capacités physiques et techniques de leurs producteurs. Dès lors, la mise à l'échelle des infrastructures informatiques les accueillant est nécessaire, impliquant le passage de paliers techniques (p. ex. passer d'une machine mono-processeur à une machine parallèle) et engendrant des coûts d'investissement, humains et de maintenance potentiellement importants. Dans le cas d'organisations mettant à disposition du public leurs jeux de données, la maintenance de telles infrastructures peut dépasser leurs capacités au point qu'elles ne puissent ou ne souhaitent plus les gérer seules. Elles ont donc recours au *cloud computing*.

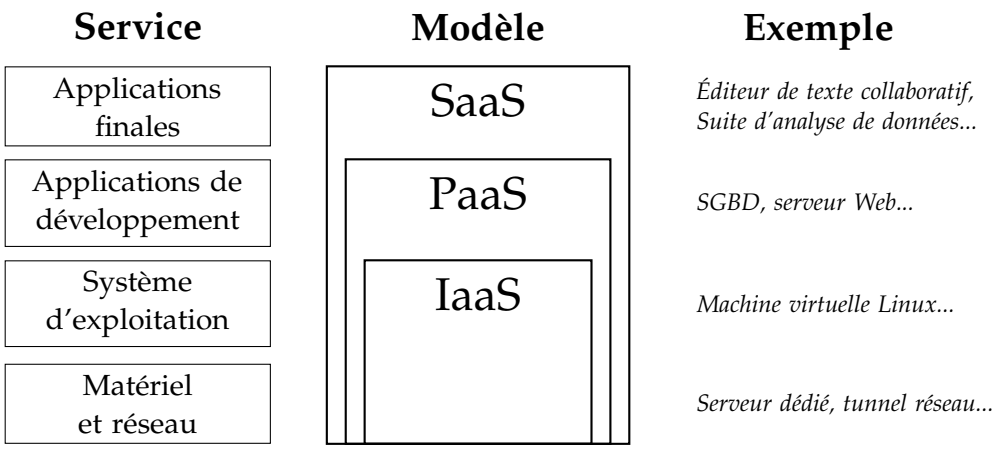


FIGURE 1.1 – Les différents modèles de service cloud.

1.1 *Le* cloud computing

En se réappropriant la notion d'informatique utilitaire, dont l'idée remonte aux années 1960 [4], des entreprises du numérique ont fait émerger un marché pour des services informatiques distants dans les années 1990 [5] dont l'existence est la conséquence conjointe des évolutions de la technique, essentiellement grâce à l'avènement de l'Internet généralisé et des grilles de calcul, et d'un terreau idéologique favorable dans les milieux économiques suivant la doctrine de l'externalisation des activités ne relevant pas du cœur de compétence des organisations [6]. Ce marché est celui du cloud, désormais discrètement omniprésent, dont l'utilisation par de multiples acteurs a explosé ces dernières années, bouleversant la conception des systèmes informatiques à un point tel qu'il engendre des enjeux géo-stratégique [7, 8].

Plusieurs fournisseurs cloud co-existent, proposant à leurs locataires (c-à-d leurs clients), des services se positionnant à différents niveaux d'abstraction de la technique mais promouvant tous la même idée : celle de la mise à disposition contractualisée de ressources virtuellement infinies permettant un passage à l'échelle transparent et flexible tout en restant économiquement compétitifs grâce à l'élasticité [1] et aux économies d'échelle.

1.1.1 *Types de service cloud*

D'un point de vue technique, trois modèles de service cloud existent [10, 11], se situant sur un continuum d'abstraction du matériel, des logiciels et de la configuration de ces derniers pour les locataires (cf. fig. 1.1) ; d'un point de

1. Ce mot étant polysémique, il convient donc de préciser que ses définitions physiques, économiques et informatiques sont différentes, bien que les deux dernières évoquent métaphoriquement la première. Dans le cadre du cloud, il désigne la capacité d'un système à adapter automatiquement sa capacité pour qu'elle soit, en tout temps, au plus près de la demande [9].

vue économique, les fournisseurs appliquent ces modèles à des politiques de ventes en commerce de gros (quand les clients sont d'autres organisations) ou de détail [2] (quand les clients sont des consommateurs) [12]. Dans tout les cas, le modèle de facturation du fournisseur est de type *pay-per-use*, c'est-à-dire que les locataires paient selon leur utilisation des infrastructures du fournisseur.

Dans le modèle *Infrastructure-as-a-Service* (IaaS), faisant généralement l'objet d'un commerce de gros, les ressources que le fournisseur met à disposition de ses locataires sont de bas niveau dans la pile logicielle. Elles sont une abstraction des machines physiques, similaires aux plus classiques services d'hébergement sur serveurs, permettent de déployer des machines virtuelles ou des conteneurs. Le locataire aura à sa charge la configuration de ces dernières.

Dans le modèle *Platform-as-a-Service* (PaaS), faisant aussi l'objet d'un commerce de gros, le locataire loue des ressources de développement préconfigurées qu'il peut utiliser comme plate-forme pour faire fonctionner d'autres applications. On peut citer comme exemple de ressources PaaS un serveur Web où le locataire pourra y déployer son site Internet, ou encore un Système de Gestion de Base de Données (SGBD) où le locataire pourra stocker et interroger une base de données.

Dans le modèle *Software-as-a-Service* (SaaS), faisant à la fois l'objet d'un commerce de gros et de détail, le locataire manipule une application complète fournie par le fournisseur cloud. Il n'a alors accès qu'aux fonctionnalités de celle-ci, et ne peut décider du déploiement et des plate-formes sous-jacentes à l'application. Des exemples de tels services sont un système de messagerie ou encore un logiciel d'édition collaborative.

Ces modèles étant génériques, leur application à un service spécifique peut donner naissance à une appellation à la construction analogue. Le *Data-as-a-Service* (DaaS) désigne ainsi des services niveau IaaS stockant des fichiers ou encore le *Database-as-a-Service* (DBaaS) [13], de niveau PaaS, qui désigne une offre encapsulant un SGBD *multi-tenant*, c'est-à-dire un SGBD utilisé par plusieurs locataires.

1.1.2 *Les accords de niveau de service*

La contractualisation des relations entre les fournisseurs cloud et les locataires se fait au moyen de *Service-Level Agreement* (SLA), des accords de niveau de service. S'ils sont de nature variées [14] ils définissent systématiquement une méthode de facturation, parfois un *Service-Level Objective* (SLO) spécifiant des objectifs de performances à atteindre par le fournisseur et d'éventuelles pénalités sanctionnant le fournisseur s'il ne respecte pas ses engagements contractuels. Dans le cadre du DBaaS, diverses méthodes de

2. Respectivement « *wholesale* » et « *relail* » dans la littérature en anglais.

facturation sont envisageables. Un fournisseur peut choisir de faire payer les opérations effectuées par les locataires sur leurs données sous forme de forfaits, proportionnellement aux ressources de calcul qu'ils utilisent ou à la quantité de données qu'ils stockent et interrogent, voire selon un prix résultant d'une négociation avec ses locataires [15].

Les SLO peuvent aussi bien définir des engagements généraux de disponibilité des services de la part du fournisseur (il doit par exemple garantir que son service DBaaS soit disponible 98 % du temps) à des engagements très spécifiques concernant les performances d'une opération particulière (il s'engage par exemple à exécuter une requête donnée en moins de 42 ms). Ils peuvent aussi être génériques ou résultant d'une négociation. La variété des pénalités envisageables est à l'image de celle des méthodes de facturation et des SLO ; elles vont de l'avoir au remboursement, et peuvent aussi être sujettes à une négociation.

1.2 *Des données publiques dans le cloud*

Cette grande diversité dans les services et les modalités disponibles pousse divers producteurs de données publiques à distribuer leurs jeux de données sur des plateformes cloud de type DaaS et DBaaS, comme l'Agence Spatiale Européenne (ESA) dans le cas des données du programme Sentinel-2 [16], l'agence états-unienne *National Oceanic and Atmospheric Administration* (NOAA) [17] dans le cadre de sa politique générale de publication de données ou encore divers consortiums de recherche en médecine et en biologie [18]. Cette mise à disposition de données publiques auprès des fournisseurs cloud a un intérêt pour les deux parties, dans la mesure où les producteurs peuvent diffuser leurs jeux de données à moindre coût et les hébergeurs peuvent espérer augmenter leur chiffre d'affaires en rendant plus attractifs leurs autres services type SaaS pour l'analyse de données.

Des exemples de services proposant un accès clé-en-main à des données publiques sont l'offre DBaaS de Google « BigQuery » [3] permettant d'accéder au catalogue de la plateforme Google Cloud [4], les services AWS d'Amazon [5] ou bien le service de type DBaaS d'Alibaba « MaxCompute » [6] lié au catalogue de données publiques Tianchi [7]. Ces offres DBaaS sont à distinguer des services d'hébergements de bases de données dans le cloud proposés par exemple par OVHcloud [8] ou Naver [9], qui suivent un modèle IaaS dans la méthode de facturation et dans la manière de les administrer. La tendance

3. Voir `https://cloud.google.com/bigquery`
4. Voir `https://cloud.google.com/bigquery/public-data`
5. Voir `https://registry.opendata.aws`
6. Voir `https://www.alibabacloud.com/help/en/maxcompute`
7. Voir `https://tianchi.aliyun.com/dataset`
8. Voir `https://www.ovhcloud.com/fr/public-cloud/databases`
9. Voir `https://www.ncloud.com/product/database`

à la sous-traitance auprès de fournisseurs cloud de la diffusion de données publiques semble globale, appelant ainsi à la conception de systèmes permettant de croiser les données issues de ces différents clouds, à intégrer les données à un niveau multi-cloud.

1.3 *Du cloud au multi-cloud*

Certains fournisseurs cloud, à l'image de Google avec son service Omni [10] ou encore Oracle avec OCI [11], proposent désormais des services d'analyse de données multi-clouds [19, 20]. L'intention derrière la conception de telles offres est avant tout commerciale : il s'agit pour les fournisseurs d'attirer des nouveaux clients en proposant un démonstrateur de leur plate-forme *Analytics-as-a-Service* (de niveau SaaS) [21] tout en proposant des outils facilitant leur migration depuis les services concurrents. La possibilité de les utiliser pour interroger des bases de données issues de différents fournisseurs reste conditionnée à la volonté du fournisseur d'adapter son système pour supporter d'autres concurrents [12]. Quoi qu'il en soit, la clientèle cible pour ce genre d'offre reste des organisations ayant déjà recours à des services cloud, la possibilité de les utiliser pour analyser des données publiques diffusées dans le cloud relevant plus de la conséquence de l'architecture de ces systèmes que d'une réelle intention derrière leur conception.

Parallèlement aux fournisseurs cloud privés, la recherche sur la recherche sur la gestion de données en environnements multi-clouds s'est elle orientée dans deux axes. Le premier axe s'intéresse à l'utilisation des services de plusieurs fournisseurs cloud comme infrastructure dans la conception de Systèmes de Gestion de Données (SGD) distribués. Ceux-ci ont d'abord été conçus pour la sécurité [22] avant de prendre en compte l'aspect monétaire et les performances. Tous ces systèmes exploitent les plateformes IaaS de plusieurs fournisseurs plutôt que leurs offres DaaS ou DBaaS. Ces systèmes font de la macro-optimisation de leurs performances et de leurs dépenses, en agissant surtout sur la gestion des ressources louées, plutôt que de la micro-optimisation sur chacune des opérations effectuées (p. ex. des requêtes).

Le second axe suivi par la recherche se concentre sur l'optimisation de requêtes en environnements multi-clouds. L'allocation de ressources dans un cluster Hadoop et le placement de tâches sur des clouds publics pour du MapReduce ont été étudiés. Si la plupart des travaux sur le sujet se focalisent sur une optimisation mono-objectif (c-à-d soit en minimisant les dépenses [23-25], soit en maximisant les performances [26]), d'autres proposent plutôt de

10. Voir `https://cloud.google.com/bigquery/docs/omni-introduction`
11. Voir `https://www.oracle.com/fr/cloud/multicloud/`
12. Par exemple, dans l'exemple précédemment cité d'Omni, Google ne permet, pour l'instant, d'interagir qu'avec des données issues de son service BigQuery, du service Azure de Microsoft et d'Amazon Web Services.

La nébuleuse d'Orion (M42) est un endroit particulièrement élégant de l'Univers, où un ensemble de nuages de poussières et de gaz donnent naissance à des étoiles grâce à leurs interactions mutuelles.

Crédits : *NASA, ESA, M. Robberto (Space Telescope Science Institute/ESA) and the Hubble Space Telescope Orion Treasury Project Team*

FIGURE 1.2 – M42, une inspiration astronomique

faire de l'optimisation multicritères [27]. Cette idée se retrouve par ailleurs dans la littérature relative aux modèles de coûts conçus pour les optimiseurs de requêtes relationnelles multi-clouds [28, 29], afin de trouver des plans d'exécution à la fois performants et peu coûteux [30, 31].

2 *Nebula : un middleware pour l'optimisation de requêtes multi-clouds*

À l'instar de ce qu'il est possible avec le MapReduce, l'utilisation de plateformes DBaaS comme ressource de calcul rend envisageable une sous-traitance totale du traitement multi-clouds de requêtes relationnelles auprès des fournisseurs cloud qu'elles impliqueraient, par jeu de sous-requêtes et de transferts de données inter-fournisseurs. Une telle approche appelle à concevoir un middleware permettant de composer et d'orchestrer ces dernières : il s'agit là du sujet de ces travaux de thèse.

Un middleware agissant comme un courtier bénévole a donc été proposé. Il prend en charge les requêtes multi-clouds de ses utilisateurs et leur servant d'intermédiaire avec les fournisseurs cloud. Celui-ci a été nommé « Nebula », mot anglais du champ lexical astronomique désignant une nébuleuse, c'est-à-dire un objet céleste du milieu interstellaire composé de gaz, de plasma et de poussières [13] (cf. fig. 1.2). Lorsqu'observées, leur aspect diffus donne l'impression qu'elles sont en fait de grands nuages dans lesquels naissent les étoiles, métaphore du multi-clouds et des potentiels éclairages apportés par les requêtes multi-clouds.

13. Voir https://fr.wikipedia.org/wiki/Nébuleuse

L'orchestration de la sous-traitance de ces requêtes auprès des fournisseurs qu'elles impliquent se fait grâce à des plans d'exécution multi-clouds produits par le middleware. Nebula doit donc les optimiser, afin d'éviter à ses utilisateurs d'avoir à payer plus que ce qu'ils ne devraient à cause d'un plan d'exécution monétairement inefficace, ou encore qu'ils attendent trop longtemps pour leurs résultats à cause d'un plan d'exécution dont les performances sont améliorables. En fait, le middleware doit résoudre un problème d'optimisation multicritères. Plusieurs plans d'exécution multi-clouds peuvent donc coexister sur un front de Pareto, et chacun d'entre eux peut être un bon candidat pour satisfaire les attentes des utilisateurs selon leur budget et leur patience. Nebula leur laisse donc le choix entre ces plans d'exécution Pareto-optimaux en leur proposant les devis qui leur sont associés.

2.1 *Un optimiseur de requêtes multi-clouds*

S'il a été possible de calculer des devis, c'est que des plans d'exécution multi-clouds ont été construits par le middleware, et donc que des estimations sur le coût monétaire et les performances de leurs composantes ont été faites. Nebula dispose d'un optimiseur de requêtes multi-clouds, qui explore l'espace des plans d'exécution multi-clouds possibles à l'aide d'une stratégie de recherche. Cette tâche étant NP-difficile [32], la stratégie de recherche de l'optimiseur doit nécessairement suivre des heuristiques afin d'épargner aux utilisateurs du middleware des temps d'attentes importants induits par une potentielle explosion combinatoire.

Deux stratégies de recherche ont été proposées pendant cette thèse. Une première, adaptée à des requêtes multi-clouds peu complexes, explore de manière exhaustive l'ensemble des plans d'exécution multi-clouds. La simplicité de la conception de cette stratégie cache cependant sa complexité algorithmique. En effet, le nombre de plans d'exécution qu'elle considère est factoriel en le nombre de fournisseurs impliqués par la requête multi-cloud. Une seconde stratégie permettant le passage à l'échelle vers des requêtes plus complexes a été proposée. Celle-ci est aléatoire et itérative : elle explore de proche en proche le voisinage d'un plan d'exécution initial généré aléatoirement. Son exploration est bornée afin d'éviter qu'elle s'étende à l'ensemble de l'espace de recherche.

Quelque soit la stratégie de recherche employée, elles reposent sur des estimations de cardinalité en sortie, de temps de réponse et de coût monétaire des sous-requêtes. Ces estimations sont faites par le modèle de coûts des fournisseurs cloud. Elles peuvent être inexactes de plusieurs ordres de grandeur [33] car elles sont dérivées d'estimations de sélectivité des opérateurs des sous-requêtes, dont l'estimation reste un problème ouvert [34]. Toute inexactitude peut se traduire à la fois par une dégradation des performances prévues pour la requête multi-cloud et par une explosion de ses

coûts monétaires. Deux solutions ont ainsi été proposées pour limiter ce phénomène.

La première solution proposée est préventive ; elle s'applique pendant l'optimisation. Elle consiste à corriger les estimations provenant des fournisseurs en utilisant l'historique des sous-requêtes, de sorte à améliorer la précision des estimations utilisées par l'optimiseur et *in fine* l'optimalité des plans d'exécution qu'il produit. Cette correction se fait à l'aide de modèles d'apprentissage automatique en ligne, afin que le middleware puisse profiter des dernières avancées en matière de modèles de coûts appris tout en améliorant continuellement la précision de ses ré-estimations.

La seconde solution est curative ; elle consiste à ré-optimiser le plan d'exécution à l'aide des valeurs réelles observées à la fin de l'exécution de chaque sous-requête. Il s'agit d'une méthode d'optimisation dynamique, dont l'objectif est de rapprocher au maximum les performances le coût monétaire final de la requête multi-cloud de ce qui a été présenté à l'utilisateur dans le devis. Elle repose sur un Système Multi-Agent (SMA), dans lequel les agents coopèrent pour corriger le plan d'exécution de manière proactive.

2.2 *Évaluation du middleware*

Le middleware a été évalué en utilisant le *Join-Order Benchmark* (JOB) [33], un ensemble de requêtes SQL formulées sur l'*Internet Movie Database* (IMDb). Les fournisseurs cloud ont été simulés à l'aide de PostgreSQL, assorti d'une API reproduisant le comportement de facturation du service BigQuery de Google. Deux environnements de simulation ont été utilisés : l'un à petite échelle utilisant des ressources disponibles à l'IRIT, l'autre à grande échelle utilisant la grille de calcul nationale Grid'5000 [14] [35].

Les résultats expérimentaux ont démontré la sous-traitance des requêtes multi-clouds est plus rentable pour l'utilisateur que des scénarios de traitement du benchmark par une simulation d'un SGBD multi-cloud issu de la littérature, nécessitant un téléchargement des données suivi d'une exécution chez un seul fournisseur cloud. Ils indiquent également que les devis chiffrés par Nebula ont une bonne précision, en particulier dans leur estimation du coût monétaire de la requête. Cette bonne précision s'explique par plusieurs facteurs. Le premier facteur est la précision acceptable des corrections d'estimation des fournisseurs, qui permet à l'optimiseur de raisonner à partir d'estimations plus réalistes. Le deuxième facteur, et certainement le plus important, est l'optimisation dynamique. Celle-ci permet de ré-optimiser les plans d'exécution multi-clouds de sorte à limiter la quantité totale de données traitées par les plans d'exécution, et donc d'améliorer à la fois leurs perfor-

14. Grid'5000 est un banc d'essais géré par un groupement d'intérêt scientifique (GIS) hébergé par l'Inria, incluant le CNRS, RENATER, plusieurs universités ainsi que d'autres organisations. Voir `https://www.grid5000.fr`.

mances et leur coût monétaire. Le troisième facteur est la bonne précision du modèle d'apprentissage utilisé pour réévaluer les devis, qui parvient à correctement anticiper les corrections effectuées par l'optimisation dynamique. Concernant les stratégies de recherche, les résultats expérimentaux montrent que la stratégie itérative permet effectivement de passer à l'échelle, tout en améliorant la qualité des plans d'exécution qu'elle retient pour le calcul des devis grâce à la plus grande variété de plans qu'elle considère.

3 *Organisation du manuscrit*

Par la suite, un état de l'art sur la gestion et le traitement de données en environnements multi-clouds est proposé dans le chapitre 2. Les chapitres suivants traitent des contributions de la thèse. Le chapitre 3 présente l'architecture générale du middleware Nebula. Puis, le chapitre 4 présente les stratégies de recherche proposées successivement pendant cette thèse. Ensuite, le chapitre 5 détaille les méthodes utilisées pour diminuer la sous-optimalité des requêtes multi-clouds, c'est-à-dire le modèle de coûts multi-cloud et la méthode d'optimisation dynamique. Le dispositif expérimental et les résultats expérimentaux portant à la fois sur la pertinence de la sous-traitance des requêtes multi-clouds et l'évaluation de nos propositions sont détaillés et discutés dans le chapitre 6. Enfin, le manuscrit est conclut par le chapitre 7, qui rappelle les contributions de cette thèse et présente les perspectives ouvertes par ces travaux.

4 *Valorisation scientifique*

Nonobstant le présent manuscrit, ces travaux de thèse ont été valorisés par les publications et présentations suivantes.

Publications dans des conférences internationales avec comité de lecture

— D. T. Wojtowicz, S. Yin, F. Morvan et A. Hameurlain, *Cost-Effective Dynamic Optimisation for Multi-Cloud Queries*, *in* 2021 IEEE 14th International Conference on Cloud Computing (CLOUD), 2021, pp. 387-397, Chicago, États-Unis

— D. T. Wojtowicz, S. Yin, J. Martinez-Gil, F. Morvan et A. Hameurlain, *Multi-Cloud Query Optimisation with Accurate and Efficient Quoting*, *in* 2022 IEEE International Conference on Big Data (Big Data), 2022, pp. 228-233, Osaka, Japon

Publications dans des conférences nationales avec comité de lecture

— D. T. Wojtowicz, S. Yin et F. Morvan, *SLA Definition for Multi-Cloud Queries*, *in* 36ème Conférence sur la Gestion de Données : Principes, Technologies et Applications (BDA), 2020, p 80., 2 p., Paris, France

Présentations dans des conférences nationales avec comité de lecture

— D. T. Wojtowicz, S. Yin, F. Morvan et A. Hameurlain, *Cost-Effective Dynamic Optimisation for Multi-Cloud Queries*, *in* 38ème Conférence sur la Gestion de Données : Principes, Technologies et Applications (BDA), Catégorie « Articles publiés », 2022, Clermont-Ferrand, France

2

Gestion de données en environnements multi-clouds : un état de l'art

1 *Introduction*

Les premiers travaux recensés sur la thématique du multi-cloud, nommé « inter-cloud » ou encore « *cloud-of-clouds* » dans la littérature précoce, datent de 2010 [36]. Ils voient en celui-ci un intérêt sécuritaire pour les systèmes informatiques déployés dans le cloud, dans la mesure où la multiplication de fournisseurs permet de diluer la dépendance individuelle à chaque prestataire et donc à se protéger en cas de compromission de l'un d'entre eux. Cette notion de sécurité est à envisager d'une manière large, puisque la littérature sur le multi-cloud traite de thématiques allant de la cryptographie [37] à la sécurité d'un point de vue organisationnel. D'autres travaux ont été dédiés à l'interopérabilité entre fournisseurs cloud, afin de favoriser la transition d'un fournisseur à un autre en proposant des interfaces abstraites pour le déploiement de services dans le cloud [38, 39]. Plus tard, les travaux en gestion de données portant sur la thématique du multi-clouds ont suivi deux axes.

Le premier axe s'intéresse à l'utilisation des services de plusieurs fournisseurs cloud comme infrastructure dans la conception de SGD distribués, l'environnement multi-cloud étant justifié par des besoins en sûreté [22,

37, 40] (p. ex. en évitant la dépendance à fournisseur unique), en performances [41] (p. ex. en utilisant la géodistribution pour diminuer la latence), ou encore en passage à l'échelle [42, 43] (p. ex., en utilisant le cloud comme déversoir en cas de surcharge de la plateforme privée de l'utilisateur). Les systèmes proposés se situent à plusieurs niveaux d'abstraction, du allant du Système de Fichiers (SGF) [40, 44, 45] au SGBD [37, 41, 43, 46].

Les SGF multi-clouds proposés dans la littérature utilisent différentes stratégies de placement, de réplication et de chiffrage des données selon les fonctionnalités qu'ils implémentent (p. ex. pour la gestion des accès utilisateurs [47], des transactions [48], ou encore comme surcouche pour le stockage de données dans un SGBD [22]). Les architectures proposées sont dites « légères » ou « lourdes », respectivement dans le cas où le programme du SGF installé sur la machine où il est utilisé est suffisant à sa gestion [40, 49] et dans le cas où un ou plusieurs serveurs proxy sont en charge de la gestion des données [45, 48, 50-53].

Les SGBD multi-clouds proposés dans la littérature ont pour point commun l'utilisation de la plateforme IaaS de plusieurs fournisseurs pour le déploiement d'instances de ces systèmes. Certains utilisent les services de stockage de données de type DaaS ou bien des services de location d'espace disque plus classiques pour stocker les données du SGBD chez plusieurs fournisseurs. Ces systèmes font de la macro-optimisation : l'aspect monétaire induit par le cloud y est essentiellement pris en compte globalement, c'est-à-dire qu'ils cherchent à diminuer la facture finale pour le déploiement et le fonctionnement de leur système plutôt que le coût spécifique de chaque opération. Même si les SGF multi-clouds et les SGBD multi-clouds ne sont pas conçus pour de l'analyse de données publiques en environnements multi-clouds, ils restent cependant intéressants à étudier car ils reflètent l'impact du multi-cloud sur la conception des SGD, dans la mesure où ils doivent trouver un compromis entre coût monétaire et performances du système.

Le second axe suivi par la recherche prend le contrepied de cette approche globale, en étudiant la micro-optimisation sur les requêtes en environnements multi-clouds, à la fois dans le contexte du MapReduce et dans les modèles de coûts. La plupart des travaux s'intéressent soit à l'optimisation des performances en s'intéressant à l'allocation de ressources pour les opérateurs MapReduce [23-25], soit à l'optimisation des dépenses [26]. La nature multi-critères de l'optimisation de ces plans d'exécution a également été brièvement abordée [27], et a également été soulignée dans la littérature relative aux modèles de coûts multi-clouds [28, 29]. L'optimiseur de requêtes voit sa tâche devenir multicritères, c'est-à-dire qu'il doit trouver un compromis dans la minimisation de plusieurs fonctions de coûts, car les plans d'exécution qu'il propose ne doivent plus seulement être performants mais aussi peu couteux [30, 31].

Cet état de l'art de la littérature relative à la gestion et à l'intégration de bases de données en environnements multi-clouds se décompose en deux parties. La section 2 traite des SGBD en environnement multi-clouds. La section 3 présente les travaux existant en matière de traitements de données multi-clouds. Enfin, la section 4 conclut cet état de l'art.

2 *Systèmes de gestion de données distribués en environnements multi-clouds*

Les SGBD multi-clouds recensés dans la littérature sont de nature variée : on trouve ainsi des SGBD *Not only SQL* (NoSQL), des SGBD Relationnels (SGBDR) et des multistore, utilisant les possibilités offertes par le cloud de manière variée. En effet, là où certains systèmes utilisent le DaaS pour la sous-traitance du stockage des données, d'autres préfèrent déployer des machines virtuelles sur une offre IaaS afin d'héberger un SGBD ou une instance de leur système. Le cloud hybride est utilisé par certains SGBD, qui sont alors déployés simultanément sur une infrastructure privée complémentée par des clouds publics. Une présentation des SGBD multi-clouds de la littérature, classifiés par nature des données qu'ils gèrent, est proposée ci-après.

2.1 *Systèmes au modèle de données homogène*

L'essentiel des efforts relatifs à l'étude des SGBD multi-clouds ont été consacrés aux systèmes NoSQL, en particulier sur les systèmes clé-valeur. Ce phénomène s'explique par le fait que les différentes offres DaaS sont des *object-store* aux interfaces similaires à celles des SGBD clé-valeur. Une organisation utilisant déjà un service DaaS peut donc facilement migrer vers un SGBD multi-cloud de type clé-valeur, de manière transparente pour les applications déjà construites pour fonctionner avec un stockage DaaS. Outre les systèmes clé-valeur, la recherche s'est aussi intéressée à la conception de SGBD orientés graphe et de gestionnaires de ressources Hadoop en environnements multi-clouds.

2.1.1 *Systèmes clé-valeur*

Les SGBD clé-valeur gèrent un tableau associatif, dans lequel les données, abstraites comme étant des valeurs, sont récupérables à l'aide d'une clé unique qui leur est associée [1]. Les données peuvent être des enregistrements contenant des champs (p. ex. au format JSON), des fichiers (p. ex. une image), des objets ou encore des données non-structurées comme du texte libre. L'essence de l'interface des SGBD clé-valeur consiste en quelques primitives [3, ch.

1. La relation entre les clés et les valeurs est en fait une application de domaine fini.

11, p.521] présentées ci-dessous, qui peuvent être étendues et complétées par les différents SGBD (p. ex. pour la gestion des transactions ou le groupement d'opérations).

PUT(K, V) Création ou mise à jour d'une entrée associant une clé K à une valeur V.

GET(K) Récupération de la valeur associée à la clé K.

DELETE(K) Supprime l'entrée associée à la clé K.

La similarité de cette interface avec les *object-store* des services DaaS facilite largement l'encapsulation de celles-ci au sein d'un SGBD clé-valeur multi-cloud, simplifiant ainsi la mise à jour des programmes d'application existants lors d'un éventuel passage au multi-cloud. Plusieurs architectures de SGBD clé-valeur multi-clouds différentes ont été proposées, conçues pour l'exploitation de plusieurs fournisseurs cloud afin de répondre à des besoins divers en termes de volume de données et de géodistribution. Si certains systèmes, à l'architecture centralisée [54, 55] ou décentralisée [41, 56], stockent l'intégralité des données qu'ils gèrent dans le cloud, une approche hybride voyant le cloud comme un complément à une infrastructure privée a été proposée [46]. Ces différentes contributions sont présentées ci-après.

Le premier SGBD clé-valeur multi-cloud recensé est ICStore [54]. Celui-ci est déployable sur un serveur et utilise à la fois des *object-store* DaaS et des service DBaaS type clé-valeur pour stocker les données qu'il gère. Ce système fonctionne en trois couches, chacune associée à un niveau de sécurité (chiffrement, signature puis *erasure coding*), qui doivent être traversées par les requêtes. ICStore est conçu pour la sécurité : l'impact sur les performances des requêtes et le coût monétaire général du système n'ont pas été pris en compte lors de la conception de ce système. Enfin, son architecture est fortement centralisée et crée un goulet d'étranglement limitant la scalabilité du système au passage à l'échelle vertical. Le système CHARM [55] partage cette limite avec ICStore, dans une moindre mesure car sa conception est plus simple. En effet, il s'agit d'un proxy gérant de manière transparente le placement et la réplication des données sur l'infrastructure DaaS de plusieurs fournisseurs, et assurant la redirection des requêtes chez ces derniers. Le goulet d'étranglement, situé au niveau de la redirection, reste néanmoins présent et rend peu réaliste l'utilisation de ce système à grande échelle.

Plusieurs systèmes ont donc été conçus pour pouvoir passer à l'échelle de manière horizontale. Le premier d'entre eux, MetaStorage [56], fonctionne lui aussi à partir de composants agencés en couches. La première couche est un ensemble d'adaptateurs, déployés sur l'infrastructure IaaS des fournisseurs proposant du DaaS utilisés pour le stockage, ayant pour objectif de fournir une interface virtuelle unifiée entre les différents stockages sous-jacents. La deuxième couche est un ensemble d'instances d'un couple coordinateur-distributeur déployés sur une offre IaaS. Le coordinateur est en

charge de la collaboration entre les instances du système et de l'aiguillage des requêtes ; l'un d'entre eux ayant le rôle de maître afin de maintenir une liste des instances, chacune d'entre elle pouvant prendre le relais en cas de perte du maître. Le distributeur gère le placement des données, leur réplication, et leur récupération. La géodistribution, à la fois des données et des instances système, permet d'améliorer les performances des requêtes perçues par les utilisateurs en diminuant la latence de la réponse. Elle permet également d'améliorer la continuité de service en cas de perte de contact avec un fournisseur cloud. Une architecture similaire a été proposée pour le système SPANStore [41], également dans une optique de gestion de données géo-distribuées à grande échelle. Ce système est un SGBD clé-valeur transactionnel : il propose des garanties de cohérence des données à tout instant. Ce système suit une politique de réévaluation périodique de l'existence des réplicats afin d'éviter les surcoûts liés au stockage de réplicats peu utilisés, et d'améliorer les performances et diminuer le coût monétaire des opérations de mise à jour des données.

Enfin, une approche hybride du multi-cloud combinant cloud privé et cloud public a été introduite dans le contexte des SGBD clé-valeur par Hybris [46]. Ce système conserve toujours un réplicat des données sur une infrastructure privée, et utilise des services IaaS pour déployer dynamiquement des instances du système sur lesquelles sont stockés les réplicats, créés par *erasure coding*, à des fins de tolérance aux fautes et de passage à l'échelle. La politique de réplication proposée est optimale à terme d'un point de vue monétaire car elle permet de limiter la réplication à une duplication.

2.1.2 *SGBD orientés graphe*

Acacia [57] est un SGBD orienté graphe conçu pour du cloud hybride. Il utilise les ressources IaaS et voit en le cloud public un déversoir permettant de passer temporairement à l'échelle lorsque la plateforme privée est saturée. Son architecture est centralisée : un processus maître contrôle le système, qui est composé d'un ensemble de travailleurs chargés de stocker et d'interroger une partie des données. Acacia cherche à la fois à minimiser le temps de réponse des requêtes qui lui sont soumises et à minimiser l'espace de stockage consommé par le système. Le partitionnement des données dépend de la densité du graphe géré, de sorte à éviter des transferts de données entre nœuds de calcul et entre fournisseurs, qui ralentiraient l'exécution des requêtes. L'optimisation du système par rapport à l'environnement multi-cloud est macroscopique : c'est au travers du placement des travailleurs et des données que les coûts monétaires sont minimisés.

2.1.3 *SQL-on-Hadoop*

Le système TIRAMOLA [58-60] propose d'utiliser des ressources IaaS en supplément de ressources privées pour déployer des instances d'un cluster Hadoop modifié, les ressources privées hébergeant le contrôleur du système. TIRAMOLA est élastique : l'*upscaling* et le *downscaling* se font en suivant un ensemble de règles définies par les utilisateurs, prenant en compte à la fois la quantité de requêtes à traiter et le coût monétaire. Ces règles sont incorporées dans un processus de prise de décision Markovien à partir duquel de l'apprentissage par renforcement Q-learning est effectué, dans une optique d'amélioration progressive du processus de prise de décision. Des stratégies particulières pour l'allocation de ressource dans le cadre de requêtes SPARQL ont été proposées, afin de prendre en compte le nombre de lectures et d'écriture que les requêtes vont engendrer. TIRAMOLA peut par ailleurs être utilisé comme infrastructure sous-jacente pour le stockage de tables interrogeables avec SQL et gérées avec HBase [2] ou Cassandra [3].

2.1.4 *SGBD relationnels*

Les SGBDR multi-clouds recensés dans la littérature ont eux aussi une architecture hybride. Cumulus [61] est un SGBDR multi-cloud transactionnel. Son architecture est hybride : les contrôleurs, en charge du partitionnement des données et de l'aiguillage des requêtes sur les différentes instances du système déployées sur des infrastructures IaaS, sont centralisés, tandis que lesdites instances coopèrent pour la mise en œuvre des transactions. Le système est équipé d'une stratégie de réplication et de placement dynamique, qui cherche d'abord à diminuer le nombre de transactions distribuées réalisées par le système, puis à lisser la charge sur l'ensemble des instances du système. Les instances sont également créées et supprimées dynamiquement en fonction de la charge de travail.

À l'opposé de Cumulus, SHAMC [37] est un SGBDR multi-cloud décisionnel. Le système utilise des ressources IaaS pour le déploiement d'un ensemble d'instances disposant chacune d'un réplicat de la base de données. La géo-distribution est exploitée pour diminuer le risque de perte de données en cas de défaillance d'un fournisseur et également la latence à l'interrogation, mais a un impact négatif sur les performances des opérations de mise à jour des données. Les données de ce système sont stockées et manipulées en étant chiffrées de manière homomorphique, de sorte à ce que les opérations de l'algèbre relationnelles effectuées pendant le traitement des requêtes ne nécessitent pas de déchiffrement des données qu'ils manipulent, évitant ainsi les fuites de données en cas de corruption d'un fournisseur cloud. Les interactions avec SHAMC se font au travers d'un logiciel installé localement, qui

2. Voir `https://hbase.apache.org/`.
3. Voir `https://cassandra.apache.org/`.

aiguille les requêtes de ses utilisateurs vers une instance du système dans le cloud afin de lisser la charge de travail à travers le système tout en cherchant à diminuer la latence. Ce logiciel permet par ailleurs de déchiffrer le résultat des requêtes avant de présenter leur résultat aux utilisateurs.

2.2 *Systèmes multistores*

L'hétérogénéité des différentes offres cloud ainsi que la variété des systèmes NoSQL existant a naturellement conduit à la conception de systèmes multistore multi-clouds. Deux systèmes ont été recensés dans la littérature, tous deux issus du même projet de recherche [4] [42]. Dans les deux cas, les systèmes sont soumis à des Clé-valeur concernant l'application SaaS pour laquelle ils servent de SGD : leurs décisions reflètent ainsi les contraintes définies par les locataires. De plus, les deux systèmes utilisent le cloud public en complément d'un cloud privé, d'une part comme déversoir pour absorber une charge de travail inhabituelle et d'autre part pour offrir des garanties de sécurité en matière de perte de données. La stratégie de *polyglot persistence* [5] pour le stockage des données de ces applications est justifiée à la fois par leur hétérogénéité et leur grand volume.

Le premier système, nommé PERSIST [62], est un middleware servant d'infrastructure sous-jacente à un logiciel de gestion de *logs* proposé en SaaS, produisant un flux de données massif en provenance de différents domaines d'application. Les SGBD NoSQL sous-jacent de PERSIST sont Cassandra et MongoDB [6], déployés en partie sur du cloud privé et du cloud public sur plate-forme IaaS. Les informations stockées par le système étant parfois sensibles, les SLA dirigeant la plate-forme permettent aux locataires de définir des attentes en matière de réplication, de chiffrement et de localisation du placement des données. Ces derniers contraignent les décisions des composantes en charge de l'*upscaling* et du *downscaling* des déploiements dans le cloud public, et des mouvements de données qu'ils entraînent, afin que les décisions du middleware minimisent ses coûts monétaires tout en garantissant le respect des SLA .

Le second système est le middleware SCOPE [43], originellement un SGBD orienté colonnes multi-cloud [42] puis étendu en multistore. Sa reconfiguration, c'est-à-dire la quantité, la nature et la localisation des services cloud utilisés par le middleware, est effectuée périodiquement de manière décentralisée. L'objectif principal de SCOPE est de faire respecter des SLO. Ceux-ci

4. Il s'agit du projet DeCoMAdS, conduit par des membres de la *Katholieke Universiteit Leuven* (Louvain, Belgique), s'intéressant à la conception de middleware sous-jacent à des offres SaaS. Voir `https://distrinet.cs.kuleuven.be/research/projects/DeCoMAdS`.

5. C'est-à-dire l'utilisation de plusieurs SGBD de nature différente en fonction de l'application finale les utilisant. Leurs propriétés sont alors complémentaires à l'échelle du système d'information.

6. Voir `https://www.mongodb.com/`.

Type	Réf.	Système	Architecture	Stockage	Objectif
Clé-valeur	[54]	ICStore	Centralisée	DaaS	Sécurité
Clé-valeur	[55]	CHARM	Centralisée	DaaS	Sécurité
Clé-valeur	[56]	MetaStorage	Décentralisée	DaaS	Tolérance aux fautes, Scalabilité
Clé-valeur	[41]	SPANStore	Décentralisée	DaaS	Transactions, Scalabilité
Clé-valeur	[46]	Hybris	Hybride	IaaS	Coût monétaire, Performances, Sécurité, Scalabilité
Orienté graphe	[57]	Acacia	Hybride	IaaS	Coût monétaire, Performances
SQL-on-Hadoop	[59]	TIRAMOLA	Hybride	IaaS	Coût monétaire, Performances
Relationnel	[61]	Cumulus	Hybride	IaaS	Transactions, Performances, Scalabilité
Relationnel	[37]	SHAMC	Décentralisé	IaaS	Sécurité
Multistore	[62]	PERSIST	Hybride	IaaS	Scalabilité, Coût monétaire, Performances
Multistore	[43]	SCOPE	Hybride	IaaS	Performances, Scalabilité, Coût monétaire

TABLE 2.1 – Comparaison entre les SGBD multi-clouds de l'état de l'art

sont inclus dans les SLA liant indirectement le système aux locataires de l'application SaaS, un éditeur de documents collaboratif, utilisant SCOPE comme SGBD. SCOPE maximise ses performances, plus précisément en cherchant à minimiser la latence perçue par les utilisateurs dans l'application SaaS, puis minimisera ensuite le coût monétaire *ceteris paribus*.

2.3 *Synthèse*

Plusieurs systèmes de traitement de requêtes faisant intervenir différents fournisseurs cloud existent. Le tableau 2.1 présente un rapide récapitulatif de leurs différentes caractéristiques. Dans les systèmes dont la nature le permettrait (orientés graphe, SQL-on-Hadoop, relationnels et multistore), l'environnement multi-cloud est pris en compte dans l'allocation des ressources. Enfin, tous ces systèmes ont pour point commun de gérer eux-même leurs données ; leur déploiement en environnements multi-clouds ne les rend pas plus intéressants que d'autres systèmes, utilisables en local ou sur des offres mono-cloud, pour l'analyse de données publiques issues de plusieurs fournisseurs cloud.

3 *Optimisation de requêtes multi-clouds*

La littérature relative au traitement de données en environnements multi-clouds est moins étendue que son homologue relative à la gestion de données. Ce sujet a essentiellement été étudié dans le contexte des *workflow* MapReduce,

leur nature décentralisée et fortement distribuée ayant invité à une réflexion sur l'opportunité de les distribuer au-delà d'un cloud unique, pour mettre en place des traitements de données multi-clouds. L'aspect multicritères de l'optimisation de ces *workflow*, et plus généralement des requêtes multi-clouds, a aussi été évoqué sous l'angle des modèles de coûts.

3.1 *Allocation de ressources*

L'opportunité d'utiliser des ressources issues de plusieurs fournisseurs cloud pour la mise en place d'un système Hadoop a d'abord été étudiée pour utiliser des ressources issues de cloud public en complément de à des infrastructures privées, principalement dans le cas d'analyse de données issues de différentes sources potentiellement confidentielles et donc impossibles à déplacer dans un cloud public. Les premiers travaux sur le sujet se sont concentrés sur la faisabilité de l'utilisation d'environnements multi-clouds [63, 64], en fournissant des outils techniques déployables à la fois sur une infrastructure privée et sur le service EC2 d'Amazon[7], et sur l'impact de cette approche sur les performances des analyses de données.

Les communications intra-cloud et inter-cloud étant critiques pour les performances de MapReduce, une méthode de dimensionnement dynamique des réservations de réseaux dédiés entre les fournisseurs cloud a été proposée, complétée par une stratégie de réservation dynamique de machines virtuelles dans le cloud [65]. Outre le dimensionnement automatique des composantes du cluster Hadoop, l'allocation des ressources pour les tâches des *workflow* MapReduce a également été étudiée. Le placement des données [66] et des tâches [67] ont en effet été étudiés afin de minimiser les transferts sur le réseau, respectivement par anticipation de la charge de travail, et en fonction de la configuration courante.

Par ailleurs, l'aspect monétaire des réservations de ressources IaaS pour les machines virtuelles et le réseau a été abordé. Un système centralisé en charge du placement des tâches MapReduce sur diverses infrastructures cloud en maximisant les performances tout en minimisant le coût monétaire des traitement a été suggéré [27]. Ce dernier reste cependant embryonnaire. De plus, la centralisation des décisions sur une seule machine à l'échelle d'un cluster Hadoop déployé en environnements multi-clouds crée un goulet d'étranglement appelant à l'étude de la décentralisation de ces décisions. Les conclusions de ces travaux sont néanmoins intéressants, car ils démontrent qu'il est possible de gagner en performances tout en diminuant les coûts monétaires d'une charge de travail en exploitant plusieurs fournisseurs cloud en comparaison avec traitement mono-cloud.

7. Voir `https://aws.amazon.com/fr/ec2/`.

3.2 *Modèles de coûts multi-clouds*

L'aspect multicritères de l'optimisation des charges de travail en environnements multi-clouds a été évoqué dans la littérature relative aux modèles de coûts pour le SGBD NoSQL HBase. Un modèle de coûts conçu pour le placement de tâches dans le contexte de fournisseurs cloud facturant le temps d'exécution des requêtes a été conçu [28, 29]. Celui-ci cherche à modéliser la satisfaction utilisateur, en évaluant les économies réalisables par les différents choix d'allocation de ressources pondérées par leurs performances respectives. Les estimations de prix et de temps de réponse des composantes des requêtes multi-clouds sont faites à l'aide d'une fonction mathématique. Le modèle de coûts est donc produit donc des estimations rapidement, mais est très sensible au paramétrage initial de la fonction.

3.3 *Synthèse*

Les travaux sur l'optimisation de requêtes multi-clouds sont relativement peu nombreux. Concernant l'allocation de ressources et le placement de tâches, ceux-ci se sont déroulés dans le cadre de clusters Hadoop et de *workloads* MapReduce. Leur optimisation multicritères a été étudiée, dans l'objectif de minimiser les coûts monétaires et de maximiser les performances.

Un modèle de coûts au service de cette optimisation multicritères en environnements multi-clouds a par ailleurs été proposé. Ce dernier a cependant des limitations inhérentes à son paramétrage, et des avancées plus récentes en matière d'auto-adaptation des SGBD [68] pour le raffinement progressif des modèles analytiques utilisés pourraient être utilisées.

4 *Conclusion*

La littérature relative à la gestion et au traitement de données en environnements multi-clouds s'est principalement concentrée sur l'étude de SGD multi-cloud, plus précisément de SGF et de SGBD multi-cloud, en particulier des SGBD NoSQL de type clé-valeur. Cette situation s'explique par la similarité de l'interface des *object-store* DaaS, utilisés de plus en plus massivement, avec les SGBD clé-valeur et par la facilité de les encapsuler dans ces derniers ou de les cacher derrière un SGF. Le multi-cloud étant une thématique issue de la recherche en sécurité [36], la plupart des travaux se focalisent sur les apports sécuritaires du multi-cloud ou sur les performances globales des systèmes proposés. Deux SGBDR multi-clouds ont été recencés : le premier est transactionnel [61], et les apports du second relèvent plus de la cryptographie que de l'efficacité de la gestion de données [37]. La conception d'optimiseurs de requêtes multi-clouds est absente de la littérature sur les SGBDR multi-

clouds, et c'est du côté de l'allocation de ressources pour le MapReduce et des systèmes SGBD non-relationnels que l'on trouve des travaux à ce sujet. Quelque soit le cadre dans lequel se placent ces travaux, ils finissent par converger vers le constat que l'optimisation de requêtes multi-clouds doit être multicritères [27, 29]. Les environnements multi-clouds sont donc des extensions de l'environnement distribué dans lequel l'optimiseurs doit prendre en compte les dépenses induites par l'utilisation d'infrastructures dans le cloud, qu'elles soient de type PaaS, IaaS, DaaS ou encore DBaaS. Un premier modèle de coûts multi-cloud intégré à HBase a été proposé [29], ouvrant alors la voie à plus de travaux sur l'optimisation de requêtes multi-clouds.

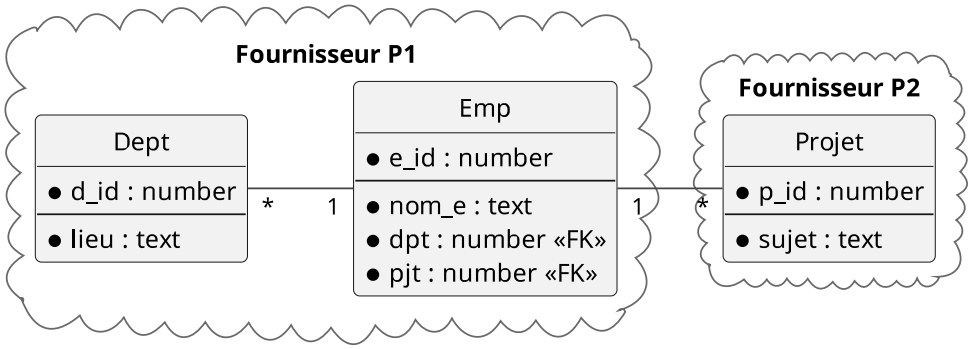
L'étude de la littérature montre que cette voie reste largement à explorer. En effet, les modalités des interactions entre les utilisateurs et les systèmes permettant de traiter les requêtes multi-clouds restent à définir, notamment en raison de l'aspect monétaire induit par le cloud. Par ailleurs, les différences entre l'espace de recherche des plans d'exécution multi-clouds et celui des plans d'exécution distribués appelle à l'adaptation des stratégies de recherche classiques voire à la découverte de nouvelles méthodes. Ensuite, les récents progrès en termes de modèles de coûts appris invitent à les appliquer aux requêtes multi-clouds. Enfin, le succès qu'a rencontré par le passé l'optimisation dynamique de requêtes dans différents environnements (monoprocesseur, distribué, parallèle, etc.) indique que cette technique sera certainement adaptée à ce nouvel environnement.

3

Nebula : un middleware pour le traitement de requêtes multi-clouds

1 *Introduction*

Les travaux de cette thèse ont conduit à la conception d'un middleware dédié au traitement de requêtes multi-clouds. Celui-ci, nommé Nebula, fait le lien entre des utilisateurs et des fournisseurs cloud, plus précisément avec leur offre DBaaS proposant à l'analyse des données publiques. Le middleware permet aux utilisateurs de lister les jeux de données qui les intéressent au sein d'un schéma multi-cloud, de calculer des devis pour les requêtes multi-clouds qu'ils formulent en SQL sur différentes sources, et d'orchestrer la sous-traitance de l'exécution de ces requêtes multi-clouds auprès des fournisseurs cloud. Cette dernière permet au middleware de ne pas nécessiter de ressources importantes pour le faire fonctionner (un simple ordinateur relié à internet est suffisant). Nebula impose à ses utilisateurs de suivre une procédure en deux étapes lorsqu'ils souhaitent interroger les sources de données : ils doivent d'abord accepter un devis afin d'autoriser le middleware à orchestrer l'exécution de la requête multi-cloud. Ce système de devis permet aux utilisateurs de garder le contrôle sur leurs dépenses, de sorte à éviter de mauvaises surprises au moment de la facturation. Ces devis sont chiffrés à partir de plans d'exécution multi-clouds générés par un optimiseur suivant



(a) Diagramme entité-association du schéma multi-cloud

```sql
SELECT nom_e, lieu
FROM Emp, Dept, Projet
WHERE dpt = d_id
  AND pjt = p_id
  AND lieu IN ('Toulouse', 'Linz')
  AND sujet LIKE '%multi-cloud%';
```

(b) Code SQL de la requête

FIGURE 3.1 – Exemple fil-rouge d'un schéma multi-cloud et d'une requête sur celui-ci

une stratégie de recherche et utilisant des estimations produites par le SGBD des fournisseurs cloud corrigées par un modèle de coûts. Le middleware a donc différentes composantes, travaillant ensemble pour le calcul des devis et l'orchestration de la sous-traitance de l'exécution des requêtes multi-clouds.

Le présent chapitre présente l'architecture générale du middleware, et est organisé comme suit. Les différents composants de Nebula et leur articulation sont d'abord présentés dans la section 2, puis les mécanisme des devis est présenté dans la section 3. Enfin, ce chapitre est conclu par la section 4.

Par ailleurs, afin de faciliter la compréhension des lecteurs, un exemple est proposé en fil-rouge (cf. fig. 3.1) des chapitres traitant des contributions de cette thèse. En supposant qu'il existe le schéma multi-cloud présenté en figure 3.1a, où deux relations `Dept` et `Emp` sont hébergées chez le fournisseur P_1 et une relation `Projet` chez P_2, nous considèrerons la requête multi-cloud présentée en figure 3.1b qui permet de récupérer le nom et le lieu des employés situés à Toulouse ou à Linz, travaillant sur un projet en lien avec le multi-cloud.

2 *Composants du middleware*

L'architecture de Nebula a été fortement influencée par l'architecture traditionnelle des SGBD dérivée de INGRES [69] et de System R [70]. Ses

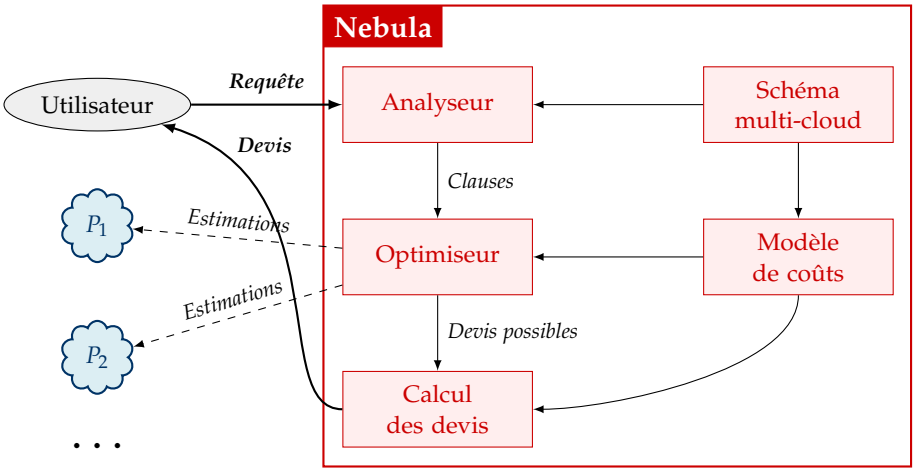


FIGURE 3.2 – Intéractions entre les composants de Nebula pour le calcul du devis

composants et leur intéractions pour l'estimation des devis sont illustrés par la figure 3.2. Lorsqu'une requête multi-cloud est proposée à Nebula, son analyseur vérifie qu'elle est syntaxiquement correcte et sémantiquement cohérente avec le contenu du schéma multi-cloud. Celui-ci extrait l'ensemble des clauses de la requêtes et le transmet à l'optimiseur. Il explore l'espace des plans d'exécution multi-clouds possibles, plus précisément un sous-ensemble de ce dernier qui lui est accessible en suivant une stratégie de recherche, afin de recenser des plans d'exécution se situant sur un front de Pareto dans un espace coût monétaire - performances. Ces plans sont transmis au composant chargé d'estimation des devis. Ce dernier fait ré-estimer par le modèle de coûts les estimations de temps de réponse et de coût monétaire des plans, et filtre ceux qui ne sont plus sur le front de Pareto. Ces estimations deviennent alors des devis, qui sont présentés à l'utilisateur. Si ce dernier accepte un devis, alors Nebula procèdera à l'exécution de la requête : l'optimiseur orchestre alors l'exécution des composantes du plan d'exécution associé au devis. Les valeurs de cardinalité en sortie, de coûts monétaires et de temps de réponse constatées à l'exécution sont transmises au modèle de coûts pour apprentissage. Une description plus détaillée de chaque composant est proposée ci-après.

2.1 *Schéma multi-cloud*

Le schéma multi-cloud permet de connaître les informations de placement chez les fournisseurs des relations, la politique tarifaire de leur hébergeur ainsi que la liste de leurs attributs. Il est construit avec l'aide de l'utilisateur, qui recense les bases de données qu'il souhaite exploiter en formulant des requêtes multi-clouds. Il est utilisé par l'analyseur sémantique pour vérifier si les requêtes interrogent bien des relations et des attributs connus. Il est

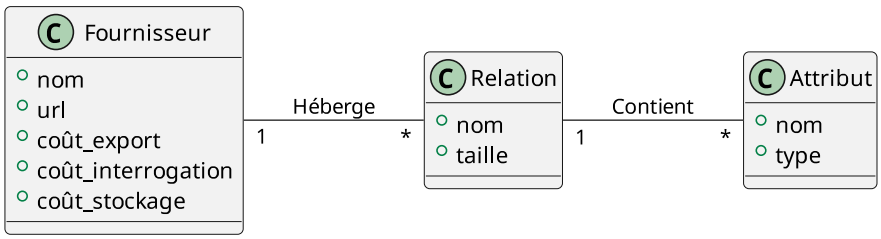


FIGURE 3.3 – Diagramme de classes du schéma multi-cloud

également utilisé par le modèle de coûts pour le calcul des coûts monétaires des composantes des plans d'exécution multi-clouds.

La modélisation des données du schéma multi-cloud est illustrée par la figure 3.3. Formellement, le schéma multi-cloud est modélisé comme étant l'ensemble $\mathcal{R}$ des relations, chacune hébergée sur un fournisseur $P_R \in \mathcal{P}$ et composées d'un ensemble d'attributs A_R. La politique tarifaire des fournisseurs est également comprise dans le schéma multi-cloud, en supposant que les opérations d'interrogation (*querying*), d'export et de stockage sont proportionnelles à la quantité de données manipulées, respectivement notées $\epsilon_P^{(Q)}$, $\epsilon_P^{(E)}$ et $\epsilon_P^{(S)}$ pour un fournisseur $P \in \mathcal{P}$.

2.2 *Analyse syntaxique et sémantique*

Les requêtes multi-clouds sont écrites en SQL. Par la suite, on considère qu'elles sont de type *Select-Project-Join* (SPJ), c'est-à-dire qu'elles effectuent des jointures entre toutes les relations sur lesquelles elles portent, qu'elles comprennent une projection et éventuellement de prédicats de sélection autres que les prédicats de jointure (c-à-d une requête sans clause `GROUP BY` ni `HAVING`). On suppose également que les requêtes ont une forme acyclique, c'est-à-dire qu'elles ont une forme en étoile, en arbre ou en chaîne, car la transformation d'une requête au graphe cyclique en une autre requête équivalente au graphe acyclique est un problème NP-difficile [71] déjà traité dans le contexte des SGBD distribués [72, ch. 8, p. 269] et donc hors sujet de cette thèse.

En suivant ces hypothèses, l'analyseur procède à vérification syntaxique et sémantique des requêtes multi-clouds. En supposant une requête Q valide, il en extrait un ensemble de clauses, noté C_Q, dans lequel chaque élément $c_i \in C_Q$ est un triplet $<\tau_i, \varphi_i, A_i>$ associant chaque prédicat φ_i à son type $\tau_i \in \{\Pi, \sigma\}$ [1] et un ensemble d'attributs A_i intervenant dans φ_i à partir desquels il est possible de retrouver grâce au schéma multi-cloud (cf. fig. 3.3)

1. Conformément à la notation classique de l'algèbre relationnelle, Π est l'opérateur de projection et σ de sélection.

les relations et les fournisseurs impliqués dans φ_i. L'ensemble C_Q de la requête en exemple est présenté ci-dessous dans l'équation 3.1.

$$C_Q = \left\{ \begin{array}{l} c_1 = <\overbrace{\sigma}^{\tau_1}, \overbrace{\text{« dpt = d\_id »}}^{\varphi_1}, \overbrace{\{d\_id, dpt\}}^{A_1}>, \\ c_2 = <\sigma, \text{« pjt = p\_id »}, \{p\_id, pjt\}>, \\ c_3 = <\sigma, \text{« lieu IN ('Toulouse', 'Linz') »}, \{lieu\}>, \\ c_4 = <\sigma, \text{« sujet LIKE '\%multi-cloud\%' »}, \{sujet\}>, \\ c_5 = <\Pi, \text{« nom\_e, lieu »}, \{lieu, nom\_e\}> \end{array} \right\} \quad (3.1)$$

2.3 *Optimiseur*

L'optimiseur est le composant ayant reçu le plus d'attention durant ces travaux de thèse. Il intervient à la fois lors du calcul des devis et lors de l'optimisation dynamique de la requête. Pour le calcul des devis, celui-ci prend l'ensemble des clauses C_Q retourné par l'analyseur pour produire un ensemble de plans d'exécution multi-clouds $\mathcal{E}_Q$ permettant de répondre à Q. Les fournisseurs cloud sont vus comme des boîtes noires prenant en entrée des requêtes SQL. Ils peuvent soit les exécuter, soit retourner des estimations sur la cardinalité en sortie, le temps de réponse et le coût monétaire de ces dernières. En considérant les fournisseurs comme des boîtes noires, on fait ainsi évoluer la nature du problème auquel l'optimiseur s'attaque en contraignant les choix qu'il peut faire.

Il s'agit finalement d'une extension du problème d'optimisation de requêtes en environnement distribué, dans lequel l'optimiseur doit construire un plan d'exécution multi-cloud en composant des sous-requêtes et en les ordonnançant. Il pourra donc faire varier les clauses qu'elles contiennent, leur ordre, ainsi que leur placement chez les fournisseurs lorsque c'est possible ; le choix de l'ordre des jointures et des algorithmes à employer dans celles-ci restant à la charge de l'optimiseur du fournisseur.

Afin de bien visualiser ce qu'est un plan d'exécution multi-cloud au sein de Nebula, un exemple est présenté dans la figure 3.4. Les résultats finaux sont stockés dans la relation `Res` chez le fournisseur P_1. Le plan d'exécution produit par l'optimiseur permet de faire du parallélisme en *pipeline*. Les opérations de suppression des résultats intermédiaires sont réalisées dès que possible et de manière asynchrones afin d'éviter de faire attendre inutilement leur complétion à l'utilisateur et ne sont donc pas comptées dans le temps de réponse du plan d'exécution multi-cloud.

L'optimisation est guidée par une stratégie de recherche, objet abstrait noté $\mathcal{T}_Q$ regroupant l'ensemble des plans d'exécution possibles, afin de construire l'ensemble $\mathcal{E}_Q \subseteq \mathcal{T}_Q$ des plans d'exécution situés sur un front de Pareto. Ce dernier se place dans un espace dont les dimensions sont les performances

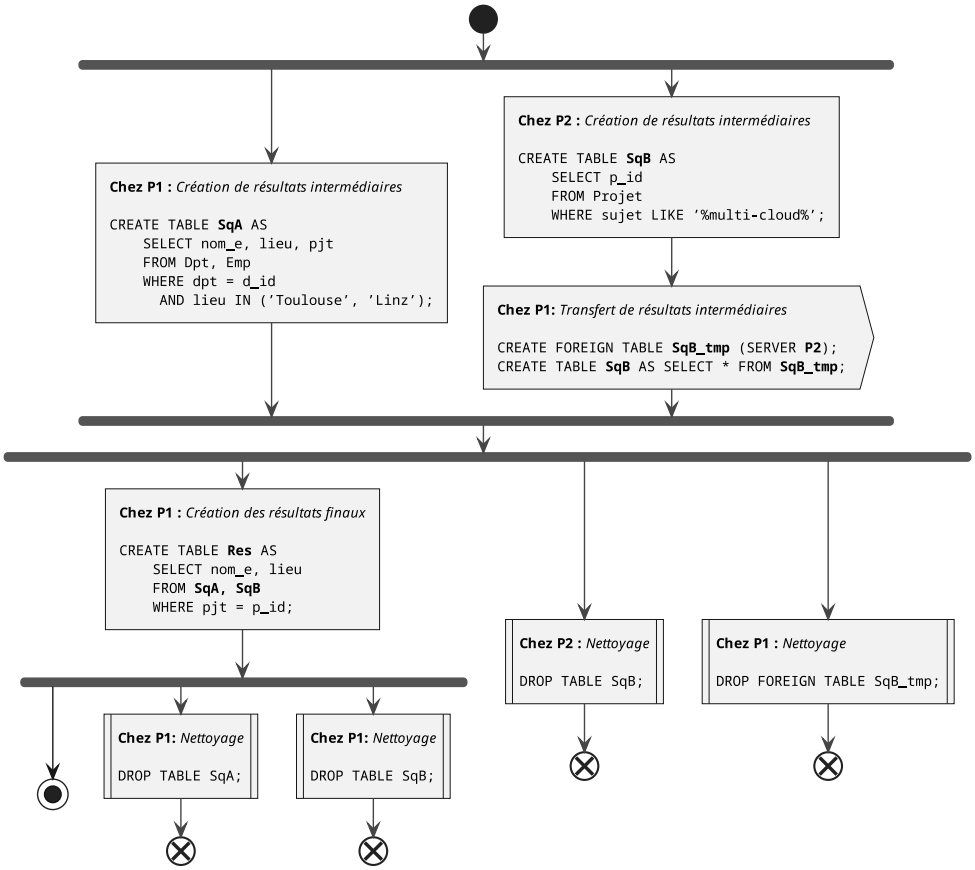


FIGURE 3.4 – Exemple d'un plan d'exécution multi-cloud possible pour répondre à la requête de la figure 3.1b.

et le coût monétaire des requêtes multi-clouds. La stratégie employée utilise des heuristiques, car la nature NP-difficile du problème d'optimisation de requêtes rend inenvisageable l'exploration de la totalité de l'espace de recherche [73]. Ces heuristiques ont nécessairement des conséquences sur les performances de l'optimisation du fait de la taille de l'espace de recherche qu'elles autorisent à explorer, ainsi que sur la qualité de $\mathcal{E}_Q$ en raison de leurs hypothèses sous-jacentes.

Deux stratégies de recherche ont été proposées pendant cette thèse : une exhaustive et une aléatoire itérative présentées dans le chapitre 4. Le contenu de $\mathcal{E}_Q$ dépend également de la fonction objectif considérée, ainsi que de la précision des estimations sur la cardinalité en sortie et les performances des sous-requêtes. Ces estimations sont issues des fournisseurs, puis ré-estimées par le modèle de coûts multi-cloud présenté ci-après.

De plus, l'optimiseur est dynamique. Il modifie les plans d'exécution après l'exécution des sous-requêtes qu'ils comportent pour diminuer leur sous-optimalité induite par les erreurs d'estimation persistantes malgré leur

correction par le modèle de coûts présenté ci-après. La méthode d'optimisation dynamique repose sur un SMA. Elle est présentée dans la section 3 du chapitre 5.

2.4 *Modèle de coûts multi-cloud*

Utiliser directement l'estimation des cardinalités en sortie et du temps de réponse des sous-requêtes produits par les fournisseurs exposerait le middleware aux erreurs commises par les fournisseurs, conduisant à des devis de mauvaise qualité et à de mauvais choix d'optimisation. S'agissant en fait d'estimations produites par le modèle de coûts du SGBD des fournisseurs, il est possible qu'elles aient jusqu'à plusieurs ordres de grandeur de décalage avec la réalité [33] à cause des méthodes employées pour résoudre le problème d'estimation du facteur de sélectivité des sélections et des jointures dans les SGBDR. Nebula se propose donc de corriger l'estimation de ce facteur en s'inspirant de récents travaux sur le sujet [34] en utilisant des modèles d'apprentissage automatique en ligne (*online machine learning*).

Le modèle de coûts intervient également auprès du calculateur de devis pour ré-évaluer les devis. En effet, l'utilisation de l'optimisation dynamique provoque des changements dans les plans d'exécution multi-clouds. Dès lors, il convient de les anticiper afin d'éviter de présenter aux utilisateurs des devis peu précis. Là encore, la ré-évaluation des devis se fait par apprentissage automatique en ligne.

Le modèle de coûts multi-cloud est présenté en détails dans la section 2 du chapitre 5.

2.5 *Calculateur de devis*

Le calculateur de devis intervient suite au processus d'optimisation lorsqu'un utilisateur demande un devis. Il filtre l'ensemble $\mathcal{E}_Q$ des plans d'exécution et travaille conjointement avec le modèle de coûts pour corriger les estimations faites par l'optimiseur sur le temps de réponse et le coût monétaire des requêtes multi-clouds afin de ne conserver que les plus compétitifs pour construire l'ensemble des devis $\mathcal{D}_Q$ qui sera présenté aux utilisateurs.

Un devis $D \in \mathcal{D}_Q$ est qualifié de compétitif lorsqu'il n'existe aucun autre devis $D' \in \mathcal{D}_Q$ ne proposant à la fois un coût monétaire et un temps de réponse inférieur à celui de D. Formellement, l'ensemble $\mathcal{D}_Q$ présenté à l'utilisateur respecte la condition énoncée en équation 3.2, où les prix m sont arrondis au centime supérieur et les temps de réponse r à la seconde supérieure.

$$\begin{aligned} \forall D \in \mathcal{D}_Q \nexists D' \in \mathcal{D}_Q : D \neq D' \\ \wedge \ \lfloor m_Q^{(D')} \rceil \leq \lfloor m_Q^{(D)} \rceil \ \wedge \ \lfloor r_Q^{(D')} \rceil < \lfloor r_Q^{(D)} \rceil \end{aligned} \tag{3.2}$$

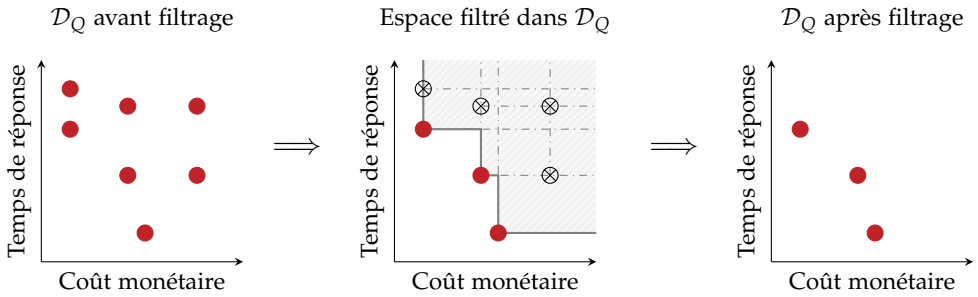


FIGURE 3.5 – Illustration graphique du filtrage des devis, représentés par des points rouges, n'étant pas compétitifs dans $\mathcal{D}_Q$.

Comme illustré par la figure 3.5, l'application de la condition aux différents devis crée un espace dans lequel les devis sont considérés inacceptables : ils sont donc supprimés de $\mathcal{D}_Q$ afin de ne pas les présenter aux utilisateurs.

2.6 *Synthèse*

Le middleware est constitué de six composants. L'analyseur, le schéma multi-cloud, l'optimiseur, le modèle de coûts et le calculateur de devis sont utilisés pour le calcul des devis. L'optimiseur est également utilisé pour orchestrer la sous-traitance des sous-requêtes auprès des fournisseurs au moment de l'exécution de la requête multi-cloud pour ré-optimiser la requête en fonction des résultats d'exécution. Il transmet également les valeurs constatées de cardinalité en sortie, de coûts monétaires et de temps de réponse au modèle de coûts pour que ce dernier puisse apprendre à partir ce celles-ci.

L'articulation entre ces composantes est dictée par la procédure suivie par les utilisateurs lorsqu'ils formulent leurs requêtes multi-clouds. Cette procédure est décrite plus en détails ci-après.

3 *Les devis dans Nebula*

Afin d'obtenir les résultats de sa requête multi-cloud, l'utilisateur suit la procédure illustrée par la figure 3.6. Il va d'abord soumettre sa requête au middleware afin d'obtenir un ou plusieurs devis pour celle-ci. S'il accepte l'un des devis, le middleware traitera la requête en orchestrant sa sous-traitance par les différents fournisseurs qu'elle implique, et retournera à l'utilisateur l'emplacement de ses résultats stockés sous la forme d'une relation hébergée par le service DBaaS de l'un des fournisseurs. Le middleware agissant au nom de l'utilisateur, ce dernier sera facturé par les fournisseurs pour les ressources utilisées lors du traitement de la requête multi-cloud. Cette procédure en deux étapes est justifiée par le fait que le middleware agit au nom de l'utilisateur ;

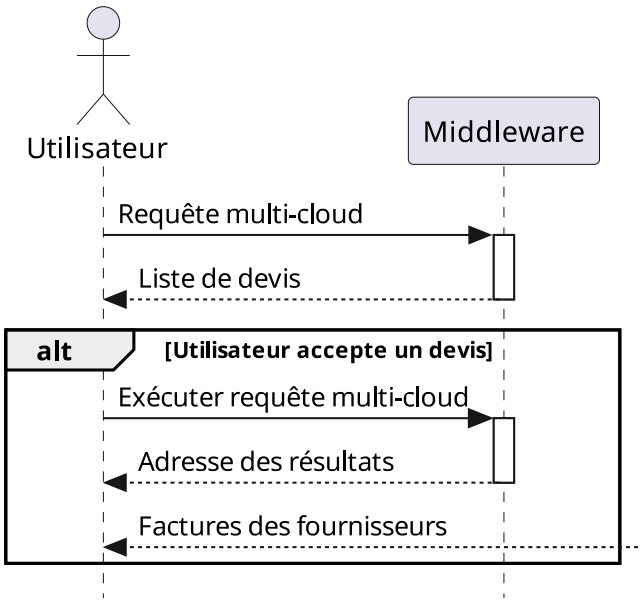


FIGURE 3.6 – Diagramme de séquence système illustrant les interactions entre l'utilisateur et le middleware pour demander le traitement d'une requête multi-cloud

celui-ci étant à envisager comme un outil permettant le traitement de requêtes multi-clouds plutôt que comme un service commercial. Le devis est donc le moyen par lequel l'utilisateur contrôle ses dépenses : si l'utilisateur n'est satisfait par aucun des devis proposés pour sa requête multi-cloud, il peut tout à fait refuser de procéder à son exécution. Les devis sont également le moyen par lequel le middleware guide l'exécution des requêtes multi-clouds afin de respecter le plus possible les souhaits de l'utilisateur.

Les devis pour requêtes multi-clouds sont inspirés des *Personalised Service-Level Agreement* (PSLA) que l'on retrouve dans la littérature sur les SGBD *multi-tenant*. Une présentation rapide de celle-ci est proposée ci-après afin de bien comprendre l'origine des devis dans Nebula et leur différence avec les PSLA.

3.1 *Des devis inspirés des SLA personnalisés*

Dans le domaine de la gestion de données dans le cloud, les PSLA sont basés sur l'idée qu'une requête, ou un lot de requête, devraient être tarifée en fonction de sa complexité et des exigences de performance des locataires. *Ceteris paribus*, les locataires paieraient donc moins pour les requêtes moins complexes, et les fournisseurs pourraient facturer plus cher une requête devant être exécutée plus rapidement.

L'idée de PSLA a été introduite pour la première fois pendant les travaux relatifs au déploiement dans le cloud du polystore Myria [2] [74]. Ceux-ci sont conçus pour aider les utilisateurs à choisir entre différents *tiers*, c'est-à-dire différentes configurations de déploiement dudit polystore au sein d'une plateforme IaaS. Ils associent une charge de travail, c'est-à-dire un lot de requêtes, à son coût monétaire final pour l'utilisateur et à une estimation de son temps de réponse. Ces SLA sont calculées pour une base de données et une charge de travail fournie par l'utilisateur. Un nombre donné de *tiers* est alors présenté aux utilisateurs de Myria, ayant chacun un prix et une estimation du temps d'exécution total de la charge de travail.

D'autres travaux ultérieurs se sont également concentrés sur le calcul de PSLA, cette fois afin de proposer un cadre de négociation entre fournisseurs et locataires permettant de fixer les modalités pour le traitement d'une charge de travail par le SGBD *multi-tenant* [15]. Ces charges de travail correspondent à des requêtes-type. Cette fois, les attentes des locataires sont formulées *a priori*, ceux-ci exposant leur souhait en matière de temps de réponse et leur tolérance vis-à-vis de ces attentes. Une fonction de prix est donc définie selon les attentes de chaque locataire. La base de la fonction de prix est déterminée par échantillonnage des différent intervalles de sélectivité des opérateurs de la requête. Cette méthode est particulièrement adaptée au calcul de SLA pour des requêtes récurrentes, comme celles formulées par les *Object-Relationnal Mappings* (ORMs) habituellement utilisés pour assurer l'interface entre une base de données relationnelle et un logiciel conçu selon une approche orientée objet.

De plus, l'objectif de cette technique est de proposer un cadre de négociation entre un fournisseur cloud et ses locataires pour atteindre un juste compromis sur le prix payé par le locataire, le profit du fournisseur et le SLO. Elle n'est donc pas réutilisable directement dans le cadre de notre middleware, ce dernier étant conçu pour traiter des requêtes *ad-hoc* et n'étant pas un service cloud de type *Analytics-as-a-Service*. Cependant, l'idée du seuil de tolérance des locataires et son utilisation par le fournisseur dans l'optimisation des requêtes est tout à fait applicable aux devis proposés aux utilisateurs.

3.2 *Utilisation des devis dans Nebula*

Nebula n'ayant pas une nature commerciale, offrir un SLA avec un SLO et des mesures de compensation aux utilisateurs en cas de violation de ces engagements n'est pas réaliste. En effet, le middleware n'a aucun contrôle sur les choix d'optimisation des sous-requêtes effectués par les fournisseurs sous-traitant. Nebula est donc à envisager comme une forme de courtier,

2. Ceux-ci s'inscrivent dans un projet beaucoup plus large de conception d'une plateforme d'analyse de données massives et hétérogènes ayant eu cours jusqu'en 2018 au sein de Université de Washington (Seattle, États-Unis). Voir `http://myria.cs.washington.edu/`.

travaillant bénévolement pour mettre en relation de manière transparente ses utilisateurs avec les fournisseurs. Pour une requête multi-cloud donnée Q, un ensemble de devis $\mathcal{D}_Q$ est présenté aux utilisateurs où chaque devis $D \in \mathcal{D}_Q$ est un tuple $<m_D, r_D>$ contenant une estimation du coût monétaire et du temps de réponse pour répondre à Q, calculé à partir d'un plan d'exécution multi-cloud. En choisissant l'un d'entre eux, l'utilisateur peut fixer un seuil de tolérance $\Lambda > 0$ par rapport à ces devis de sorte à exprimer plus finement ses attentes en matière de performances et de coût monétaire. Ce seuil est alors utilisé dans le processus d'optimisation dynamique pour donner ou non plus de marges de manœuvre au middleware dans ses choix de ré-optimisation.

Le middleware étant à but non-lucratif, pas un centime ne circule par celui-ci : les utilisateurs sont facturés directement par les fournisseurs. Dès lors, il est nécessaire que ces derniers aient confiance en Nebula. Nous proposons de gagner celle-ci en présentant l'historique des requêtes aux utilisateurs en toute transparence. L'utilisation d'un seuil de tolérance équitable, c'est-à-dire ni trop strict pour le middleware ni trop défavorable à l'utilisateur, par exemple à $\Lambda = 0,1$, permet par ailleurs aux utilisateurs d'avoir des attentes réalistes vis-à-vis de Nebula.

3.3 *Synthèse*

Les devis dans Nebula sont inspirés par les PSLA, en ommettant l'aspect commercial de ces derniers. Ils sont composés d'un coût monétaire et d'une estimation du temps de réponse de la requête multi-cloud. Plusieurs devis pareto-optimaux sont présentés aux utilisateurs. Ils permettent aux utilisateurs d'avoir des attentes réalistes quand aux performances et au coût monétaire de leurs requêtes, et au middleware de guider l'orchestration de la sous-traitance des requêtes en fonction de la volonté de ses utilisateurs.

4 *Conclusion*

La problématique du traitement des requêtes multi-clouds nous a conduit à concevoir un middleware à cet effet. Nommé Nebula, son architecture est centralisée et orchestre la sous-traitance des requêtes multi-clouds auprès des fournisseurs cloud les impliquant. Il agit comme un courtier entre ses utilisateurs et les fournisseurs, utilisant des devis comme outil de contrôle laissé aux utilisateurs sur leurs dépenses monétaires et leurs temps d'attentes dans le processus d'optimisation des requêtes. S'il décide qu'un devis lui convient, alors l'optimiseur de Nebula va procéder au traitement de la requête en l'optimisant dynamiquement par rapport à ce devis. Celui-ci va orchestrer la sous-traitance des sous-requêtes auprès des fournisseurs ainsi que les transferts de données inter-fournisseurs. L'optimiseur va également

transmettre les informations progressivement récoltées sur les résultats d'exécution au modèle de coûts afin que celui-ci puisse apprendre de cette vérité terrain fraîchement récoltée.

Les plans d'exécution multi-clouds sont générés par l'optimiseur en suivant une stratégie de recherche, utilisant des estimations sur les sous-requêtes pour guider son exploration en direction. Deux stratégies de recherche ont été proposées pendant cette thèse, et sont présentées dans le chapitre ci-après.

4

Générer des plans d'exécution multi-clouds économiques et performants

1 *Introduction*

Afin de pouvoir calculer des devis et permettre l'exécution des requêtes multi-clouds, Nebula doit construire des plans d'exécution multi-clouds. Cette construction est réalisée par son optimiseur, doit donc résoudre le problème d'optimisation de requêtes multi-clouds. Ce dernier est une extension du problème d'optimisation de requêtes distribuées, en prenant en compte les particularités des services DBaaS utilisés pour sous-traiter l'exécution des sous-requêtes. En cherchant à trouver des plans d'exécution à la fois économiques et performants, l'optimiseur fait en réalité de l'optimisation multicritères de requêtes [28, 30, 31, 75]. L'optimiseur explore l'espace de recherche en suivant une stratégie de recherche. Deux stratégies ont été proposées pendant cette thèse.

La première est une stratégie exhaustive pour explorer l'espace des plans d'exécution multi-clouds possibles pour une requête donnée et son utilisation pour le calcul de devis. Le support de cette énumération est un Graphe

orienté acyclique (DAG) modélisant les dépendances entre sous-requêtes possibles. Chacun de ces plans est chiffré en utilisant les estimations sur ses résultats intermédiaires produits par les fournisseurs participant à la requête. Nous montrons que cette stratégie de recherche a une complexité algorithmique factorielle : elle permet donc de trouver des plans d'exécution optimaux par rapport aux estimations des fournisseurs pour des requêtes faisant intervenir peu de fournisseurs mais expose le middleware à un risque d'explosion combinatoire.

Cette méthode n'est donc pas pertinente dans le cas de requêtes impliquant plus que quatre fournisseurs en un temps acceptable. Une seconde méthode d'exploration de l'espace de recherche aléatoire et itérative a donc été proposée. En explorant progressivement l'espace de recherche en transformant petit à petit un plan d'exécution généré aléatoirement en un plan d'exécution optimal, au moins localement, elle permet de diminuer le nombre de plans d'exécution considérés tout en augmentant la variété de ces derniers, en trouvant des plans d'exécution *bushy* et déséquilibrés.

Ce chapitre est structuré comme suit. La section 2 présente la méthode exhaustive et la section 3 présente la méthode aléatoire itérative. Il est conclut par la section 4.

2 *Exploration exhaustive des plans d'exécution multi-clouds*

La méthode d'exploration exhaustive de l'espace de recherche $\mathcal{T}_Q$ proposée pour Nebula consiste à évaluer l'ensemble des plans d'exécution multi-clouds $\mathcal{T}'_Q \subset \mathcal{T}_Q$ tels que $T_Q \in \mathcal{T}'_Q$ est linéaire, c'est-à-dire que chaque sous-requête $q_i \in T_Q$ est soit une sous-requête mono-fournisseur, soit une sous-requête multi-fournisseurs ayant comme prédécesseurs deux sous-requêtes mono-fournisseur ou une sous-requête mono-fournisseur et une sous-requête multi-fournisseurs. L'exploration se limite à ces derniers afin de circonscrire le nombre de plans considérés tout en respectant la philosophie de la recherche exhaustive.

L'exploration exhaustive des plans d'exécution linéaires pour une requête Q suit une procédure en trois parties. Premièrement, l'ensemble des clauses C_Q de Q retourné par l'analyseur est transformé en un DAG $G_Q = <V, E>$, où chaque sommet $v \in V$ est une sous-requête q_i et chaque arc $e = <v_0, v_1> \in E$ est une dépendance entre les deux sous-requêtes modélisées par v_0 et v_1. Ensuite, le SGBD du fournisseur estime la cardinalité en sortie $|q_i|$, le coût monétaire m_{q_i} et le temps de réponse r_{q_i} des sous-requêtes qui lui sont affectées ; cette dernière information est ensuite corrigée par le modèle de coûts multi-cloud et utilisée pour calculer les éventuels transferts de données entre fournisseurs. Enfin, les devis possibles $\mathcal{D}_Q$ sont identifiés à partir de $\mathcal{T}'_Q$,

puis réévalués à leur tour par le modèle de coûts. Chaque $D \in \mathcal{D}_Q$ a donc un homologue $T_Q \in \mathcal{T}'_Q$ qui servira d'entrée à l'optimiseur dynamique dans le cas où l'utilisateur souhaite procéder à l'exécution de Q étant donné D_Q.

2.1 *Décomposition des requêtes multi-clouds en un graphe de sous-requêtes*

La méthode exhaustive d'exploration de l'espace de recherche adapte à la fois la technique historique de la décomposition de requêtes [73] proposée pour les SGBDR en environnement mono-processeur, ainsi que des travaux plus récents sur l'optimisation multicritères appliquée à de l'optimisation de requêtes en environnement cloud IaaS [28].

La première étape de la stratégie de recherche exhaustive consiste à construire un DAG $G_Q = <V, E>$, dans lequel les sommets $v_i \in V$ sont des tuples $<P^{(q_i)}, \mathcal{P}_{q_i}, C_{q_i}>$ contenant, pour une sous-requête q_i, l'ensemble des fournisseurs qu'elle implique $\mathcal{P}_{q_i}$, le fournisseur auquel elle est affectée $P^{(q_i)}$ et l'ensemble des clauses qu'elle contient C_{q_i}. Les arcs $e = <v_i, v_j>$ représentent les dépendances entre les sous-requêtes (c-à-d que la requête q_i ne peut pas être exécutée tant que q_j ne l'a pas été). Celui-ci permet de modéliser l'ensemble des plans d'exécution linéaires pour la requête Q : les sommets à degré sortant nul ($d^+(v) = 0$, assimilables aux puits d'un réseau de flot) représentent les sous-requêtes permettant de terminer l'exécution de la requête et les anti-arbres [1] enracinés dans ces derniers ayant un degré entrant nul pour toutes leurs feuilles ($d^-(v) = 0$, assimilables aux sources d'un réseau de flot) sont chacun le squelette d'un plan d'exécution multi-cloud linéaire.

La procédure de construction de G_Q est illustrée par l'algorithme 4.1, où chaque $c_i = <t_i, \varphi_i, A_i>$ est une clause telle que définie précédemment dans la section 2.2 du chapitre 3, $\mathcal{P}_{c_i}$ l'ensemble des fournisseurs impliqués par c_i et $P^{(a)}$ le fournisseur hébergeant un attribut $a \in A_i$. $Perm(k, X)$ note les k-permutations, c'est-à-dire l'ensemble des permutations sans répétitions de taille k des éléments d'un ensemble X [2]. L'algorithme construit progressivement les couches du DAG en (i) regroupant les clauses par l'ensemble des fournisseurs stockant les données qu'elles manipulent, (ii) ajoutant des projections pour les attributs qui seront utilisés par les requêtes suivantes, (iii) transformant ces groupes de clauses en requêtes associées à un sommet du DAG, et (iv) en reliant ces dernières entre-elles. La figure 4.1 illustre le graphe G_Q obtenu à partir de l'ensemble de clauses de l'équation 3.1 extraites de la requête en exemple fil-rouge (cf. fig. 3.1b). Les requêtes associées aux sommets v_1 et v_2 peuvent être qualifiées de sous-requêtes mono-fournisseur,

1. Un anti-arbre est un arbre dans lequel les arcs pointent vers la racine. Par exemple, le graphe orienté $a \rightarrow b$ est à la fois un arbre enraciné en a et un anti-arbre enraciné dans b.

2. Par exemple, l'ensemble $\{a, b, c\}$ a comme 2-permutations l'ensemble des couples $\{<a, b>, <a, c>, <b, a>, <b, c>, <c, a>, <c, b>\}$.

Algorithme 4.1 : Génération du graphe de requête

Entrées : C_Q un ensemble de clauses.
Sorties : Un graphe orienté acyclique $G_Q = <V, E>$

1 $V, E \leftarrow \emptyset$;
2 Soit M une application entre un ensemble de fournisseurs $K = \{\mathcal{P}_c \mid c \in C_Q\}$ et un ensemble de clauses telle que $M : k \rightarrow \{c \mid c \in C_Q, \mathcal{P}_c = k\}$;
3 **pour chaque** $k \in K$ *par ordre croissant de* $|k|$ **faire**
    `/* Ajout des projections à chaque sous-requête */`
4 $C^{(k)} \leftarrow M(k) \cup \{<\Pi, 'a', \{a\}> \mid a \in A_i, \forall c_i \in M(x)\ \forall x \in K\ :\ k \subset x\ \wedge\ P^{(a)} \in k\}$;
    `/* Itération les permutations de k pour étendre G_Q */`
5 **pour chaque** $k' \in Perm(|k|, k)$ **faire**
6 $v \leftarrow <k'_0, k', C^{(k)}>$;
7 **si** $|k| > 1$ **alors**
            `/* Raccord avec les prédécesseurs de v */`
8 $B \leftarrow \bigcup_{w \in \{k'_0, k'_{|k|}\}} <w, w'>$ with $w' \in V \wedge w'.\mathcal{P} = k - w$;
9 **pour chaque** $<a, b> \in B$ **faire**
10 $v' \leftarrow v$;
11 Ajouter v' à V ;
12 Ajouter $<a, v'>$ and $<b, v'>$ à E;
13 **sinon**
14 Ajouter v à V ;
15 $G_Q \leftarrow <V, E>$;

n'interrogeant des relations issues d'un seul fournisseur, et celles associées à v_3 et v_4 peuvent être qualifiées de sous-requêtes multi-fournisseurs, interrogeant des relations issues de plusieurs fournisseurs.

2.2 *Estimation du coût monétaire et des performances des composantes du graphe*

L'étape suivante dans le processus d'optimisation est l'estimation du coût monétaire et des performances des sous-requêtes et des transferts de données inter-fournisseurs. Le code SQL des sous-requêtes q_i est généré à partir de $C^{(v_i)} : v_i \in V$, puis les estimations de la cardinalité des résultats, du coût monétaire et du temps de réponse de la requête, notés $<s_{q_i}, m_{q_i}, r_{q_i}>$, sont demandés au fournisseur associé $P^{(v_i)}$.

Les transferts de données sont ensuite évalués à l'aide de l'estimation de la cardinalité en sortie. Chaque arc $e = <v_i, v_j> \in E$ modélise un transfert

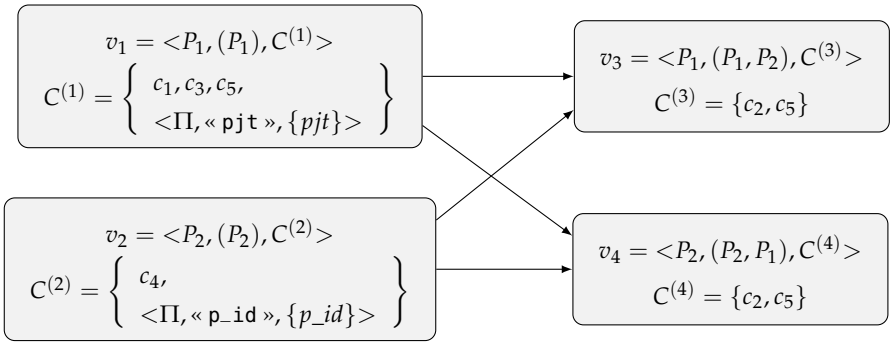


FIGURE 4.1 – Graphe de requête G_Q dérivé de la requête multi-cloud Q de l'exemple fil-rouge.

de données, avec v_i associé à la sous-requête source q_i et v_j associé à la sous-requête cible q_j. Un tuple $<m_e, r_e>$ est calculé pour chaque arc, en suivant les formules décrites par les équations 4.1 et 4.2, avec $\varepsilon^{(E)}_{P_v}$ le coefficient de facturation (en euros par gigaoctets) pour l'export des données de P_v (associé à un sommet v), $\varepsilon^{(S)}_{P_v}$ le coefficient de facturation pour le stockage des données (en euros par gigaoctets) et b_P la bande passante d'un fournisseur P.

$$m_e = \begin{cases} s_{q_i} \times \varepsilon^{(S)}_{P^{(v_i)}} & \text{si } P^{(v_i)} = P^{(v_j)} \\ s_{q_i} \times (\varepsilon^{(E)}_{P^{(v_i)}} + \varepsilon^{(S)}_{P^{(v_j)}}) & \text{sinon} \end{cases} \tag{4.1}$$

$$r_e = \begin{cases} 0, & \text{si } P^{(v_i)} = P^{(v_j)} \\ s_{q_i} \times min(b_{P^{(v_i)}}, b_{P^{(v_j)}}) & \text{sinon} \end{cases} \tag{4.2}$$

La complexité algorithmique de cette étape de calcul de devis découle de l'ordre de G_Q (c-à-d le nombre de sommets $|V|$ de celui-ci), calculable en suivant la proposition 1. La complexité, exprimée dans le corollaire 1.2, est factorielle en le nombre de fournisseurs impliqués dans la requête multi-cloud, notée $\mathcal{O}(|\mathcal{P}_Q|!)$. Ce constat vient du fait que l'ordre du graphe G_Q est de l'ordre de $|\mathcal{P}_Q|$, tout comme sa taille (c-à-d le nombre de ses arcs $|E|$), calculable grâce au corollaire 1.1. Il existe donc un risque d'explosion combinatoire en utilisant la méthode exhaustive, comme illustré par la figure 4.2.

Proposition 1 (Ordre de G_Q)**.** *L'ordre $|V|$ du graphe de requêtes est la somme du nombre de k-permutations de $\mathcal{P}_Q$ de chaque couche.*

$$|V| = \sum_{k=1}^{|\mathcal{P}_Q|} \frac{|\mathcal{P}_Q|!}{(|\mathcal{P}_Q| - k)!}$$

Démonstration. L'algorithme 4.1 construit G_Q par couches successives. Chaque couche étant une k-permutation de l'ensemble des fournisseurs $\mathcal{P}_Q$, le nombre

de sommets d'une couche est donc le nombre d'éléments de ladite permutation [76, p. 18], soit $\frac{|\mathcal{P}_Q|!}{(|\mathcal{P}_Q|-k)!}$. Le graphe comporte $|\mathcal{P}_Q|$ couches, la première étant les 1-permutations de $\mathcal{P}_Q$ et la dernière étant les $|\mathcal{P}_Q|$-permutations de $\mathcal{P}_Q$. L'ordre $|V|$ de G_Q est donc la somme de la taille de ces k-permutations, pour $k \in [\![1..|\mathcal{P}_Q|]\!]$ CQFD

Corollaire 1.1 (Taille de G_Q)**.** *La taille $|E|$ du graphe de requêtes est le double de la somme du nombre de k-permutations de $\mathcal{P}_Q$ de chaque couche dans laquelle les sommets ont des prédécesseurs.*

$$|E| = 2 \times (|V| - |\mathcal{P}_Q|)$$

Démonstration. L'algorithme 4.1 construit G_Q par couches successives, et relie chaque sommet à deux prédécesseurs. Or, la première couche, c'est-à-dire celle des 1-permutations de $\mathcal{P}_Q$ de ayant trivialement $|\mathcal{P}_Q|$ éléments, ne peut avoir de prédécesseurs. CQFD

Corollaire 1.2 (Complexité de l'estimation de G_Q)**.** *Le nombre d'opérations o nécessaires pour mener à bien l'estimation de G_Q est proportionnel à la factorielle du nombre de fournisseurs impliqués par Q.*

$$o \propto |P_Q|! \implies \mathcal{O}(|P_Q|!)$$

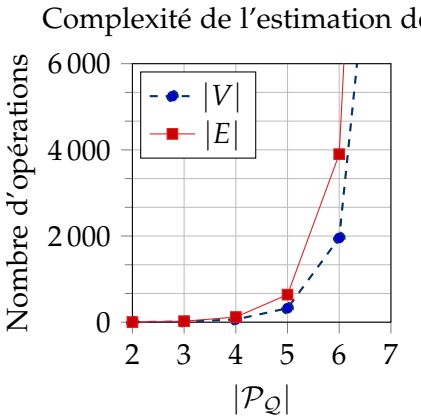
Démonstration. Le nombre d'opérations à effectuer pour estimer G_Q est $a|V| + b|E|$, avec $a \in \mathbb{N}$ un nombre défini d'opérations à effectuer pour estimer chaque sous-requête et $b \in \mathbb{N}$ un nombre défini d'opérations à effectuer pour estimer chaque transfert de résultats intermédiaires.

$$\begin{aligned}
\mathcal{O}(a|V| + b|E|) &= \mathcal{O}\left(\cancel{a}|V| + \cancel{2b}(|V| - \cancel{|\mathcal{P}_Q|})\right) \\
&= \mathcal{O}(\cancel{2}|V|) \\
&= \mathcal{O}\left(\sum_{k=1}^{|\mathcal{P}_Q|} \frac{|\mathcal{P}_Q|!}{(|\mathcal{P}_Q| - k)!}\right) \\
&= \mathcal{O}\left(\cancel{\frac{|\mathcal{P}_Q|!}{(|\mathcal{P}_Q| - 1)!}} + \cancel{\frac{|\mathcal{P}_Q|!}{(|\mathcal{P}_Q| - 2)!}} + \ldots + \cancel{\frac{|\mathcal{P}_Q|!}{(|\mathcal{P}_Q| - |\mathcal{P}_Q|)!}}\right) \\
&= \mathcal{O}(|\mathcal{P}_Q|!)
\end{aligned}$$

CQFD

2.3 *Calcul et identification des devis*

Après avoir estimé les composantes de G_Q, l'ensemble des plans d'exécution $\mathcal{T}_Q$ permettant de répondre à Q est extrait de G_Q. Comme illustré par la figure 4.3 reprenant l'exemple fil-rouge, ceux-ci sont des anti-arbres



$\lvert\mathcal{P}_Q\rvert$	2	3	4	5	6
$\lvert V\rvert$	4	15	64	325	1956
$\lvert E\rvert$	4	24	120	640	3900

FIGURE 4.2 – Illustration de la complexité algorithmique de la méthode exhaustive

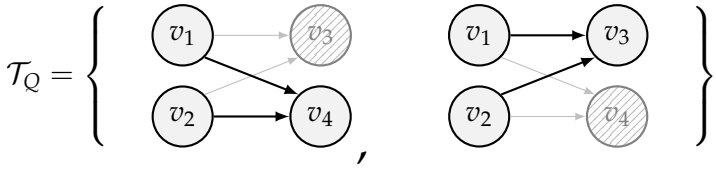


FIGURE 4.3 – Ensemble $\mathcal{T}_Q$ des plans d'exécution extraits depuis G_Q

$T_i = <V^{(T_i)}, E^{(T_i)}>$ extraits de G_Q. Chacun est enraciné dans un sommet $v_i \in V$ tel que $d^+(v_i) = 0$, où $V^{(T_i)} \subseteq$ tel que $V^{(T_i)} = \mathcal{N}_{T_i}^+$ est l'ensemble des prédécesseurs de T_i dans G_Q et où $E^{(T_i)} \subseteq E$ est l'ensemble des arcs les reliant. Les feuilles de T_i ont un degré entrant nul.

Proposition 2 (Cardinalité de $\mathcal{T}_Q$)**.** *La cardinalité de $\mathcal{T}_Q$ est le nombre de sommets dans G_Q tel que, pour $v \in V$, $d^+(v) = 0$, c'est-à-dire :*

$$|\mathcal{T}_Q| = |\mathcal{P}_Q|!$$

Le coût monétaire m_{T_i} et le temps de réponse r_{T_i}de chaque plan d'exécution $T_i \in \mathcal{T}_Q$ est ensuite calculé ; m_{T_i} est la somme du coût monétaire associé à chaque arc et sommet de T_i et du coût monétaire du stockage des résultats finaux produits par la racine t (cf. éq. 4.3) ; r_{T_i} est la somme des temps de réponse de chaque arc et sommet situé sur le plus long chemin entre la source et les feuilles de T_i.

$$m_{T_i} = \sum_{v \in V^{T_i}} m_v + \sum_{e \in E^{T_i}} m_e + s_t \varepsilon_{P(t)}^{(S)} \tag{4.3}$$

Le contenu de $\mathcal{T}_Q$ correspond au paramètre $\mathcal{E}_Q$ du calculateur de devis décrit précédemment (cf. ch. 3 sect. 2.5), qui va appliquer ses règles de filtrage de sorte à ne présenter à l'utilisateur de Nebula que des devis compétitifs. Comme indiqué dans la proposition 2, le nombre de plans d'exécution

multi-clouds extraits du graphe de requête est la factorielle du nombre de fournisseurs impliqués par la requête. Ceux-ci n'étant vraisemblablement pas tous sur le front de Pareto coût monétaire - temps de réponse, ce filtrage est donc nécessaire afin de ne pas noyer l'utilisateur de devis inutiles.

2.4 *Synthèse*

La méthode d'exploration exhaustive de l'ensemble des plans d'exécution multi-clouds linéaires repose sur l'utilisation d'un DAG, modélisant les sous-requêtes et leurs interdépendances. Ces sous-requêtes sont utilisées pour demander des estimations de leur cardinalité en sortie, de leur coût monétaire et de leur temps de réponse aux fournisseurs auxquelles elles sont associées. Ces estimations permettent à leur tour d'estimer le coût monétaire et le temps de réponse des plans d'exécution multi-clouds, qui seront transmis au calculateur de devis pour réévaluation et filtrage. La complexité algorithmique de cette méthode est factorielle en le nombre de fournisseurs impliqués par la requête : si elle permet de trouver des plans d'exécution de bonne qualité pour des requêtes multi-clouds peu complexes, elle expose aussi le middleware à un risque majeur d'explosion combinatoire lorsque les requêtes multi-clouds se complexifient.

3 *Une méthode d'exploration aléatoire et itérative de l'espace de recherche*

C'est précisément pour permettre le traitement de requêtes complexes que nous proposons une méthode d'exploration aléatoire, qui ne subit pas l'explosion combinatoire tout en explorant une plus grande variété de plans d'exécution. Cette méthode est inspirée des méthodes d'amélioration itératives, ayant déjà fait leurs preuves dans les SGBDR [77]. Le choix de la méthode s'est porté sur l'amélioration itérative plutôt que sur le recuit simulé (*simulated annealing*) [78] pour sa simplicité relative, dans la mesure où il est difficile de déterminer si une méthode est meilleure que l'autre en termes de performances et d'optimalité des solutions [79, ch. IV, p. 159], [3]. La méthode que nous proposons part d'un plan d'exécution construit aléatoirement à partir des clauses de la requête multi-cloud, et explore de proche en proche son voisinage, en suivant des règles de génération des plans d'exécution voisins, jusqu'à découvrir un minimum local. Ce minimum local peut être dépassé pour tenter de trouver un nouveau gradient en direction d'un autre minimum local.

3. Le recuit simulé ne devrait cependant pas être écarté, et il serait intéressant de poursuivre la controverse sur la supériorité de l'une ou l'autre méthode pour l'optimisation de requêtes multi-clouds.

Nous présentons par la suite la méthode de construction du plan d'exécution initial, puis présentons les règles de génération du voisinage et enfin l'algorithme que suit la méthode itérative pour énumérer les plans d'exécution.

3.1 *Construction des plans d'exécution multi-clouds*

La première étape du processus d'exploration itérative de l'espace de recherche est la construction d'un premier plan d'exécution multi-cloud T_0, servant de point de départ à l'exploration de proche en proche des plans d'exécution voisins en direction d'un optimum local.

La construction de T_0 par transformation successives de l'ensemble des clauses C_Q extrait de la requête multi-cloud Q par l'analyseur (cf. ch. 3 sect. 2.2). L'approche suivie est mono-phase, c'est-à-dire que la composition des sous-requêtes et leur placement chez les fournisseurs est fait en même temps. Les dépendances entre les clauses de C_Q sont représentées par un graphe $L_Q = <V, E>$, où les sommets $v_i \in V$ ont comme homologues une clause $c_i \in C_Q$, et où les arcs $e \in E$ (avec $e = <s, t>$) représentent une dépendance entre les clauses. E est défini formellement dans l'équation 4.4, avec $\mathcal{R}_v$ l'ensemble des relations impliquée par la clause associée à v.

$$E = \left\{ <s, t> \mid s, v \in V \ : \ s \neq v \wedge |\mathcal{R}_s| \leq |\mathcal{R}_v| \wedge \mathcal{R}_s \cap \mathcal{R}_v \neq \emptyset \right\} \tag{4.4}$$

Toutes les composantes connexes de L_Q dont les clauses associées aux sommets impliquant le même ensemble de fournisseurs sont fusionnées afin de construire le graphe L'_Q. Ses sommets sont cette-fois tous associés à un ensemble de clauses C_v satisfaisant la définition de l'équation 4.5. Ils peuvent être vus comme les prédicats de sélection d'une sous-requête mono-fournisseur.

$$\forall c \in C_v \nexists c' \in C_v : \bigcup_{R \in \mathcal{R}_c} P_R \neq \bigcup_{R \in \mathcal{R}_{c'}} P_R \tag{4.5}$$

Un arbre couvrant orienté est ensuite extrait aléatoirement de L'_Q : il s'agit du squelette de T_0 qui représente un plan d'exécution multi-cloud. Celui-ci est complété des projections nécessaires à la génération de sous-requêtes correctes permettant de réaliser la requête multi-cloud Q et par l'affectation aléatoire de chacune des sous-requêtes à l'un des fournisseurs impliqué par ses clauses. Ces transformations sont illustrées par la figure 4.4, qui montre L_Q, L'_Q et T_0, étant en fait le « squelette » du plan d'exécution de la figure 3.4, construits à partir de C_Q dérivé de la requête Q en fil-rouge[4] (cf. fig. 3.1b

4. Pour rappel, la requête est formulée sur un schéma multi-cloud contenant les relations `Emp(eId, nom_e, #dpt, #pjt)` et `Dept(d_id, lieu)` hébergées chez le fournisseur P_1 et `Projet(p_id, sujet)` chez P_2. Elle retourne le nom et la localisation des employés travaillant sur un projet en rapport avec le multi-cloud à Toulouse et à Linz.

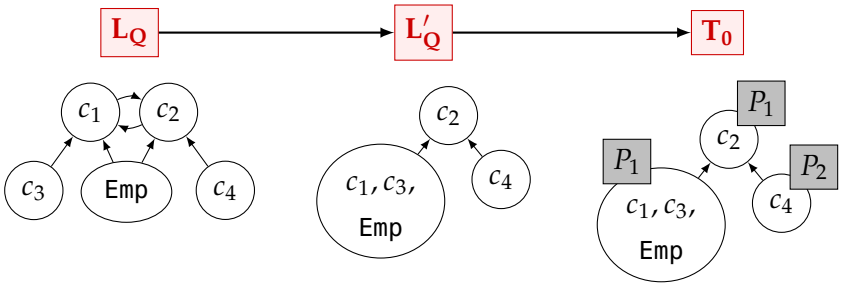


FIGURE 4.4 – Construction d'un plan d'exécution T_0 à partir d'un graphe L_Q modélisant les dépendances entre les clauses C_Q de la requête multi-cloud Q (cf. fig. 3.1b).

puis éq. 3.1). Ce plan d'exécution généré aléatoirement T_0 n'est certainement pas optimal, que ce soit vis-à-vis de son temps de réponse ou de son coût monétaire : il convient de rappeler qu'il n'est là que pour servir de point de départ au processus d'optimisation. L'exploration de l'espace de recherche commence donc à partir de T_0 et va explorer les plans d'exécution possibles de voisinage en voisinage. Les règles permettant de générer les voisins d'un plan sont décrites ci-après.

La première règle ϱ_1 permet d'explorer différents placements des sous-requêtes sur les fournisseurs qu'elles impliquent $\mathcal{P}_v = \bigcup_{c \in C_v} \bigcup_{R \in \mathcal{R}_c} P_R$. Pour chaque sommet $v \in V$ dont la sous-requête associée est placée chez un fournisseur $P_v \in \mathcal{P}_v$, un nouveau plan est généré dans dans lequel P_v est substitué par un autre fournisseur pour chaque $P_{v'} \in \mathcal{P}_v - \{P_v\}$. Il est attendu que des plans d'exécution multi-clouds avec de moindres transferts de données inter-fournisseurs soient générés par cette règle, permettant ainsi de diminuer son coût monétaire et son temps de réponse tout en préservant la distribution du traitement des sous-requêtes.

La seconde règle ϱ_2 revient à considérer un ordre des jointures différent. Pour chaque couple de sommets successifs $<v_1, v_2>$ associés à des sous-requêtes multi-fournisseurs tels que définis par l'équation 4.6, un nouveau plan d'exécution est généré par permutation de v_1 avec v_2.

$$\{<v_1, v_2> \mid v_1, v_2 \in V \ : \ \exists <v_1, v_2> \in E \wedge |\mathcal{P}_{v_1}| \geq 2 \leq |\mathcal{P}_{v_2}|\} \tag{4.6}$$

Cette règle permet de découvrir des plans d'exécution multi-clouds dans lesquels la somme de la taille de tous les résultats intermédiaires se retrouverait diminuée en faisant descendre dans le plan les sous-requêtes à forte sélectivité, diminuant de fait au moins son coût monétaire et, selon les sous-requêtes, son temps de réponse.

La troisième règle ϱ_3 consiste à chercher un compromis entre la distribution des sous-requêtes et leur localité en fusionnant des sous-requêtes successives affectées au même fournisseur. Formellement, pour chaque couple de sommets successifs $<v_1, v_2>$ placés sur le même fournisseur tel que définis par

l'équation 4.7, un nouveau plan d'exécution est construit en fusionnant v_1 et v_2 en v', donc l'ensemble des clauses est $C_{v'} = C_{v_1} \cup C_{v_2}$.

$$\{<v_1, v_2> \mid v_1, v_2 \in V \ : \ \exists <v_1, v_2> \in E \wedge P_{v_1} = P_{v_2}\} \tag{4.7}$$

Cette règle permet de découvrir des plans d'exécution multi-clouds stockant moins de résultats intermédiaires, permettant de faire des économies sur les coûts monétaires de stockage et surtout sur les coûts d'interrogation. Remarquons également que cette règle conduit à explorer des plans d'exécution multi-clouds qui ne sont pas linéaires, car les prédécesseurs de v_1 et de v_2 deviennent tous les prédécesseurs de v' : en sacrifiant une partie du potentiel de distribution du plan, on le renforce en explorant de nouveaux plans avec des *pipelines* potentiellement plus longs.

3.2 *Exploration itérative de l'espace de recherche*

La méthode itérative consiste à partir d'un plan d'exécution généré aléatoirement et à naviguer récursivement dans son voisinage de minimum local en minimum local. Il est possible de permettre à l'exploration de continuer à chercher au-delà du minimum local de l'espace déjà exploré afin de peut-être trouver un nouveau minimum local qui serait plus optimal que le précédent. La recherche d'un minimum global étant une tâche NP-difficile [32] dans le contexte de l'optimisation des plans d'exécution dans les SGBD, il est nécessaire de borner le nombre de plans d'exécution évalués lors de l'exploration itérative de l'espace de recherche de sorte à éviter de retomber dans l'explosion combinatoire engendrée par la méthode exhaustive présentée précédemment [5].

Deux type d'itération sont proposée. La première intervient lorsqu'un plan d'exécution T_j situé dans le voisinage de T_i est plus optimal que ce dernier par rapport à une fonction objectif ω, c'est-à-dire que T_j est l'optimum local de l'espace de recherche déjà exploré. Dans ce cas, T_j remplace T_i comme curseur dans la frontière de l'espace exploré $\mathcal{F}$. Le nombre de sauts de curseur est borné par une constante n. Le deuxième genre d'itération se produit lorsque le curseur T_i est l'optimum local de l'espace exploré et qu'aucun plan dans la frontière n'est plus optimal que lui. Dans ce cas, la frontière est repoussée et deviens désormais le voisinage de tous les plans d'exécution situés dans $\mathcal{F}$. Le nombres de fois que l'on s'autorise à repousser la frontière est borné par une constante p.

La procédure est formalisée dans l'algorithme 4.2. Celui-ci prends en entrée un plan T, les paramètres n et p ainsi qu'un ensemble de fonctions objectif Ω, et retourne une application R associant à chaque fonction $\omega \in \Omega$ in

5. Plus précisément, la complexité de la méthode itérative sans bornes serait en fait exponentielle, donc pire que celle de la méthode exhaustive se restreignant aux plans d'exécution linéaires.

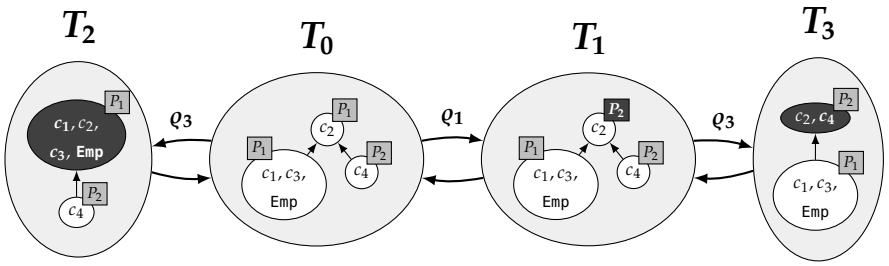


FIGURE 4.5 – Espace de recherche exploré à partir d'un plan T_0 suivant l'exemple de la figure 4.4. Les changements effectués par les règles de génération de voisins ϱ_i sont assombris.

plan d'exécution optimal par rapport à celle-ci $\hat{T}_\omega$. Pendant le déroulement de l'algorithme, ces plans d'exécution sont en fait les curseurs mentionnés précédemment. L'espace de recherche est représenté sous la forme d'un graphe orienté $S = <V, E>$ dont les sommets sont des plans d'exécution qui dénotent la relation de voisinage entre ces derniers selon les règles ϱ_1, ϱ_2 et ϱ_3. La frontière $\mathcal{F}$ est un ensemble de plans d'exécution. Lorsque l'algorithme cherche un optimum local dans le voisinage des curseurs, $\mathcal{F}$ contient l'union de l'ensemble des plans voisins $\mathcal{N}_t$ des meilleurs plans t par rapport à chaque $\omega \in \Omega$ n'ayant pas encore été ajouté au graphe de l'espace de recherche S. Lorsque l'algorithme cherche un plan d'exécution en repoussant la frontière, celle-ci contient l'ensemble des voisins des plans de V n'étant pas encore dans V. L'algorithme choisit ensuite le meilleur plan dans $\mathcal{F} \cup V$ pour chaque fonction objectif afin de mettre à jour l'application R. Une fois que les bornes n et p en paramètre de l'algorithme ont été atteintes, R est retourné.

La procédure d'exploration itérative est illustrée par la figure 4.5, avec $n = 2$ et $p = 0$. T_0 est le plan d'exécution initial, généré aléatoirement. À la première étape de l'exploration, T_1 est généré en suivant la règle ϱ_1 et T_2 en suivant ϱ_3. En supposant que T_1 était un minimum local, T_3 est généré à la seconde étape en suivant ϱ_3. En supposant que T_1 reste l'optimum local, l'exploration s'arrête car n et p ont été atteints et T_1 est retourné car c'est le plan le plus optimal qui ait été trouvé.

Les fonctions objectif ω utilisées dans Nebula sont décrites ci-après. Elles prennent toutes en paramètre un plan d'exécution t, l'objectif que la stratégie de recherche itérative cherche à remplir est $\min_{t \in \mathcal{T}_Q} \omega(t)$. Trois fonctions basiques sont définies pour la phase de calcul des devis : $\omega^{(r)}$ retourne le temps de réponse de t, $\omega^{(m)}$ son coût monétaire et $\omega^{(r/m)}$ le rapport entre son temps de réponse et son coût monétaire. Pour la phase de ré-optimisation, présentée ultérieurement dans la section 3 du chapitre 5, deux fonctions sont proposées. Leur formulation est fidèle à l'esprit de budget à respecter introduit par le devis. Avec un devis $<r_Q, m_Q>$, où r_Q est un temps de réponse et m_Q un coût monétaire, augmenté d'un seuil de tolérance vis-à-vis

Algorithme 4.2 : Exploration itérative de l'espace de recherche

Entrées : T un plan d'exécution déjà chiffré, Ω un ensemble de fonctions objectif, n un nombre maximal d'amélorations successives, p un nombre maximal d'extensions de la frontière de l'espace de recherche

Sorties : Une application R associant une fonction objectif $\omega \in \Omega$ à un plan d'exécution $\hat{T_\omega}$ minimisé selon ω

1 Soit $S = <V \leftarrow \{T\}, E \leftarrow \emptyset>$ un graphe orienté ;
2 $R : \Omega \mapsto \hat{\mathcal{T}}, \omega \rightarrow \arg\min_{v \in V} \omega(v)$;
3 $i \leftarrow 1$; $j \leftarrow 1$;
`// Frontière de l'espace de recherche`
4 $\mathcal{F} \leftarrow \{T\}$;
5 **répéter**
6 | **répéter**
| | `// Sauvegarde de l'historique`
7 | | $R' \leftarrow \{R(\omega) \mid \omega \in \Omega\}$; $\mathcal{F}' \leftarrow \mathcal{F}$;
| | `// Agrandissement de l'espace de recherche`
8 | | Soit $V' = \{t' \mid t' \in \mathcal{N}_t, t \in \mathcal{F}\} \setminus V$ l'ensemble des plans d'exécution voisins à la limite de l'espace de recherche ;
9 | | $E \leftarrow E \cup \{<v, v'> \mid v' \in V',\ v \in V : v' \in \mathcal{N}_v \setminus V\}$;
10 | | $V \leftarrow V \cup V'$;
11 | | **pour chaque** $v \in V'$ **faire** Calcul des coûts du plan v;
| | `// Recherche d'un changement dans l'ensemble des meilleurs plans rencontrés`
12 | | **si** $R' \neq \{R(\omega) \mid \omega \in \Omega\}$ **alors** $j \leftarrow j + 1$;
| | `// Définition de la nouvelle frontière`
13 | | $\mathcal{F} \leftarrow \{R(\omega) \mid \omega \in \Omega\}$;
14 | **jusqu'à** $j = p \vee \mathcal{F} = \mathcal{F}'$;
15 | $i \leftarrow i + 1$;
| `// Agrandissement dans toutes les directions`
16 | $\mathcal{F} \leftarrow \{v \mid v \in V, \mathcal{N}_v \nsubseteq V\}$;
17 **jusqu'à** $i = n$;
18 **retourner** R

du devis $\Lambda \geq 1$ fixé par l'utilisateur, on propose deux fonctions objectif dans cet esprit de budget. Si l'utilisateur souhaite minimiser en priorité le coût monétaire de la requête tout en respectant le temps de réponse fixé par le devis, la fonction $\omega^{(E,m)}(t)$ définie par l'équation 4.8 lui sera proposée.

$$\omega^{(E,m)}(t) = \begin{cases} \omega^{(m)}(t) & \text{si } \omega^{(r)}(t) \leq \Lambda \times r_Q \\ \omega^{(r/m)}(t) & \text{sinon} \end{cases} \tag{4.8}$$

Si l'utilisateur souhaite en revanche maximiser les performances de sa requête en respectant le coût monétaire du devis, la fonction $\omega^{(E,r)}(t)$ définie par l'équation 4.8 lui sera proposée.

$$\omega^{(E,r)}(t) = \begin{cases} \omega^{(r)}(t) & \text{si } \omega^{(m)}(t) \leq \Lambda \times m_Q \\ \omega^{(r/m)}(t) & \text{sinon} \end{cases} \tag{4.9}$$

Enfin, si l'utilisateur souhaite simplement avoir les meilleurs performances, le coût monétaire le plus bas ou bien le meilleur rapport performances-prix, les fonctions utilisées pour le calcul des devis peuvent toujours être utilisées.

3.3 *Synthèse*

La méthode itérative qui vient d'être présentée permet de calculer des devis et d'optimiser les plans d'exécution de requêtes multi-clouds impliquant plus de quatre fournisseurs. Son bornage permet de maîtriser totalement des risques d'explosion combinatoire tout en l'autorisant à explorer une plus grande variété de plans d'exécution que la méthode exhaustive. Les fonctions objectifs qui la guident sont un outil de plus à disposition des utilisateurs de Nebula pour contrôler le comportement du middleware, et viennent en complément des devis et du seuil de tolérance proposés précédemment.

Cette méthode pourrait être complétée par l'ajout de nouvelles règles, permettant par exemple d'explorer des plans d'exécution multi-clouds reproduisant la jointure à base de semi-jointures existant dans les environnements distribués et parallèles [3, ch. 4, p. 153] dans une optique de diminution des coûts monétaires en économisant sur les transferts de données.

4 *Conclusion*

Nous avons dans un premier temps proposé une stratégie de recherche exhaustive des plans d'exécution multi-clouds linéaires. Cette stratégie produit une DAG modélisant la requête multi-cloud, dont le coût des composantes, c'est-à-dire les sous-requêtes et les transferts de données inter-fournisseurs, est évalué à partir d'estimations produites par les fournisseurs cloud. L'utilisation de cette stratégie a pour conséquence d'exposer le middleware à un risque d'explosion combinatoire lors de la construction du DAG en raison de sa complexité factorielle en le nombre de fournisseurs impliqués par la requête multi-cloud. De plus, l'heuristique qu'elle utilise, en ne considérant que des plans d'exécution linéaires, expose davantage le middleware aux erreurs d'estimations lorsque les plans d'exécution sont profonds. Ces derniers risquent également de manipuler une plus grande quantité totale de données, pouvant alors rendre les plans d'exécution complexes très chers d'un point de

vue monétaire. Malgré ses limitations, la méthode exhaustive nous a permis de valider la pertinence d'une sous-traitance des requêtes multi-clouds dans un environnement expérimental à petite échelle, et de constater que l'optimisation dynamique est doublement pertinente dans le contexte du multi-cloud (cf. ch. 6). Afin de permettre au middleware de s'affranchir des limitations de la stratégie exhaustive, nous avons proposé une stratégie aléatoire et itérative. Cette dernière explore l'espace de recherche de voisinage en voisinage, de plan d'exécution optimal localement en plan d'exécution optimal localement. Les règles de génération de plans voisins permettent de considérer une plus grande variété de plans d'exécution tout en limitant le nombre total de plans d'exécution à évaluer.

Les deux stratégies de recherche utilisent des estimations sur la cardinalité en sortie, le temps de réponse et le coût monétaire des sous-requêtes pour calculer le coût monétaire et le temps de réponse des plans d'exécution multi-clouds. La qualité de ces estimations est cruciale, car les plans d'exécution sont en fait optimisés par rapport à celles-ci. Elles proviennent du modèle de coûts du SGBD des fournisseurs, et peuvent donc être inexactes. La correction de ces inexactitudes peut diminuer la sous-optimalité des plans d'exécution trouvés et *in fine* la qualité des devis présentés aux utilisateurs. De plus, la détection de ces inexactitudes lors de l'exécution des sous-requêtes donne l'opportunité au middleware de corriger les erreurs commises lors de l'optimisation : il peut donc ré-optimiser le reste du plan d'exécution à l'aune des valeurs constatées de cardinalité en sortie, de temps de réponse et de coût monétaire des sous-requêtes. Ces deux points sont traités par le chapitre suivant.

5

Prévenir et corriger la sous-optimalité des plans d'exécution multi-clouds

1 *Introduction*

L'optimalité des plans d'exécution multi-clouds produits par l'une ou l'autre stratégie présentées précédemment peut encore être améliorée, en prévenant les mauvais choix d'optimisation et en les corrigeant.

Premièrement, nous proposons d'améliorer la précision des estimations faites par les fournisseurs cloud sur les résultats des requêtes qui leurs seront soumises en les corrigeant grâce à un modèle de coûts appris, utilisant l'apprentissage en ligne [80]. En effet, cette technique a fait ses preuves dans les SGBDR pour la correction des estimations de cardinalité en sortie des opérateurs dans les plans d'exécution [34]. Nous proposons à cet effet un modèle d'apprentissage ensembliste [81], permettant ainsi à Nebula de profiter des avancées permises par l'apprentissage automatique en matière de modélisation des coûts.

Deuxièmement, nous proposons de réoptimiser les plans d'exécution multi-clouds à la fin de l'exécution de leurs sous-requêtes. Les plans d'exécution multi-clouds construits lors de l'exploration de l'espace de recherche sont réutilisés à cet effet. Leurs sous-requêtes et les possibles dépendances entre

elles sont utilisées comme environnement d'un SMA assurant l'optimisation dynamique de la requête. La sous-optimalité des plans d'exécution multi-clouds vis-à-vis de leur temps de réponse et de leur coût monétaire s'en trouve donc diminuée, et le respect des devis est renforcé.

Dans ce chapitre, nous présentons d'abord le modèle de coûts multi-cloud conçu pour Nebula dans la section 2. Nous présentons ensuite la méthode d'optimisation dynamique dans la section 3. Ce chapitre est conclut par la section 4

2 *Modèle de coûts multi-cloud*

Comme évoqué précédemment, le processus d'optimisation de requêtes multi-clouds, permettant à la fois de calculer les devis et d'orchestrer la sous-traitance de l'exécution des sous-requêtes auprès des fournisseurs, a besoin d'informations sur celles-ci afin de permettre à sa stratégie de recherche de chiffrer les plans d'exécution qu'elle considère. Les sous-requêtes étant formulées sur des relations stockées chez des fournisseurs cloud, c'est vers eux que l'optimiseur va se tourner pour connaître leur cardinalité en sortie et une estimation de leur temps de réponse. Ces dernières, produites par le modèle de coûts du SGBD sous-jacent à l'offre DBaaS du fournisseur, sont certainement erronées : l'inexactitude des modèles de coûts faisant consensus [33], le middleware ne peut pas se contenter de croire aveuglément les estimation du fournisseur et doit donc essayer de les corriger. S'il est irréaliste de supposer un libre accès aux statistiques sur la distribution statistique des valeurs des attributs au sein des relations qu'ils stockent (le fournisseur ne maintient d'ailleurs pas nécessairement de telles informations), et si l'accès aux données en elles-mêmes est facturée, alors la correction des estimation des fournisseurs apparaît comme une tâche difficile à réaliser car le middleware n'a à sa disposition immédiate que les métadonnées que les fournisseurs veulent bien partager ainsi que l'historique des requêtes qu'il a soumises aux fournisseurs pour constater l'écart entre les estimations et la réalité de leur cardinalité en sortie et de leur temps de réponse.

L'utilisation de l'historique pour l'amélioration de la précision des modèles de coûts est, depuis une vingtaine d'années, au centre de la recherche sur ces derniers [82] et a conduit à l'étude des modèles de coûts appris (*learned cost model*), utilisant des modèles d'apprentissage automatique pour faire leurs estimations ; les développement récents de cette approche sont brièvement présentés ci-après. L'architecture de l'essentiel de ces techniques repose sur une modèle dit de *batch learning*, où le modèle d'apprentissage est entraîné sur un jeu de données pré-existant avant d'être mis en production, c'est-à-dire exploité pour ses estimations. L'entraînement d'un tel modèle aurait un coût exorbitant dans le contexte du multi-cloud, coût que n'aurait pas

à amortir un modèle d'apprentissage en ligne (*online learing*). Conçus pour traiter des flux de données et apprenant à chaque nouveau point de donnée qu'ils rencontrent, ils semblent être particulièrement adaptés pour un modèle de coûts multi-cloud et sont donc exploités dans Nebula.

Ce dernier dispose plusieurs composants : le premier est un modèle d'apprentissage ensembliste proposant de résoudre à la fois le problème de la correction d'estimation de sélectivité et d'estimation du temps de réponse des sous-requêtes ; le second est un ensemble de régressions linéaires en ligne permettant de modéliser les temps de transferts de données inter-fournisseurs ; enfin, le troisième utilise deux modèles légers d'apprentissage en ligne permettant de réévaluer le coût monétaire et le temps de réponse estimé dans les devis. Chacun de ces modèles est présenté après les quelques notions relatives aux modèles de coûts appris évoquées ci-après.

2.1 *Travaux existants sur les modèles de coûts appris en ligne*

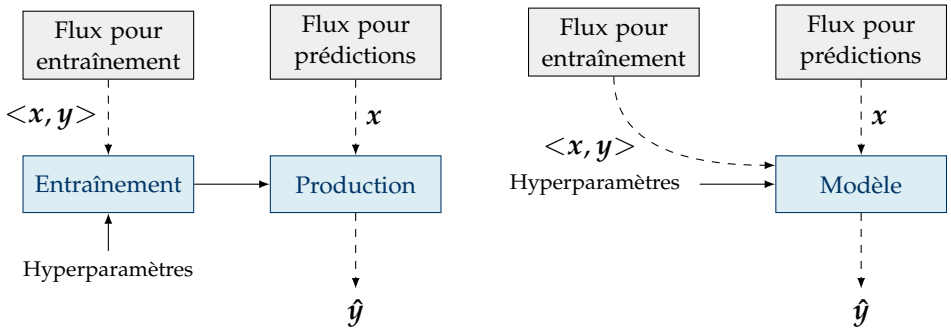
Le problème de l'estimation de la cardinalité en sortie des opérateurs dans un plan d'exécution s'est posé dès la conception des premiers SGBDR à la fin des années 1970 [83], et il est généralement admis que ce problème est encore ouvert [34]. S'agissant d'une extension du problème statistique de l'estimation de fonction de densité, il doit s'envisager dans le contexte du traitement de requêtes par les SGBD : la capacité à produire des prédictions précises de la cardinalité en sortie des opérateurs doit donc faire l'objet d'un compromis afin de limiter l'impact du calcul de ces estimations sur le temps de réponse des requêtes et sur les ressources laissées disponibles pour le traitement de potentielles autres requêtes concurrentes. Certains SGBD comme PostgreSQL utilisent donc en pratique des hypothèses simplificatrices comme l'uniformité dans la distribution des données des attributs, l'indépendance de ces derniers ainsi que le principe d'inclusion des domaines des clés sur lesquelles les jointures sont formulées [84] afin d'accélérer le processus d'optimisation des requêtes.

Ces contraintes de performances vis-à-vis de la modélisation des coûts et de l'optimisation ont longtemps rendu inenvisageable l'utilisation de l'historique des requêtes traitées par le SGBD dans ces processus. En effet, ce n'est pas avant le début des années 2000 que ce dernier a commencé a être exploité, d'abord plutôt sommairement dans l'optimiseur LEO de DB2 pour l'estimation de la cardinalité [82], puis au sein de modèles d'apprentissage automatique de plus en plus complexes dont les évolutions contemporaines s'inscrivent dans la mouvance *ML for systems* [85] et vont jusqu'à remettre en cause l'architecture fonctionnelle du SGBD [86-88]. Plusieurs modèles d'apprentissage profond ont été proposés jusqu'à présent [89, 90], permettant d'atteindre une grande précision même pour les requêtes de similarité [91]. L'estimation du temps de réponse des requêtes est un autre problème sur

lequel l'apprentissage automatique a largement été utilisé, soit pour le reparamétrage des modèles analytiques utilisés pour la modélisation du coût des opérateurs par les SGBD [92], soit pour la réestimation des valeurs calculées par ceux-ci. Divers modèles de régression ont été proposés, comme des arbres de décision [93] prédisant les intervalles pour le temps de réponse d'une requête, des machines à vecteurs de support [94] ou encore des régressions aux k-voisins [95], ces dernières permettant d'obtenir les résultats les plus précis à ce jour. Les derniers travaux consacrés à une recherche plus appliquée sur le sujet de l'apprentissage d'un estimateur de cardinalité en sortie d'un opérateur, dans le cadre du prototypage du modèle Bao au sein de PostgreSQL [96], ont démontrés qu'il était relativement aisé de greffer un modèle d'apprentissage automatique à un SGBDR ayant atteint un stade avancé de maturité tout en lui permettant d'améliorer spectaculairement ses performances.

Ces résultats, quoique particulièrement prometteurs, sont malheureusement à relativiser dans le contexte du traitement de requêtes multi-clouds en raison de l'essence même de la puissance de ces approches : leur exploitation, lors de leur phase d'entrainement, d'un jeu de données massif et varié construit à partir d'un historique de requêtes. Il est en effet inenvisageable pour le middleware de construire un jeu de données suffisant en raison du coût prohibitif que cela engendrerait et qui ne serait amortit que par le traitement de milliers de requêtes multi-clouds. Les ambitions du middleware en matière de modélisation des coûts doivent donc se circonscrire à tenter de corriger les estimations des fournisseurs, en n'ayant à sa disposition qu'un maigre jeu de données construit à mesure que les utilisateurs formulent des requêtes multi-clouds. Un modèle d'apprentissage intégré au middleware devra donc nécessairement démarrer à froid, empêchant alors l'utilisation des modèles d'apprentissage classiques fonctionnant en *batch* tout en rendent l'approche de l'apprentissage en ligne pertinente.

Vu par le prisme du génie logiciel, l'apprentissage en *batch* se calque sur un type d'architecture dite « lambda » [97], dans laquelle un modèle est préalablement entraîné puis exposé à un flux de données sans évoluer de son propre chef. La capacité de ces modèles à généraliser est fortement liée à la fois à la composition du jeu de données d'entraînement, mais également au processus même d'entraînement et de validation afin notamment d'éviter un sur-apprentissage du jeu de données d'entraînement. Toute évolution imprévue, ou dont la manifestation n'a pas été matérialisée dans le jeu de données, dans le contenu du flux de données aura des conséquences négatives sur la capacité prédictive du modèle : l'historisation du flux de données et le ré-entraînement du modèle sont alors de rigueur pour que la qualité de celui-ci perdure, impactant nécessairement sa complexité logicielle et la difficulté de leur maintenance sur le long terme [98]. L'apprentissage en ligne prend le contrepied de cette approche en suivant une architecture dite « kappa » [99],



(a) Vie d'un modèle qui apprends en *batch*. (b) Vie d'un modèle qui apprends en ligne.

FIGURE 5.1 – Vie de modèles apprenant en *batch* (figure 5.1a) et en ligne (figure 5.1b). Les prédictions $\hat{y}$ sont faites à partir de points de données généralement inconnus x. Le modèle d'apprentissage en ligne est régulièrement exposé à la vérité terrain y correspondant à x pour qu'il apprenne progressivement.

comme illustré par la figure 5.1, dans laquelle la vie du modèle est totalement intégrée dans le flux de données qu'il a à traiter. Il n'est donc plus question de phase d'entraînement et d'exploitation : le modèle se met à jour grâce à l'ingestion constante de nouveaux points de données [80], permettant un démarrage à froid et une amélioration continuelle de sa capacité de généralisation. La dilution progressive des mises à jour induites par des points de données anciens est à double tranchant, permettant à la fois à ces modèles d'être résistants face aux glissements dans la distribution du flux de données, au prix du risque d'un oubli d'une partie de la distribution des données, qualifié alors de catastrophique. Le choix de modèles d'apprentissage en ligne est cependant plus restreint que pour du *batch* du fait de la nature de leur entraînement, la descente stochastique du gradient, qui n'est pas applicable à tous les modèles.

Ce sont donc des modèles d'apprentissage automatique nécessairement simples qui ont été appliqués lors de travaux de thèse récents [34] sur la correction de l'estimation du facteur de sélectivité. Leur simplicité n'a pas empêché leur succès, ces travaux démontrant qu'une régression linéaire bayésienne en ligne permet d'obtenir des estimations bien plus précises que l'état de l'art d'alors [100] et que PostgreSQL sur la cardinalité en sortie des opérateurs dans un plan d'exécution, en particulier lorsque la distribution des données dans les relations change. Les autres modèles évalués, à savoir une régression linéaire, une *factorisation machine*, un petit réseau de neurones ainsi qu'un arbre de décision de Hoeffding, ont également obtenus de bons résultats [101]. La précision des estimations portant sur la cardinalité en sortie des opérateurs et le temps de réponse des requêtes ayant progressé proportionnellement à la complexité des modèles d'apprentissage *batch* employés

dans les SGBD, il est raisonnable de penser que des modèles d'apprentissage en ligne plus complexes seraient plus performants. S'il semble difficile d'envisager l'utilisation de modèles plus complexes au sein de notre middleware car ils nécessiteraient plus de données pour quitter la phase de démarrage à froid, il est cependant possible de combiner des modèles simples qui améliorent rapidement leur précision. Cette approche, tout à fait adaptable en ligne, est celle de l'apprentissage ensembliste (*ensemble learning*) [81] : c'est donc un modèle d'une telle nature qui a été intégré dans le modèle de coûts multi-cloud.

2.2 *Apprentissage ensembliste pour la réestimation des sous-requêtes et des devis*

Plus précisément, c'est un empilement (*stacking*) de régressions qui est utilisé pour réestimer les estimations des fournisseurs co-existant avec un métamodèle décidant quelle estimation sera retournée à l'utilisateur. Avant de pouvoir réestimer les estimations qui leurs sont relatives, le middleware doit encoder les sous-requêtes afin de passer leur représentation vectorielle au modèle d'apprentissage ensembliste du modèle de coûts.

2.2.1 *Vectorisation des sous-requêtes*

Les travaux consacrés à la conception des *learned cost models* ont nécessairement étudié la manière d'encoder les requêtes SQL en entrée des modèles. La complexité des processus de vectorisation est variable, allant de l'encodage du graphe de jointures de la requête sous forme de vecteurs *1-hot* [102] à du calcul sur des plongements de tuples (*row vector embeddings*) inspirés des plongements lexicaux (*word embeddings*) [87] en passant par des représentations apprises par renforcement des sous-requêtes [103]. Ces méthodes font toutes l'hypothèse que le schéma relationnel de la base de données sur laquelle sont formulées les requêtes qu'elles encodent sont statiques : un changement dans le schéma comme l'ajout d'un attribut va changer la taille des vecteurs en entrée des modèles d'apprentissage, rendant alors nécessaire leur ré-entraînement à l'aune de leur nouveau format d'entrée. Ces phases de collecte de données et de ré-entraînement n'étant toujours pas envisageable dans notre middleware, ce sont les métadonnées de la requête ainsi que les caractéristiques du plan d'exécution envisagé par le fournisseur qui sont encodées, plutôt que sa sémantique.

Le modèle de coûts construit pour une sous-requête q un vecteur $X_q = [x_0, \ldots, x_5]$, détaillé dans le tableau 5.1, construit à partir des estimations du fournisseur. Celui-ci encode la quantité de données manipulées par la sous-requête (x_0, x_3, x_5), sa complexité (x_2) ainsi que les choix effectués par l'optimiseur du fournisseurs (x_1, x_4). Ces informations sont obtenues en

Feature	**Description**
x_0	Estimation de la cardinalité en sortie
x_1	Estimation du temps de réponse
x_2	Nombre de jointures
x_3	Taille des tuples en sortie
x_4	Profondeur du plan d'exécution
x_5	Taille totale des relations en entrée de q

TABLE 5.1 – *Features* du vecteur X_q extraites à partir des plans d'exécution et des estimations produits par le fournisseur

pratique à l'aide de la métabase du SGBD sous-jacent à l'offre DBaaS et de la commande SQL `EXPLAIN`. La plupart de ces *features* sont utiles pour l'estimation du temps de réponse des requêtes, ce dernier étant corrélé à la quantité de données à traiter et à la complexité de la requête. Les estimations de cardinalité en sortie faites par le modèle de coûts du fournisseur ainsi que la quantité de données à traiter en entrée de la requête et sa complexité sont quant à elles pertinentes pour ré-estimer la cardinalité en sortie de la sous-requête.

2.2.2 *Modèle d'apprentissage ensembliste*

L'architecture du modèle $\mathbf{M} : \mathbb{R}^6 \mapsto \mathbb{R}^*$ est illustrée par la figure 5.2. Sa composition a été déterminée de manière empirique. Il prend en entrée un vecteur X_q modélisant une requête q donnée. Lorsqu'il est instancié pour estimer la cardinalité en sortie $\hat{y}_q^{(SE)}$, le modèle est noté $\mathbf{M}^{(SE)}$ et est noté $\mathbf{M}^{(RT)}$ lorsqu'il retourne une estimation du temps de réponse $\hat{y}_q^{(RT)}$. Par la suite, il faut lire respectivement x_0 et x_1 pour $\mathbf{M}^{(SE)}$ et $\mathbf{M}^{(RT)}$ à la place de x_i.

$\mathbf{M}$ est un modèle de régression ensembliste multi-couches. Le première couche consiste en une régression linéaire $f_0^{(1)} : \mathbb{R}^3 \mapsto \mathbb{R}^*$ retournant $y_0^{(1)}$ et en une régression aux k-voisins $f_1^{(1)} : \mathbb{R}^3 \mapsto \mathbb{R}^*$ retournant $y_1^{(1)}$, toutes deux prenant en entrée $X^{(1)} = [x_0, x_1, x_2]$. La deuxième couche est une couche de mixage (*blending layer*), dont les modèles prennent en entrée $X^{(2)} = [x_i, y_0^{(1)}, y_1^{(1)}]$. Ceux-ci sont une fonction $f_0^{(2)} : (\mathbb{R}^*)^3 \mapsto \mathbb{R}^*$ retournant l'estimation moyenne $y_0^{(2)}$ des modèles de la première couche et du SGBD du fournisseur, ainsi que d'une régression aux k-voisins $f_1^{(2)} : (\mathbb{R}^*)^3 \mapsto \mathbb{R}^*$ retournant $y_1^{(2)}$. La troisième couche est une fonction $f^{(3)} : (\mathbb{R}^*)^5 \mapsto \mathbb{R}^*$ prenant en entrée $X^{(3)} = [x_i, y_0^{(1)}, y_1^{(1)}, y_0^{(2)}, y_1^{(2)}]$ et retournant la valeur moyenne de toutes les estimations précédentes, notée $y^{(3)}$.

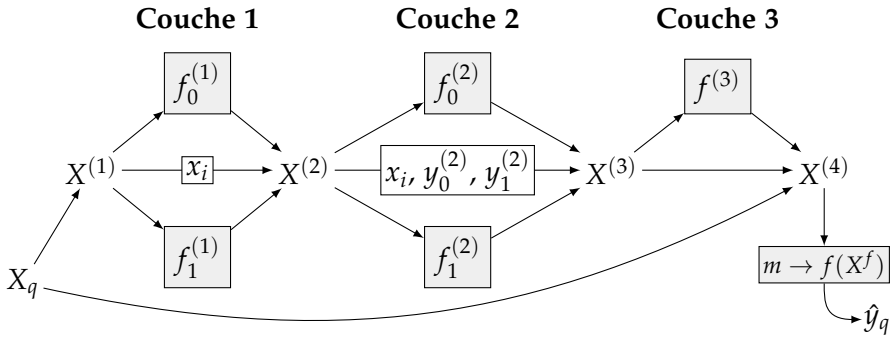


FIGURE 5.2 – Modèle d'apprentissage ensembliste multi-clouches **M**.

Enfin, **M** a un méta-modèle. Il s'agit d'un classifieur à base d'arbre de Hoeffding, un arbre de décision ayant comme propriété de n'avoir besoin que d'échantillons de données de petite taille pour diviser ses sommets, $m : \mathbb{R}^{11} \mapsto \{f^{(0)}, f_0^{(1)}, f_1^{(1)}, f_0^{(2)}, f_1^{(2)}, f^{(3)}\}$ (avec $f^{(0)}$ notant le modèle de coûts du fournisseur retournant x_i). Celui-ci prend en entrée $X^{(4)} = X_q \parallel [y_0^{(1)}, y_1^{(1)}, y_0^{(2)}, y_1^{(2)}, y^{(3)}]$ et détermine le modèle f qui devrait produire la meilleure estimation pour q. L'estimation de f est alors retournée par **M**.

Dès que de la vérité terrain y_q correspondant à un point de données X_q est disponible, le modèle est entraîné en trois phases, en commençant par le méta-modèle m, puis le modèle $f_1^{(2)}$ de la deuxième couche et enfin les régressions $f_0^{(1)}$ et $f_1^{(1)}$ de la première couche. Les autres fonctions $f_0^{(2)}$ et $f^{(3)}$ calculant des moyennes, celles-ci ne nécessitent évidemment pas d'entraînement.

Ce modèle de coûts restant un modèle d'apprentissage automatique, ce dernier ne sera pas en mesure d'effectuer des estimations fiables dès sa mise en route du fait du faible nombre d'échantillons sur lesquels il aura été entrainé. Afin de protéger l'optimiseur de mauvaises estimations conséquentes de ce démarrage à froid, le modèle ne sera pas utilisé jusqu'à ce qu'un seuil de nombre de requêtes traitées ne sera pas franchit. Lors de cette phase de démarrage, l'estimation x_i est retournée à la place de l'estimation du modèle de coûts. Enfin, la nature récursive des plans d'exécution est exploitée pour récolter plus de données d'exécution en n'apprenant pas seulement sur la sous-requête q, mais également sur ses résultats intermédiaires.

2.2.3 *Finalisation des estimations sur les sous-requêtes*

En supposant que l'offre DBaaS du fournisseur cloud suit un modèle économique similaire au service BigQuery de Google, où les locataires sont facturés proportionnellement à la quantité de données qu'ils manipulent, il est possible de calculer le coût d'une sous-requête q soumise à un fournisseur P_i comme suit. Soient $\epsilon_{P_i}^{(Q)}$, $\epsilon_{P_i}^{(E)}$ et $\epsilon_{P_i}^{(S)}$ respectivement les coefficients de

facturation pour l'interrogation, l'export et le stockage par gigaoctet de P_i. Le coût d'interrogation de q est définit comme $M_{P_i}^{(Q)}(q) = \epsilon_{P_i}^{(Q)} \times x_5^{(q)}$, le coût de stockage des résultats intermédiaires comme $M_{P_i}^{(S)}(q) = \epsilon_{P_i}^{(S)} \times \hat{y}_q^{(SE)}$, et le coût d'export de ceux-ci vers un autre fournisseur P_j comme $M^{(E)}(q, P_i, P_j) = \hat{y}_q^{(SE)} \times (\epsilon_{P_i}^{(E)} + \epsilon_{P_j}^{(S)})$.

2.2.4 *Temps de transfert sur le réseau*

Les temps de transfert des résultats intermédiaires entre fournisseurs ayant un impact majeur sur le calcul des devis et le processus d'optimisation, il convient de les estimer correctement. Le modèle de coûts est donc capable d'apprendre à les estimer en utilisant des régressions linéaires en ligne associées à chaque couple de fournisseurs recensés dans le schéma multi-cloud. Soit $\mathbf{N}$ une matrice dans laquelle chaque élément est une régression linéaire $g_{ij} : \mathbb{R}_+ \mapsto \mathbb{R}_+$ définie comme $g_{ij}(x) = a_{ij}x + b_{ij}$ permettant d'estimer le temps de transfert d'une quantité de données x d'un fournisseur i à un fournisseur j. Afin d'accélérer la convergence des paramètres a (exprimé en gigaoctets par seconde) et b (exprimé en secondes) de chaque régression, a est initialisé à l'aide des informations disponibles sur le débit de la connexion des fournisseurs à Internet et à $b = 0$. Lorsqu'un transfert de données entre i et j se produit, la fonction g_{ij} correspondante est mise à jour.

2.2.5 *Retraitement des devis*

Outre les estimations relatives aux sous-requêtes, le modèle de coûts multi-cloud réévalue également les devis que Nebula présentera à ses utilisateurs. L'utilisation d'une méthode d'optimisation dynamique dans le middleware aura pour conséquence de réduire l'écart entre les estimations et les valeurs d'exécution réelle des requêtes multi-clouds, à tel point qu'il est possible de postuler qu'il existe une relation linéaire entre celles-ci. Les devis étant chiffrés à partir de plans d'exécution multi-clouds, il convient de les réévaluer afin de prendre en compte l'impact de la ré-optimisation.

Le modèle de coûts multi-cloud est donc équipé de deux modèles d'apprentissage en ligne pour la réévaluation des devis. Le premier, noté $\mathbf{P}^{(M)}$ et défini comme $\mathbf{P}^{(M)} : \mathbb{R}_+^4 \mapsto \mathbb{R}_+$, est une régression linéaire permettant de réévaluer la composante prix du devis. Le second, noté $\mathbf{P}^{(RT)}$ et défini comme $\mathbf{P}^{(RT)} : \mathbb{R}_+^4 \mapsto \mathbb{R}_+$, est une régression aux k-voisins réévaluant l'estimation de temps de réponse de la requête multi-cloud. Tous deux prennent en entrée un vecteur $X_D = \mathbb{R}_+^4$ modélisant un devis D à partir de ses estimations de coût monétaire et de temps de réponse, de la somme de la taille des relations en entrée de la requête multi-cloud ainsi que le nombre de fournisseurs qu'elle implique.

2.3 *Synthèse*

Le modèle de coûts de Nebula utilise l'apprentissage en ligne pour accomplir l'intégralité des tâches qu'il doit accomplir. L'utilisation de ces techniques permet de profiter de la bonne précision apportée par l'apprentissage automatique appliqué à la modélisation des coûts dans les SGBDR d'une manière adaptée à leur déploiement en environnements multi-clouds : la capacité de démarrage à froid et d'apprentissage au fil de l'eau des modèles en ligne permettant ainsi au middleware de s'épargner le coût monétaire de la construction d'un jeu de données d'entraînement à partir de requêtes à blanc, lequel serait vraisemblablement impossible à amortir. Nous avons proposé un modèle d'apprentissage ensembliste constitué d'un empilement de modèles d'apprentissage simples consacré à l'estimation de la cardinalité en sortie des sous-requêtes et à la correction de leur temps de réponse par les fournisseurs. Nous avons également proposé un modèle d'apprentissage estimant les temps de transfert réseau, celui-ci étant judicieusement initialisé pour accélérer l'apprentissage. Enfin, deux régressions peu complexes sont utilisées pour réévaluer les devis (un modèle pour le coût monétaire et un autre pour le temps de réponse) afin de prendre en compte les effets de l'optimisation dynamique sur les plans d'exécution.

L'hébergement dans le cloud des sources de données empêche de modéliser les requêtes à travers leur sémantique, car les données ne sont accessibles qu'à travers un rapport commercial avec le fournisseur. Les méthodes les plus efficaces issues de la littérature restent donc inaccessibles car entraînant des dépenses prohibitives au moment de leur entraînement. Notre modèle de coûts se cantonne donc à de la correction d'estimation, et reste en définitive dépendant du modèle de coûts des SGBD *multi-tenant* des fournisseurs. L'utilisation d'un modèle ensembliste au sein du modèle de coûts a montré qu'il était possible de complexifier les modèles d'apprentissage en ligne utilisés : il serait alors envisageable d'étudier des modèles plus complexes pour améliorer leur précision, tout en gardant à l'esprit le faible volume de données disponibles pour entraîner les modèles.

3 *Optimisation dynamique de requêtes multi-clouds*

Après que l'utilisateur ait choisi un devis pour la requête multi-cloud, Nebula procède à son traitement en orchestrant la sous-traitance de ses différentes sous-requêtes auprès des fournisseurs. Exécuter aveuglément le plan d'exécution associé au devis peut s'avérer risqué pour le middleware, dans la mesure où celui-ci est construit à partir d'estimations sur la cardinalité en sortie des sous-requêtes. Nebula intègre ainsi une méthode d'optimisation dynamique, permettant de ré-optimiser la requête multi-cloud dès qu'une

sous-requête a terminé son exécution de sorte à essayer de respecter le devis. Celle-ci utilise un SMA conçu pour optimiser les requêtes multi-clouds. Elle réutilise le graphe de requêtes produit pour le calcul des devis comme environnement dans lequel naviguent des agents coopératifs ; ils commandent l'exécution progressive de la requête multi-cloud tout en la ré-optimisant. Un rapide état de l'art sur les techniques d'optimisation dynamique pertinentes dans le contexte du multi-cloud est proposé ci-après, avant la description de la méthode à base d'agents coopératifs conçue pour Nebula.

3.1 *Travaux existants sur l'optimisation dynamique*

Les performances d'un plan d'exécution sont fortement corrélées à la quantité de données que chaque opérateur du plan d'exécution doit traiter. Plutôt que d'exécuter aveuglément les plans d'exécution tels-quels en sachant par avance que la probabilité qu'ils soient sous-optimaux est forte, il serait judicieux de vérifier si les performances réelles du plan d'exécution sont conformes aux estimations et de ré-optimiser ce dernier si ce n'est pas le cas. C'est cette intuition qui est derrière l'idée d'optimisation dynamique, définissable comme l'exécution progressive d'une requête en ré-optimisant son plan d'exécution à l'aune des informations récupérées à l'exécution afin de limiter les effets des erreurs d'estimation [104]. Les travaux sur ce sujet ont produit des techniques d'optimisation dynamique pouvant être classées dans deux catégories. La première est celle des techniques centralisées, dans laquelle le contrôle est confié à un processus unique ; la seconde est celle des techniques décentralisées, dans laquelle le contrôle est confié à un ensemble de processus. De plus, la ré-optimisation peut s'effectuer pour différents niveaux du traitement de la requête.

Ce sont les techniques s'appliquant au niveau inter-opérateurs qui sont les plus intéressantes en environnements multi-clouds car les fournisseurs sont considérés comme des boîtes noires. On trouve, dans la littérature, deux genre d'approches à ce niveau. La première approche est celle du brouillage de requêtes (*query scrambling*) [105], conçue pour réduire les temps d'attente dans les plans d'exécution lorsqu'un opérateur n'a pas achevé sa tâche dans un temps imparti, voire quand il a échoué suite à une défaillance matérielle ou à une déconnexion du réseau. Les *pipelines* identifiés au sein du plan d'exécution peuvent donc être réordonnancés ou recomposés. Si le *query scrambling* est intéressant du point de vue des performances dans les SGBD classiques, il ne l'est pas nécessairement pour les coûts monétaires inhérents au cloud. En effet, les temps d'attentes dans un plan d'exécution multi-cloud se matérialisent monétairement par un surcoût lié au stockage des résultats intermédiaires, surcoût peu important du fait du caractère peu onéreux du sto-

ckage des données sur du DBaaS [1]. La seconde approche consiste à envisager les opérateurs des plans d'exécution comme des agents mobiles [106], c'est-à-dire capables de se déplacer à leur initiative d'un site à l'autre d'un SGBD distribué. Ils coopèrent afin de remplir leur objectif commun de maximiser les performances de l'exécution de la requête. En partageant les informations qu'ils obtiennent individuellement sur les résultats intermédiaires que leur opérateur produit, ils déclenchent un nouveau calcul des estimations sur la taille des résultats intermédiaires de leurs homologues n'ayant pas encore exécuté leur opérateur. Ces nouvelles estimations, à nouveau partagées entre les agents, les conduisent à revoir leur décision de placement physique (en pouvant migrer d'un site à un autre) et logique (en proposant de changer le moment auquel leur opérateur sera exécuté). Les plans d'exécution distribués peuvent donc être radicalement modifiés à la fin de l'exécution de chaque opérateur, limitant ainsi les transferts de données inutiles et la quantité totale de données traitées dans le plan d'exécution.

Les autres techniques s'appliquant au niveau des tuples, comme le routage de tuples au travers d'opérateurs nommés Eddies [107, 108], ou encore au niveau intra-opérateur, comme la décomposition de requêtes [73], les jointures à base d'agents mobiles [109] ou le gonflage d'opérateurs (*operator inflation*) [110], ne sont pas adaptées à l'optimisation de requêtes multi-clouds car elles nécessiteraient de pouvoir contrôler au moins en partie l'exécution des sous-requêtes chez les fournisseurs cloud, ce qui est irréaliste car ils sont considérés comme des boîtes noires.

La littérature sur les SGBD distribués montre que l'optimisation dynamique par agents prenant des décisions de manière décentralisée avait la capacité de produire des plans d'exécution plus efficaces que des plans construits de manière traditionnelle [106, 109] grâce à leur proactivité. Celle-ci leur permet d'éviter de conserver un mauvais ordonnancement des opérateurs lorsqu'ils constatent que celui-ci est sous-optimal grâce à des ré-estimations de leur coûts déclenchées par la collecte de nouvelles informations sur les résultats intermédiaires. Nous nous en inspirons donc pour proposer une méthode d'optimisation dynamique de requêtes multi-clouds utilisant un SMA : en changeant la fonction objectif des agents afin d'ajouter des considérations monétaires dans leur raisonnement, ils devraient être en mesure de se plier à des objectifs de temps de réponse et de coût monétaire au niveau de la requête dictés par le devis, contrôlant ainsi l'orchestration de la sous-traitance de l'exécution de la requête multi-cloud.

1. Voir p. ex. les tarifs du service BigQuery de Google, où la facturation du stockage de relations se décompte mensuellement. `https://cloud.google.com/bigquery`

3.2 *Un système multi-agents pour l'optimisation dynamique de requêtes multi-clouds*

Le SMA de Nebula consiste en un ensemble d'agents coopératifs $\mathcal{A}$ déployés pour orchestrer la sous-traitance des sous-requêtes de Q étant donné un devis $<m_Q, r_Q, \Lambda>$ associant un coût monétaire, un temps de réponse et un seuil de tolérance vis-à-vis de ce devis de la part de l'utilisateur. Cet ensemble d'agents est défini formellement comme $\mathcal{A} = \{A_{v_i} \mid v_i \in V, d^+(v_i) = 0\}$, c'est-à-dire qu'un agent est créé pour chaque sommet dans $G_Q = <V, E>$ n'ayant pas de prédécesseur, c'est-à-dire pour chaque sous-requête mono-fournisseur. Chaque agent a_i est associé à une sous-requête q_i ainsi qu'au fournisseur $P^{(v_i)}$ auquel elle est affectée. Lorsqu'ils sont créés, ils commencent à ordonner l'exécution de leur sous-requête.

Les agents évoluent dans un graphe de sous-requêtes G_Q. Celui-ci modélise les dépendances entre les sous-requêtes de tout les plans d'exécution considérés pendant l'exploration de l'espace de recherce. Les coûts monétaires et les performances de ces sous-requêtes ont donc déjà été estimées. Dans la mesure où la méthode d'optimisation dynamique a été évaluée avec la stratégie de recherche exhaustive (cf. ch. 6), nous considérons par la suite que G_Q est le graphe illustré par la figure 4.1 (cf. ch. 4 sect. 2.1). L'utilisation conjointe de la mathode d'optimisation dynamique avec la stratégie aléatoire serait possible en convertissant le graphe S (cf. ch. 4 sect. 3.2) modélisant l'espace de recherche exploré par cette dernière en un graphe de sous-requêtes.

Les agents orchestrant en fait une exécution distribuée de la requête multi-cloud, la synchronisation de leurs décisions et de leurs ordres d'exécution est nécessaire. Ainsi, chaque sommet associé à une requête multi-fournisseurs est équipé d'une barrière de synchronisation, empêchant ainsi son exécution tant que les résultats intermédiaires de ses prédécesseurs ne sont pas disponibles. Les agents communiquent en altérant leur environnement : les estimations m_v, r_v et s_v avec $v \in V$ et m_e, r_e avec $e \in E$ sont écrasées par les valeurs récupérées à l'exécution. Lorsqu'un agent $A_v \in \mathcal{A}$ associé à un sommet choisit un nouveau sommet $v' \in N_v^-$ (N_v^- étant l'ensemble des successeurs immédiats de v) auquel s'attacher, il communique sa décision en supprimant de G_Q l'ensemble des sommets associés à des sous-requêtes devenues irréalisables.

Chaque agent avance donc dans le graphe de sommet en sommet, le sommet v' mentionné est plus précisément choisi comme étant le successeur de v dans un arbre $T = <V_T, E_T> \in \mathcal{T}_Q$ retourné par la fonction définie dans l'équation 5.1.

$$MeilleurArbre \mapsto \min_{t \in \mathcal{T}_Q} \begin{cases} r_t \text{ si } \exists t : m_t \leq (1+\Lambda) m_Q \\ \qquad\qquad \wedge\ r_t \leq (1+\Lambda) r_Q \\ \left| \frac{r_t}{m_t} - \frac{r_Q}{m_Q} \right| \text{ sinon} \end{cases} \tag{5.1}$$

Celle-ci utilise le seuil de tolérance de l'utilisateur Λ. Elle cherche à minimiser le temps de réponse de Q en considérant le coût monétaire du devis comme un budget à ne pas excéder. Si cette minimisation n'est pas possible, alors la fonction cherche alors le plan d'exécution minimisant le rapport entre le temps de réponse et le coût monétaire de Q.

Un agent a_i modifie G_Q en deux occasions. La première suit immédiatement la fin de l'exécution de sa sous-requête associée q_i, en déclenchant des ré-estimations pour toutes les sous-requêtes dans l'ensemble $\mathcal{N}^-_{v_i}$, donc associée à tous les sommets atteignables depuis v_i. Les nouvelles estimations sont utilisées pour recalculer le coût monétaire et le temps de réponse des arcs atteignables depuis v_i. In convient de noter que, pour les premières phases de l'exécution de Q, cette étape de mise à jour peut être longue car de complexité factorielle en le nombre de fournisseurs impliqués par Q. La seconde est quand un agent modifie le DAG après avoir fait un choix de ré-optimisation. G_Q est alors amputé de l'ensemble des plans d'exécution qui ne sont plus réalisables. En supposant que a_i se déplacera vers v'_i, l'ensemble des sommets à supprimer est alors $V_\ominus^{(v')}$ définit dans l'équation 5.2.

$$V_\ominus^{(v')} = \left(\bigcup_{\substack{v' \notin V_T \\ <V_T, E_T> \in \mathcal{T}_Q}} V_T \right) \setminus \left(\bigcup_{\substack{v' \in V_T \\ <V_T, E_T> \in \mathcal{T}_Q}} V_T \right) \tag{5.2}$$

Celui-ci contient l'ensemble des sommets des arbres dans $\mathcal{T}_Q$ ne contenant pas v' et n'étant pas dans des arbres le contenant. Outre la communication entre agents, l'élagage du graphe permet d'éviter des étreintes fatales produites par des décisions simultanées divergentes entre agents. La mise en place d'un système de communication directe entre agents assortie d'un protocole de négociation est donc évitée. La diminution de l'ordre de G_Q a aussi comme avantage de diminuer le nombre de plans d'exécution possible à chaque décision, rendant la ré-optimisation moins gourmande en ressources à mesure que l'exécution de la requête multi-cloud progresse.

L'algorithme 5.1 formalise le comportement cyclique des agents. Ces derniers commencent par ordonner l'exécution de la sous-requête correspondant au sommet de G_Q auquel ils sont associés. Ils mettent ensuite à jour le DAG en déclenchant les estimations pour les sous-requêtes successeures. Ensuite, l'agent choisit le prochain sommet auquel il va s'attacher, noté v', le faisant progresser dans un plan d'exécution qu'il perçoit comme étant le meilleur pouvant encore être construit. L'agent déclenche ensuite le transfert de ses résultats intermédiaires vers le fournisseur $P^{(v')}$, puis notifie ensuite la barrière de synchronisation de son nouveau sommet. Dans le cas où le nouveau sommet est affecté à un fournisseur différent de l'ancien, l'agent laisse la main à son homologue allant atteindre le nouveau sommet et termine son exécution.

Algorithme 5.1 : Comportement des agents

Entrées : $G_Q =< V, E >$ le graphe de la requête multi-cloud, v le sommet auquel l'agent a est attaché dont $P^{(a)}$ est le fournisseur auquel il est affecté

tant que $P^{(v)} = P^{(a)}$ **faire**
  Attendre la levée de la barrière associée à v ;
  Exécuter q associée à v sur $P^{(v)}$ et récupérer $<m'_v, r'_v>$;
  `/* Propagation des informations */`
  Sous mutex sur G_Q **faire**
    $m_v, r_v \leftarrow m'_v, r'_v$;
    pour chaque $e \in E_v^-$ **faire** Recalculer m_e et r_e;
    pour chaque $w \in \mathcal{N}_v^-$ *par ordre croissant de profondeur* **faire**
      Ré-estimer m_w, r_w et s_w ;
      pour chaque $e \in E_w^-$ **faire** Recalculer m_e et r_e;
  `/* Choix du sommet suivant et élagage de` G_Q `*/`
  Sous mutex sur G_Q **faire**
    $v' \leftarrow N_v^- \cap MeilleurArbre$;
    $V \leftarrow V \setminus V_{\ominus}^{(v')}$;
    $E \leftarrow \{e \mid e = <I, O> \in E,$
      $\{I, O\} \cap V_{\ominus}^{(v')} = \emptyset\}$;
  si $P^{(v)} \neq P^{(v')}$ **alors**
    Chargement des résultats intermédiaires sur $P^{(v')}$;
  $v \leftarrow v'$;
  Notifier la barrière de v ;

La figure 5.3 illustre l'évolution du SMA, en prenant cette fois-ci comme exemple un graphe de requête généré pour une requête multi-cloud Q impliquant trois fournisseurs avec $\mathcal{P}_Q = \{1, 2, 3\}$. Les agents A_1, A_2 et A_3 sont affectés respectivement aux sous-requêtes q_1, q_2 et q_3 puis commandent l'exécution de celles-ci (p. ex. A_1 demande à P_1 d'exécuter q_1). À l'étape A, l'exécution de q_2 s'est terminée, déclenchant la ré-estimation des tuples $<m_e, r_e, s_e>$ pour chaque $e \in \mathcal{N}_2^-$. En appliquant la formule donnée par l'équation 5.1, A_2 choisit q_{21} comme sous-requête suivante et élague G_Q des sommets désormais inatteignables. L'agent attend alors que la sous-requête commandée par A_1 soit terminé et que celui-ci le rejoigne. À l'étape B, A_1 a terminé et A_2 ordonne l'exécution de q_{21} tandis que A_3 attends toujours que l'exécution de q_3 se termine. À l'étape C, q_3 a terminé son exécution, A_3 a orchestré le déplacement de ses résultats intermédiaires vers le fournisseur associé à q_{213} et a notifié la barrière associée à cette sous-requête. Celle-ci attend donc l'arrivée de A_2 pour être exécuté, lequel s'en chargera ultérieu-

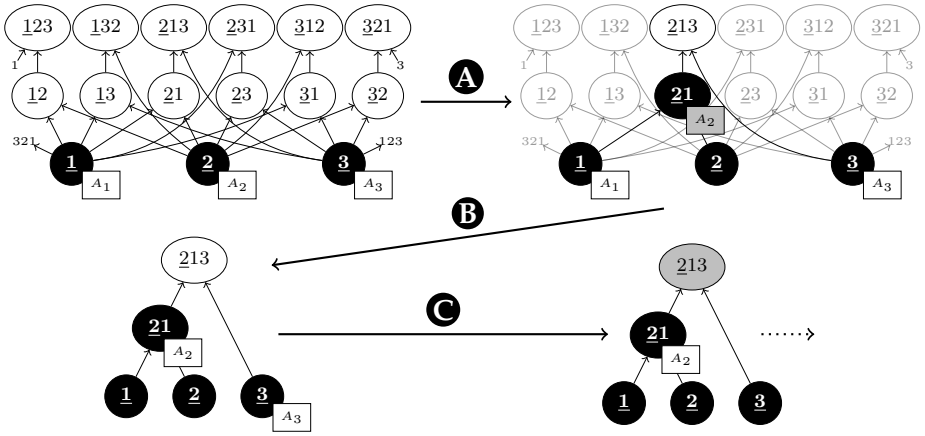


FIGURE 5.3 – Évolution d'un SMA orchestrant la sous-traitance de l'exécution d'une requête impliquant trois fournisseurs. Les chiffres soulignés correspondent au numéro du fournisseur auquel la sous-requête associée au sommet a été affecté.

rement et terminant finalement l'exécution de Q dont les résultats seront accessibles chez P_2.

3.3 *Synthèse*

La méthode d'optimisation dynamique proposée pour Nebula exploite un SMA, dans lequel des agents coopèrent au sein d'un graphe, déjà construit par la stratégie de recherche lors du calcul du devis, pour mener à bien l'exécution de la requête multi-cloud. Ils corrigent de potentiels mauvais choix d'optimisation induits par des erreurs d'estimation sur les résultats intermédiaires en ré-optimisant le plan d'exécution multi-cloud à chaque fois que de nouvelles informations sont disponibles sur lesdits résultats.

Même si la complexité algorithmique du choix des agents décroît à mesure que progresse l'exécution de la requête multi-cloud, celle-ci reste néanmoins élevée pour les premières décisions prises par l'agent et exposant donc Nebula à un risque d'explosion combinatoire lors de la ré-optimisation.

4 *Conclusion*

Le modèle de coûts utilisant l'apprentissage en ligne conçu pour Nebula utilise plusieurs modèles d'apprentissage automatique pour répondre à la fois aux problématiques (i) de correction des estimations de cardinalité en sortie et de temps de réponse des résultats des sous-requêtes des plans d'exécution multi-clouds par les fournisseurs, (ii) de l'estimation les temps de transferts desdits résultats sur le réseau, (iii) et de la réévaluation des devis afin de

prendre en compte les effets de la ré-optimisation sur les plans d'exécution. Son introduction renforce la précision des devis présentés aux utilisateurs et la qualité du processus d'optimisation des requêtes multi-clouds, permettant *in fine* de gagner plus facilement la confiance des utilisateurs de Nebula. Le modèle de coûts permet par ailleurs de prendre en compte l'impact de l'optimisation dynamique lors du calcul des devis, cette dernière modifiant les plans d'exécution lorsque les sous-requêtes ont terminées leur exécution. La méthode d'optimisation dynamique présentée dans ce chapitre, conçue pour fonctionner avec la stratégie de recherche exhaustive, permet en effet au middleware de corriger une partie de ses choix sous-optimaux. Celle-ci pourrait être adaptée pour fonctionner avec la stratégie de recherche aléatoire, en réutilisant le contenu plans d'exécution générés lors de son exploration de l'espace de recherche.

Le modèle de coûts multi-cloud et la méthode d'optimisation dynamique présentées dans le présent chapitre permettent de diminuer la sous-optimalité des plans d'exécution multi-clouds respectivement de manière préventive et palliative. Il convient alors d'évaluer leur impact : celui-ci a été mis en lumière lors des expérimentations menées pendant cette thèse. Celles-ci sont présentées dans le chapitre suivant.

6

Implémentation et évaluation du middleware

1 *Introduction*

Les propositions présentées dans les chapitres précédents ont été implémentés au sein d'un prototype de middleware. Elles ont été évaluées en utilisant le JOB, un ensemble de requêtes formulées sur la base de données IMDb. Pour des raisons financières et de flexibilité, les fournisseurs DBaaS ont été simulés. Deux environnements expérimentaux ont été utilisés : le premier à petite échelle, utilisant des machines disponibles à l'IRIT, et le second à grande échelle, utilisant la grille de calcul nationale Grid'5000 [35].

Les résultats expérimentaux montrent que la sous-traitance des requêtes multi-clouds peut être plus efficace que la stratégie de téléchargement des données sur lesquelles elles portent suivi de leur exécution mono-cloud utilisée par les SGBDR multi-clouds de la littérature [37]. Ils montrent également que les devis produits par Nebula sont suffisamment précis pour que les utilisateurs puissent lui faire confiance. Concernant plus précisément l'évaluation des stratégies de recherche, il a été constaté que les attentes vis-à-vis des capacités de la stratégie aléatoire à produire des plans d'exécution

multi-clouds plus efficaces que ceux produits par la stratégie exhaustive sont satisfaites. Le modèle de coûts multi-cloud permet d'améliorer la précision des estimations sur les sous-requêtes, et est effectivement capable d'améliorer sa précision à mesure que les expérimentations avançaient grâce à son utilisation de l'apprentissage automatique en ligne. Il parvient également à prévoir l'impact de l'optimisation dynamique sur les devis. Celle-ci réussit en effet à améliorer l'efficacité des plans d'exécution multi-clouds qu'elle ré-optimise en diminuant la quantité totale de données qu'ils manipulent.

Ce chapitre est composé comme suit. La section 2 explique la manière dont les fournisseurs DBaaS ont été simulés, décrit l'implémentation de Nebula ainsi que notre protocole expérimental. Les résultats sont ensuite présentés et débattus dans la section 3. Enfin, ce chapitre se conclut avec la section 4.

2 *Description du dispositif expérimental*

Le volet expérimental de ces travaux de thèse a conduit à l'implémentation des propositions au sein d'un prototype de middleware, interrogeant des fournisseurs simulés développés pour l'occasion. La figure 6.1 illustre l'attention particulière donnée à la mise en œuvre pratique des propositions pendant cette thèse par la quantité de code produite [1]. Les données hébergées par les fournisseurs simulés sont des données réelles, issues de l'IMDb, interrogées par des requêtes issues d'un benchmark standard de l'état de l'art : le JOB [33]. L'environnement expérimental a également évolué pendant la thèse, passant d'un dispositif artisanal à petite échelle installé à l'IRIT à la grille de calcul nationale Grid'5000.

Cette section décrit l'implémentation des fournisseurs, du middleware ainsi que les spécificités des environnements expérimentaux. Les points de comparaison sont également présentés.

2.1 *Simulation des fournisseurs DBaaS*

Dans le cadre de nos expérimentations, les fournisseurs DBaaS ont été conçu comme des serveurs, implémentant une API HTTP en Python à l'aide de la librairie `flask` [3]. PostgreSQL a été utilisé comme SGBD sous-jacent.

Les instances de PostgreSQL ont été paramétrées selon les caractéristiques des serveurs hôtes. Les modifications effectuées consistent à adapter la mémoire tampon proportionnellement à la quantité de RAM disponible en définissant les paramètres `work_mem`, `maintenance_work_mem` et

1. Voir le code de Nebula, des fournisseurs et du déroulé des expériences ici : `https://framagit.org/DWojtowicz/nebula`.
2. Voir `https://github.com/AlDanial/cloc`
3. Voir `https://flask.palletsprojects.com/en/2.2.x/`

```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Python                          27           1465           1297           4405
SQL                              5              6              2            323
HTML                             2             13              0            235
YAML                            11             23             88            120
Bourne Shell                     6             33             19            104
CSS                              3             14              4             62
JavaScript                       2              6             12             57
-------------------------------------------------------------------------------
SUM:                            56           1560           1422           5306
-------------------------------------------------------------------------------
```

FIGURE 6.1 – Quantité de lignes de code produites pour l'implémentation du middleware et la configuration des machines virtuelles sur Grid'5000 selon le logiciel `cloc`[2].

`effective_cache_size` à respectivement un quart, un vingtième et la moitié de la RAM[4], et à donner aux lectures aléatoires un coût similaire à celui des lectures séquentielles dans le cas où les données sont stockées sur un SSD en définissant le paramètre `random_page_cost` à 1,1[5].

PostgreSQL calcule un coût abstrait pour les opérateurs des plans d'exécutions. Ce coût abstrait est convertit en secondes afin de pouvoir fournir des devis compréhensibles par les utilisateurs du middleware. Pendant nos expérimentations, cette conversion a été effectuée par les fournisseurs en divisant la somme du coût abstrait de chacun des opérateurs des plans d'exécution par 255, ce nombre ayant été déterminé empiriquement. Ils calculent le coût monétaire des opérations d'interrogation, de stockage et d'export comme décrit dans la section 2.2.3 du chapitre 5, c'est à dire proportionnellement à la quantité de données qu'elles manipulent. Afin de simplifier les calculs, on suppose que le stockage n'est payé qu'une fois et que son coût ne dépend pas du temps pendant lequel les données ont été hébergées chez le fournisseur.

Les estimations sur les sous-requêtes impliquant des données issues de plusieurs fournisseurs sont effectuées en utilisant des vues définissant ces dernières. Les relations dont elles dépendent chez les autres fournisseurs sont déclarées comme des `FOREIGN VIEW`. Celles-ci ne sont en revanche pas utilisées à l'exécution pour les transferts de données car les résultats d'une requête sur une `FOREIGN VIEW` sont rapatriés tuple par tuple depuis l'hébergeur. Ce fonctionnement nuit aux performances des transferts de données en raison de

4. Voir leur documentation ici : `https://www.postgresql.org/docs/14/runtime-config-resource.html`.

5. Voir sa documentation ici : `https://www.postgresql.org/docs/14/runtime-config-query.html`.

la latence entre chaque transfert. Afin d'accélérer les transferts, un export des résultats intermédiaires avec `pg_dump` [6] sur le fournisseur source est effectué. Ce fichier est ensuite téléchargé par le fournisseur destinataire et charge ensuite ces résultats intermédiaires avec `pg_restore` [7].

Par ailleurs, une interface Web permettant de faciliter le débogage a été développée en utilisant jQuery [8] et KNACSS [9].

2.2 *Implémentation de Nebula*

De la même manière que les fournisseurs simulés, Nebula a été conçu comme un serveur, implémenté en langage Python, sous la forme d'une API HTTP construite avec la librairie `flask`.

Le schéma multi-cloud est stocké dans une base de données SQLite [10], dont le schéma correspondant au modèle entité-association présenté plus tôt (cf. fig. 3.3) est relié à des objets Python à l'aide de l'ORM SQLAlchemy [11].

L'analyseur, composé d'un lexer et d'un parser, est implémenté à l'aide de la librairie SLY [12]. Il transforme les prédicats des requêtes mutli-cloud sous la forme d'un ensemble d'objets ; c'est à partir de cet ensemble que l'optimiseur va travailler. De plus, afin de simplifier l'implémentation pour les expérimentations sur Nebula, l'analyseur de Nebula limite l'expression des jointures à la syntaxe antérieure à la norme dite SQL-92 [13] n'utilisant pas le mot-clé `JOIN`, sans sous-requêtes imbriquées, comme présenté précédemment dans la figure 3.1b de l'exemple fil-rouge.

Nebula utilise la librairie `networkx` [14] [111] pour représenter les graphes et effectuer des opérations dessus. Ainsi, les plans d'exécution multi-clouds, le graphe de requêtes de la stratégie exhaustive et le graphe de l'espace de recherche de la stratégie itérative sont tous implémentés sous la forme de `DiGraph`.

De la même manière que pour les fournisseurs simulés, une interface Web permettant de faciliter le débogage a été développée en utilisant les mêmes technologies.

6. Cet outil permet d'extraire toute ou partie d'une base de données dans un fichier. Voir `https://www.postgresql.org/docs/14/app-pgdump.html`.
7. Cet outil permet de restaurer toute ou partie d'une base de données depuis un fichier créé avec `pg_dump`. Voir `https://www.postgresql.org/docs/14/app-pgrestore.html`.
8. Voir `https://jquery.com/`.
9. Voir `https://www.knacss.com/`.
10. Voir `https://www.sqlite.org/index.html`
11. Voir `https://www.sqlalchemy.org/`
12. Voir `https://sly.readthedocs.io/en/latest/index.html`.
13. Normes ANSI X3.135-1992 et ISO/IEC 9075 :1992.
14. Voir `https://networkx.org/`.

Nom	Modèle	Hyperparamètres	
$f_0^{(1)}$	`LinearRegressor`	`optimizer` : `AdaDelta` `loss` : `Cauchy`	`l2` : 2,95
$f_1^{(1)}$	`KNNRegressor`	`n_neighbors` : 6 `p` : 1	`aggregation_method` : `weighted_mean`
$f_1^{(2)}$	`KNNRegressor`	`n_neighbors` : 3 `p` : 1	`aggregation_method` : `weighted_mean`
m	`HoeffdingTreeClassifier`	`grace_period` : 20 `split_criterion` : `gini`	`split_confidence` : 0,55

TABLE 6.1 – Hyperparamètres du modèle **M** surchargeant la configuration par défaut de la librairie River [112]

2.2.1 *Modèle de coûts*

L'ensemble des modèles d'apprentissage automatiques étudiés dans le cadre de la thèse ont été implémentés à l'aide de la librairie River [15] [112] pour le langage Python. Celle-ci est conçue pour traiter des flux de données, et a comme particularité de traiter les données qu'elle manipule sous forme de dictionnaire Python.

Le tableau 6.1 liste les modèles de River utilisés dans le modèle de coûts **M** pour la correction des estimations de sélectivité et l'estimation du temps de réponse des sous-requêtes (cf. ch. 5 sect. 2.2.2), et détaille les hyperparamètres surchargés lors des expérimentations. Ceux-ci ont été déterminés empiriquement en effectuant une *grid search*, c'est-à-dire en explorant exhaustivement des combinaisons d'hyperparamètres fixés à l'avance optimisant une métrique. Dans notre cas, nous avons choisi l'erreur moyenne absolue (*Mean Absolute Error*, MSE). Un petit jeu de données a été construit pour le *tuning* des hyperparamètres à partir d'un sous-ensemble du JOB, dont les requêtes ont été exécutées sur une instance locale de la base de données IMDb. Le seuil de confiance pour la division des sommets dans l'arbre de Hoeffding du métamodèle m est fixé à un niveau intentionnellement haut pour accélérer l'apprentissage lors du démarrage à froid en permettant à l'arbre de réagir plus vite aux évolutions dans le flux de données de la vérité terrain.

Concernant la réévaluation des devis, deux modèles sont implémentés, détaillés par le tableau 6.2. Le coût monétaire est réévalué par une régression linéaire et le temps de réponse par une régression à la moyenne des 5 voisins.

Enfin, les temps de transfert sur le réseau sont estimés à l'aide d'une `LinearRegression`, estimant leur temps de réponse y à partir de la quantité

15. Voir `https://riverml.xyz/`.

Composante	Modèle	Hyperparamètres
Coût monétaire	`LinearRegressor`	`optimizer` : `AdaDelta`; `l2` : 2,95; `loss` : `Cauchy`
Temps de réponse	`KNNRegressor`	`n_neighbors` : 5

TABLE 6.2 – Hyperparamètres des modèles réévaluant les devis surchargeant la configuration par défaut de la librairie River [112]

de données x déplacée. La régression linéaire $y = \beta_0 + \beta_1 x$ est donc initialisée avec une `intercept_init` à $\beta_0 = 0$ et avec $\beta_1 = 1,25$ correspondant au débit théorique du réseau en gigaoctets par secondes.

2.2.2 *Parallélisation*

Afin de minimiser l'impact de la latence du réseau sur les performances du middleware, les portions de code utilisant ce dernier ont été parallélisées en utilisant des threads issus de la librairie standard de Python. Le *Global Interpreter Lock* (GIL) [16] est ainsi relâché dès que possible pour que d'autres appels réseaux ou d'autres traitements puissent être fait en attendant la réponse des serveurs interrogés.

Dans la stratégie de recherche exhaustive, les demandes d'estimations sur les sous-requêtes auprès des fournisseurs cloud sont faites en parallèle dans chaque niveau du DAG, et séquentiellement entre les niveaux. Dans la stratégie de recherche aléatoire, l'estimation de chaque plan du voisinage est réalisé en parallèle. Pour chacun des plans dans le voisinage, la même logique de parallélisation que pour la stratégie exhaustive est appliquée. Pendant l'optimisation dynamique, chaque agent est associé à un thread : le GIL est relevé pendant l'exécution des sous-requêtes et les transferts inter-fournisseurs et seules les opérations de mise à jour du DAG sont soumises à exclusion mutuelle. Enfin, un thread est associé à chaque opérateur des plans d'exécution multi-clouds.

2.3 *Environnement expérimental*

Deux environnements expérimentaux ont été utilisés pendant cette thèse. Le premier, à petite échelle, a été utilisé pour la première publication [113] afin de valider l'approche de la sous-traitance des requêtes multi-clouds. Le second, à grande échelle, a été utilisé pour la seconde publication [114] afin de renforcer la crédibilité de la simulation et de valider nos travaux lorsqu'appliqués à des requêtes plus complexes.

16. Il s'agit du mécanisme utilisé par l'interpréteur par défaut CPython de Python pour garantir la *thread-safety*. Le GIL a pour conséquence d'interdire l'exécution de code par plusieurs threads à la fois. Voir `https://docs.python.org/3/glossary.html#term-global-interpreter-lock`.

Fournisseur	Prix (ct/Go)			Thème	Nb. Rels.	Taille (Mo)	Caractéristiques techniques
	E	*Q*	*S*				
Horus	8,50	1,75	0,35	Acteurs	7	5 337	Intel Core i5-7440HQ @ 2.80 GHz RAM 1 × 8 GB DDR4 2400 MHz SSD NVMe M.2
Bastet	6,00	1,00	0,20	Films	7	3 501	Intel Core i3-3227U @ 1.90 GHz RAM 2 × 4 GB DDR3 1600 MHz SSD SATA
Montou	2,50	0,25	0,075	Compagnies	3	314	Intel Core i5-2520M @ 2.50 GHz RAM 1 × 4 GB DDR3 1333 MHz HDD SATA

TABLE 6.3 – Caractéristiques des fournisseurs. « E » signifie « Export », « Q » signifie « interrogation » et « S » signifie « Stockage ».

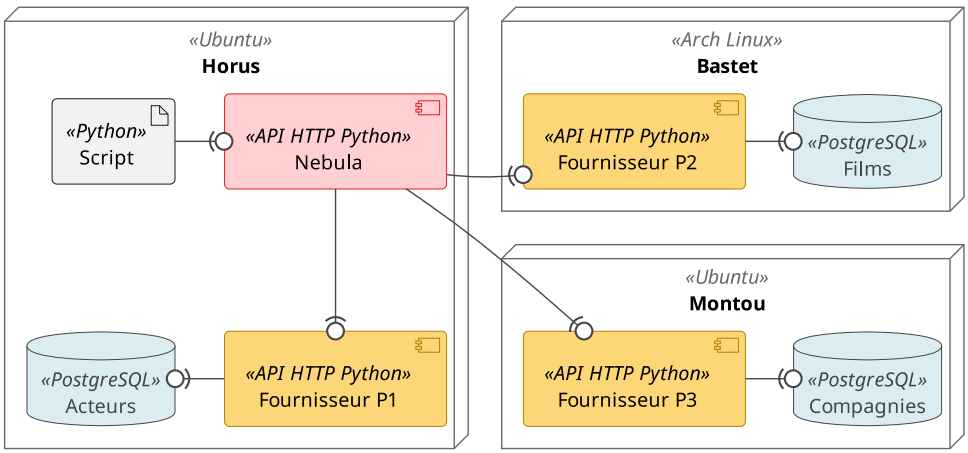


FIGURE 6.2 – Diagramme de déploiement de l'environnement expérimental à petite échelle

2.3.1 *Environnement à petite échelle à l'IRIT*

L'environnement à petite échelle servant de cadre pour les expérimentations de la première publication utilise trois ordinateurs portables. Leurs caractéristiques techniques, leur politique tarifaire et le sous-ensemble des relations de IMDb qu'ils hébergent sont détaillées dans le tableau 6.3. Comme illustré par le diagramme de déploiement de la figure 6.2, les scripts déroulant le benchmark sont exécutés sur la même machine que Nebula et qu'un fournisseur simulé. Ces machines communiquent au travers du réseau local de l'IRIT par tunnels SSH. La latence du réseau est donc presque négligeable pendant les expérimentations.

2.3.2 *Environnement à grande échelle avec Grid'5000*

Le déploiement sur Grid'5000 est similaire à celui effectué à l'IRIT. Une image Debian contenant le fournisseur simulé ainsi qu'une copie intégrale de toute la base de données IMDb est déployée sur les nœuds de calcul réservés pour les fournisseurs. Un unique noeud de calcul est réservé sur 15 clusters répartis sur les 8 sites de Grid'5000 [17]. Une image Debian contenant Nebula est déployée sur un nœud de calcul du cluster `nova` à Lyon [18]. Un nœud de calcul sur le cluster *econome* à Nantes est utilisé pour exécuter le script déroulant les expérimentations.

Chaque copie de la base de données est considérée indépendante des autres. Les requêtes du JOB sont utilisées comme patrons pour la génération d'un jeu de 551 requêtes interrogeant de 2 à 15 fournisseurs, dans lesquelles les fournisseurs sont identifiés. Plusieurs requêtes multi-clouds sont envoyées en parallèle au middleware, sous condition qu'elles n'interrogent pas les mêmes fournisseurs simulés afin d'éviter qu'elles n'impactent leurs performances mutuelles.

La politique tarifaire est la même pour tous les fournisseurs cloud simulés. Ils calculent le coût monétaire des opérations avec comme coefficients $\epsilon_{P_i}^{(Q)} =$ 1.75 ct/GB pour l'interrogation, $\epsilon_{P_i}^{(E)} =$ 8.5 ct/GB pour l'export et $\epsilon_{P_i}^{(S)} =$ 3.5 ct/GB pour le stockage.

2.4 *Points de comparaison*

Afin d'évaluer la pertinence de l'approche de sous-traitance des requêtes multi-clouds promue par Nebula, nous proposons de comparer les coûts monétaires et les performances des requêtes lorsqu'elles sont traitées avec une simulation d'une approche « naïve » par traitement local et une approche d'une traitement mono-cloud telle que mise en œuvre par le SGBDR multi-cloud comme SHAMC [37] présenté précédement (cf. ch. 2 sect. 2.1.4). La simulation de l'approche « naïve » se fait dans le cadre de l'environnement expérimental à petite échelle en calculant le coût du téléchargement du jeu de données. La simulation de SHAMC se fait en exécutant le benchmark sur seulement un des sites de la simulation.

Concernant le modèle de coûts, plusieurs points de comparaison ont été considérés. Premièrement, les modèles d'apprentissage automatique en ligne pour la correction de sélectivité recensés dans la littérature [101] ont été

17. Ces clusters sont `dahu` à Grenoble, `chetemi`, `chiclet`, `chifflet` et `chifflot` à Lille, `petitprince` à Luxembourg, `nova` et `taurus` à Lyon, `grisou` et `gros` à Nancy, `econome` à Nantes, `paranoia`, `parasilo` et `paravance` à Rennes et `uvb` à Sophia-Antipolis. Voir leurs spécification ici : `https://www.grid5000.fr/w/Hardware`.

18. La configuration des deux images est disponible ici : `https://framagit.org/DWojtowicz/my_recipes`.

écartés de la comparaison en raison de la nature des vecteurs de *features* qu'ils prennent en entrée. Celui-ci est en effet un vecteur *one-hot* [19] encodant les attributs impliqués par les requêtes. Ces modèles sont donc conçus pour fonctionner lorsque le schéma relationnel n'est pas modifié, ce qui n'est pas le cas dans le middleware car la taille du vecteur de *features* changerait lors de la matérialisation sous forme de relation des résultats intermédiaires produits par les sous-requêtes. Il s'allongerait ainsi en permanence, empêchant le modèle d'apprendre correctement car celui-ci ne pourrait jamais généraliser la vérité terrain issue des résultats intermédiaires. Ensuite, les modèles entraînés directement à partir du contenu des bases de données [115] ont également été écartés en raison du coût monétaire qu'engendrerait leur entraînement sur des données issues de fournisseurs DBaaS : celui-ci serait tellement important qu'il ne serait pas amortissable par le middleware. Enfin, nous avons retenu comme point de comparaison une implémentation en ligne de la régression aux k-voisins de l'état de l'art [95] pour la prédiction du temps de réponse des requêtes, initialisée à $k = 5$.

3 *Résultats expérimentaux*

Cette section reprend les résultats expérimentaux présentés dans les publications dont les travaux de cette thèse ont fait l'objet.

3.1 *Intérêt de la sous-traitance des requêtes multi-clouds*

L'alternative à la sous-traitance des requêtes multi-clouds serait de télécharger l'intégralité des données qu'elles interrogent en local ou bien chez un seul fournisseur cloud DBaaS. Le coût total de l'exécution du benchmark sur Nebula dans l'environnement à petite échelle est de 3,94±0,01 € et le temps total de l'exécution est de 66,4±0,2 minutes. En comparant ces résultats avec ceux d'une exécution mono-cloud sur un des fournisseurs présenté dans le tableau 6.4, on constate que la sous-traitance des requêtes multi-clouds permet d'obtenir des performances jusqu'à 50 % meilleures pour un coût monétaire plus faible ou équivalent. Cette différence s'explique par l'amortissement des coûts de réplication des données chez un fournisseur.

La figure 6.3 illustre ce phénomène, notamment à travers la partie grisée correspond au coût de réplication à amortir par l'exécution mono-cloud. L'amortissement est proportionnel au nombre de requêtes total à exécuter et à la quantité de données à répliquer. L'intérêt de la sous-traitance des requêtes multi-clouds est donc évident pour de petites charges de travail ou lorsque de grandes quantités de données doivent être répliquées ; il reste cependant

19. C'est à dire un vecteur de type B^n, donc de taille n tel que chacun de ses éléments est soit 0, soit 1.

Fournisseur	Coût (€)			Temps (min)		
	Répl.	*Requêtes*	*Total*	*Répl.*	*Requêtes*	*Total*
Horus	0,98	3,95	6,85	110,5	6,15	117,1
Bastet	0,97	2,26	5,16	95,0	8,5	103,5
Montou	0,95	0,56	3,46	> 600	465,4	> 600

TABLE 6.4 – Coût monétaire et temps de réponse d'une réplication totale des données sur un fournisseur suivie d'une exécution mono-cloud du benchmark.

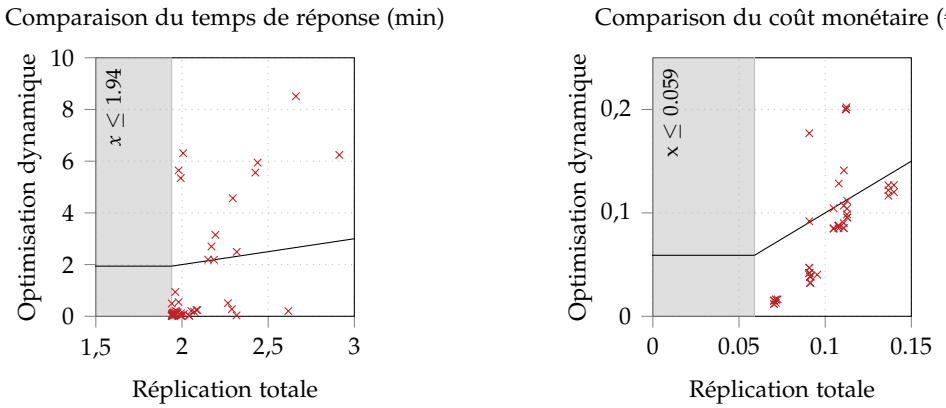


FIGURE 6.3 – Comparaison entre les résultats d'exécution des plans d'exécution multi-clouds lorsqu'elles sont optimisées dynamiquement et lorsqu'elles sont exécutées sur un seul fournisseur (Horus). La zone grise correspond au décalage dans les coûts de l'approche mono-cloud induits par la réplication.

à nuancer lorsqu'un grand nombre de requêtes doivent être exécutées, en particulier sur de petites quantités de données.

La réplication des données soulève la problématique de la mise à jour des réplicats. Celles-ci impliquent nécessairement des coûts supplémentaires, comme illustré par la figure 6.4. Elle montre l'évolution du coût monétaire total de la charge de travail à mesure qu'elle progresse suivant deux scénarios opposés : lorsque qu'il n'y a pas de changement dans les données, et lorsqu'elles changent fréquemment à tel point que les données sont répliquées à nouveau entre chaque requête. Dans le premier cas, les résultats expérimentaux tendent à confirmer qu'il existe bien un point au-delà duquel la sous-traitance des requêtes multi-clouds n'est plus rentable. Dans le second cas, le coût total de la charge de travail considérée être supérieur de deux ordres de grandeur à celui de Nebula. De plus, si les sources de données sont modifiées, il faut soit procéder à une nouvelle réplication complète,

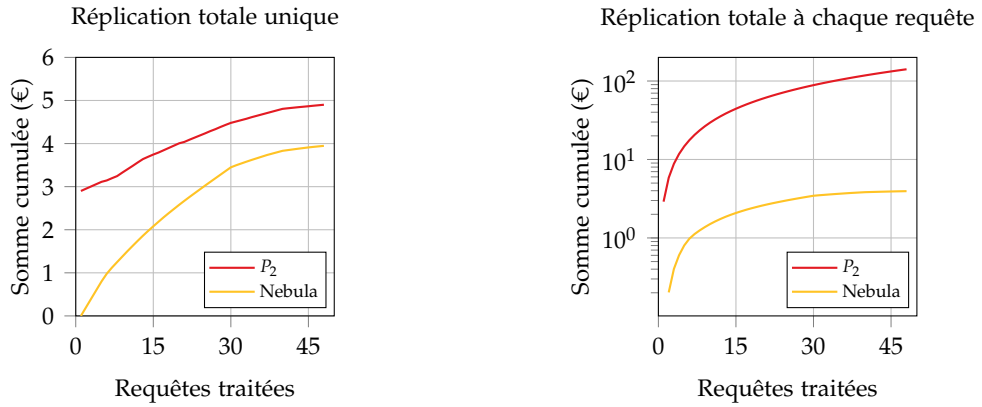


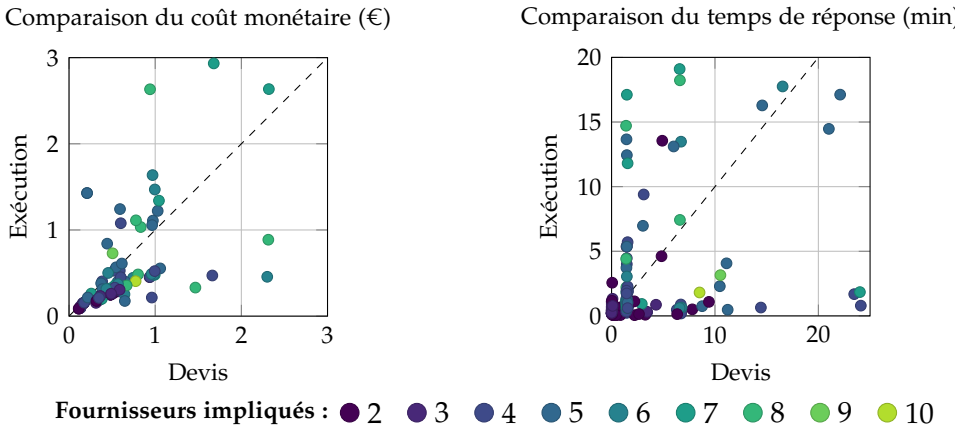
FIGURE 6.4 – Somme cumulée du coût monétaire m_Q engendré par le traitement des requêtes Q du benchmark sur Nebula et sur un fournisseur simulé (P_2), triés par valeurs croissantes de m_Q.

soit concevoir un mécanisme de mise à jour. Ce dernier point implique un travail de conception supplémentaire sur les métadonnées du schéma multi-cloud. Dans tous les cas, la sous-traitance des requêtes multi-clouds élimine ce besoin et offre en permanence l'accès aux données les plus fraîches à tout moment, contrairement à l'apprpoche suivie par le SGBDR multi-cloud SHAMC [37]. Cette observation rappelle la comparaison entre les systèmes d'intégration de données virtuels et les entrepôts de données, ces derniers devant trouver un équilibre entre la fraîcheur des données et les coûts de mise à jour.

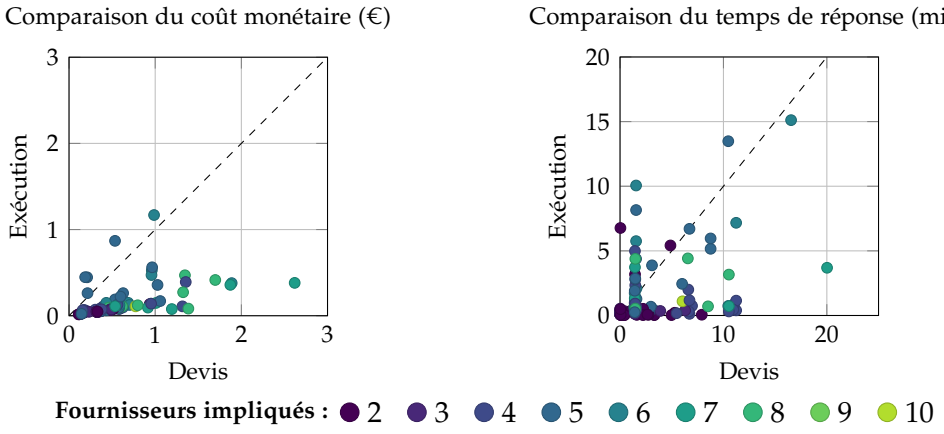
3.2 *Précision des devis*

La précision des devis calculés par la méthode aléatoire a été évaluée dans l'environnement à grande échelle [114]. La figure 6.5 montre la précision des devis calculés à l'aide de la stratégie aléatoire en termes de coûts monétaires et de temps de réponse avec deux configurations, à $<n = 1, p = 1>$ (c-à-d que seul le plan d'exécution linéaire initial est retourné par l'optimiseur, voir fig. 6.5a) et $<n = 300, p = 5>$ (c-à-d qu'au plus $n = 300$ plans d'exécution ont été estimés en tolérant au maximum $p = 5$ extensions de la frontière $\mathcal{F}$ de l'espace de recherche, cf. fig. 6.5b). Ces derniers paramètres sont un compromis entre ceux utilisés pour produire la figure 6.6 présentée ci-après.

Ces résultats sont conformes à nos attentes, et montrent que le composant de correction des devis du modèle de coûts est bien capable de généraliser ses estimations pour des plans d'exécution linéaires (cf. fig. 6.5a). En revanche, comme illustré par la figure 6.5b, il n'est pas capable de généraliser pour d'autres types de plans d'exécution. La faible précision est cependant à relativiser, car le coût monétaire réel des plans d'exécution est presque



(a) Plan d'exécution aléatoire initial



(b) Plans d'exécution trouvés avec $n = 300$ et $p = 5$

FIGURE 6.5 – Comparaison de la précision des devis calculés à partir du plan d'exécution initial généré aléatoirement avec ceux calculés à partir d'un plan d'exécution obtenu suite au processus d'optimisation.

toujours inférieur à celui du devis, et les temps de réponse sont généralement plus faibles à la fois dans les devis et dans la réalité.

3.3 *Impacts de la stratégie aléatoire*

Les stratégies de recherche ont été comparées dans l'environnement à grande échelle [114]. Comme illustré par la figure 6.6, et comme prévu théoriquement (cf. ch. 4 sect. 2.3), la stratégie de recherche aléatoire permet de calculer en un temps acceptable des devis pour des requêtes impliquant plus de $|\mathcal{P}_Q| = 4$ fournisseurs cloud. Ce n'est pas le cas de la stratégie exhaustive, qui a dépassé le temps imparti de 20 minutes dès que $|\mathcal{P}_Q| = 5$ fournisseurs ont été impliqués dans les requêtes.

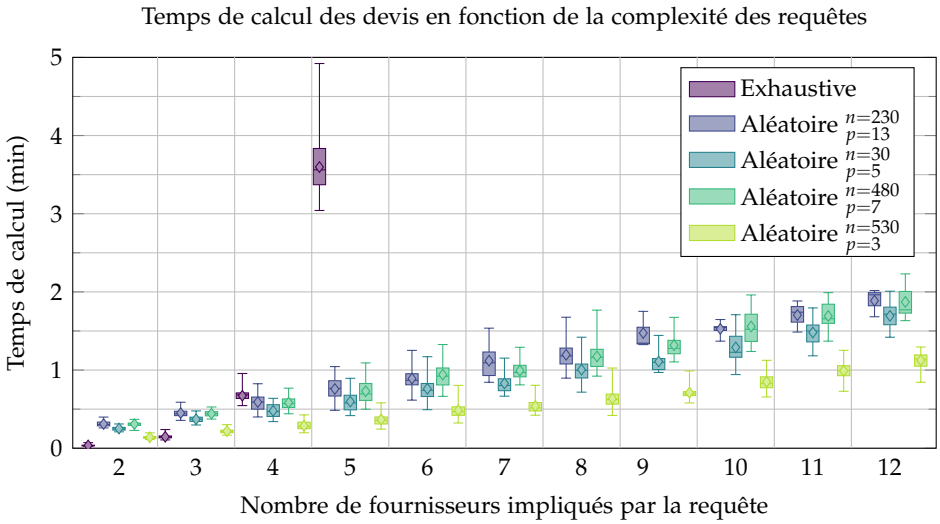


FIGURE 6.6 – Comparaison du temps de calcul du devis des requêtes multi-clouds en fonction de leur complexité, en utilisant la stratégie exhaustive ainsi que la stratégie aléatoire avec différents paramètres.

Plusieurs paramétrages de la stratégie exhaustive ont été testés, montrant que ses performances sont généralement plus influencées par le paramètre p permettant d'étendre la frontière $\mathcal{F}$ de l'espace de recherche que par la paramètre n limitant le nombre de plans d'exécution total explorés. Ces différences proviennent du potentiel de parallélisation de la stratégie de recherche aléatoire : les différents niveaux de profondeur dans les plans d'exécution doivent être estimés séquentiellement, tout comme les extensions successives de l'espace de recherche.

Quant à l'optimalité des plans d'exécution, la figure 6.7 montre que la stratégie aléatoire permet de touver des plans d'exécution presque systématiquement moins chers que ceux de la stratégie exhaustive, en particulier lorsque le nombre de fournisseurs impliqués par la requête grandit. En ce qui concerne les performances des plans d'exécution, elle se trouve également généralement améliorée. Ces résultats s'expliquent par la diminution de la profondeur des plans d'exécution multi-clouds permise par la stratégie aléatoire : en fusionnant des sous-requêtes pouvant hébergées sur le même fournisseur cloud, une étape d'écriture et de lecture des résultats intermédiaires est économisée, permettant à la fois de diminuer le coût monétaire général du plan d'exécution et d'éviter des temps d'attentes liés à l'accès physique aux données.

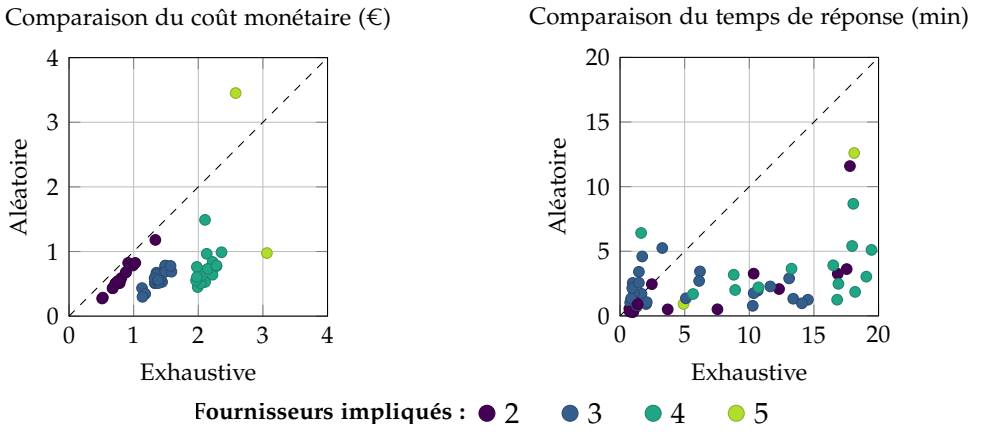


FIGURE 6.7 – Comparaison du coût monétaire et du temps de réponse des requêtes lorsqu'elles sont optimisées avec la stratégie exhaustive et la stratégie aléatoire.

3.4 *Impact du modèle de coûts*

Afin d'évaluer la précision des estimations produites par le modèle **M** du modèle de coûts, nous avons utilisé la *q-error* [116] définie dans l'équation 6.1, avec y une estimation, y' la vérité terrain et $\gamma = 10^{-4}$ (utilisé pour éviter des divisions par zéro). Le modèle de coûts a été évalué dans l'environnement à grande échelle [114].

$$q(y, y') = max(\frac{y+\gamma}{y'+\gamma}, \frac{y'+\gamma}{y+\gamma}) \tag{6.1}$$

La figure 6.8 compare la précision de notre modèle de coûts avec celle de celui du SGBD du fournisseur (appelé *baseline* par la suite) ainsi qu'une régression aux k-voisins de la littérature [95] en montrant l'évolution de leur q^{-1}-*error* moyenne [20] pendant les expérimentations. Dès que le métamodèle m a été autorisé à choisir des estimations à retourner, à partir de la 25e requête sur laquelle il a apprit, on note que notre modèle de coûts devient plus précis que la *baseline*. Après avoir ingéré environ un millier de points de données, le modèle devient également plus précis que celui de la littérature. Ces résultats montrent que le métamodèle tend à être capable de bien choisir le modèle duquel les estimations sont les plus précises. Lorsque c'est la *baseline* ou le modèle issu de la littérature (ce dernier étant de même nature que f_1^1 et f_1^2) qui semble être le plus précis, leurs estimations sont retournées : le modèle de coûts multi-cloud profite ainsi de leur précision lorsque les réestimations ne sont pas très précises.

L'impact du modèle de coûts est relativement positif, à la fois sur la précision des devis et l'optimalité des plans d'exécution, comme le montre la

20. La *q-erreur* a été inversée puis lissée pour des raisons de lisibilité.

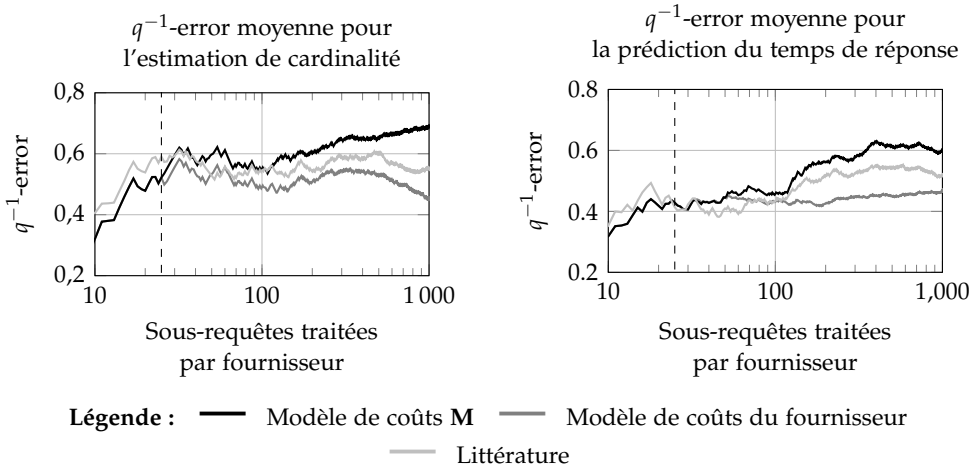


FIGURE 6.8 – Évolution de la q^{-1}-error moyenne du modèle de coûts multi-cloud **M** comparée à celle des fournisseurs simulés et à la littérature.

figure 6.9. Si l'impact de ce dernier sur le coût monétaire en lui-même semble relativement marginal, il est en revanche très positif sur le temps de réponse. Celui-ci est en effet diminué, mais les devis sont surestimés. Le modèle de coûts a donc eu un impact positif sur l'optimisation, car il a contribué à la découverte de plans d'exécution multi-clouds plus performants en corrigeant suffisamment les estimations sur le temps de réponse de leurs sous-requêtes pour qu'elles soient réalistes.

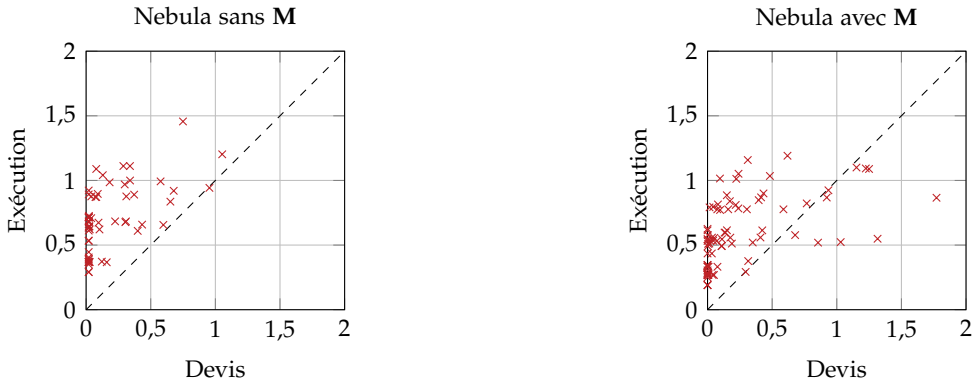
La prévention des sous-optimalités par correction des valeurs à partir desquelles raisonnent les stratégies de recherche semble être efficace.

3.5 *Apport de l'optimisation dynamique pour le respect des devis*

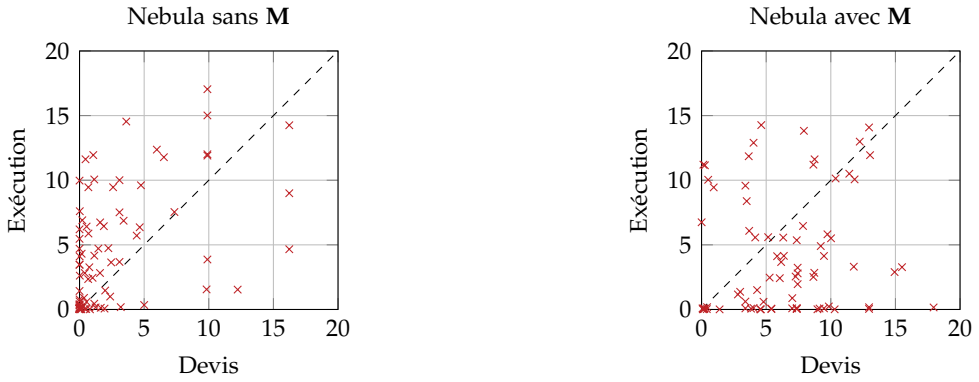
La méthode d'optimisation dynamique permettant de pallier à cette sous-optimalité dès que des sous-requêtes ont terminé leur exécution est également efficace. Celle-ci a été a été évaluée dans l'environnement à petite échelle [113].

Comme le montre la figure 6.10, à performance équivalente, l'optimisation dynamique produit des plans d'exécution multi-clouds qui sont moins chers à exécuter que leurs homologues initiaux optimisés statiquement. L'analyse de la différence relative entre les performances et le coût monétaire entre ces deux méthodes (cf. fig. 6.11), montre que ce sont en réalité la moitié des requêtes multi-clouds qui sont plus performantes lorsqu'optimisées dynamiquement. Le surcoût monétaire de l'optimisation statique est en moyenne le double de celui de l'exécution dynamique, peut monter jusqu'à 170 %, et n'est que rarement significativement négatif.

En analysant la nature des coûts monétaires engendrés par les deux méthodes (cf. fig. 6.12), on constate que la proportion des coûts d'interrogation



(a) Comparaison du coût monétaire (€)



(b) Comparaison du temps de réponse (min)

FIGURE 6.9 – Comparaison entre les prix (cf. fig. 6.9a) et les performances (cf. fig. 6.9b) dans les devis et les valeurs réelles d'exécution des requêtes multi-clouds avec et sans le modèle de coûts **M**.

est très légèrement plus faible dans les plans optimisés dynamiquement, et que les coûts d'export et de stockage sont un peu plus faibles. Si la répartition des coûts ne varie pas tellement entre l'optimisation statique et l'optimisation dynamique, c'est donc que la seconde fait des économies en parvenant à diminuer la quantité totale de données manipulées par le plan d'exécution. La figure 6.13 illustre bien ce phénomène : pour les trois-quart des requêtes multi-clouds du *benchmark*, la quantité de données déplacées par les plans optimisés statiquement est deux fois supérieure à celle des plans optimisés dynamiquement.

Dans un environnement multi-cloud reposant sur des fournisseurs DBaaS, le coût monétaire des requêtes est proportionnel à la quantité totale de données qu'ils manipulent. Par ailleurs, tout transfert de données entre fournisseurs cloud est facturé deux fois (à l'export et à l'import). Ces résultats expérimentaux étaient donc prévisibles, en connaissant la propension des

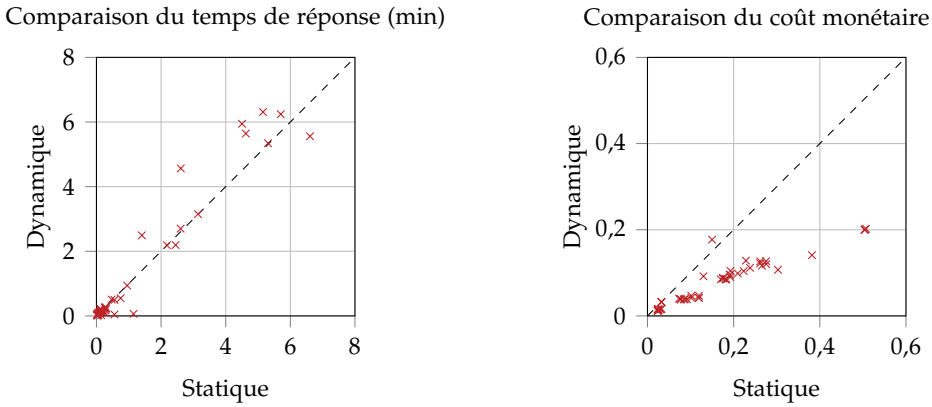


FIGURE 6.10 – Comparaison du temps de réponse et coût monétaire des requêtes entre leur exécution statique et leur exécution dynamique.

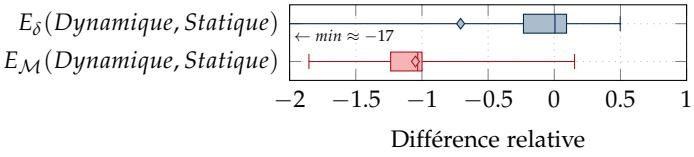


FIGURE 6.11 – Différence relative $E(x,y) = \frac{x-y}{x}$ du temps de réponse et du coût monétaire entre le plan d'exécution statique et le plan d'exécution dynamique des requêtes.

méthodes d'optimisation dynamiques à base d'agents à éviter les transferts sur le réseau.

Enfin, la figure 6.14, construite à partir des résultats expérimentaux obtenus dans l'environnement à petite échelle, illustre enfin que l'optimisation dynamique permet à Nebula de respecter, dans la plupart des cas, les devis qu'elle présente à ses utilisateurs. Si la précision des estimations de temps de réponse est encore améliorable, celle de l'estimation du coût monétaire est plutôt satisfaisante. En supposant une tolérance des utilisateurs à $\Lambda = 0.1$, c'est à dire une tolérance à $+10$ % vis-à-vis des estimations, leurs attentes

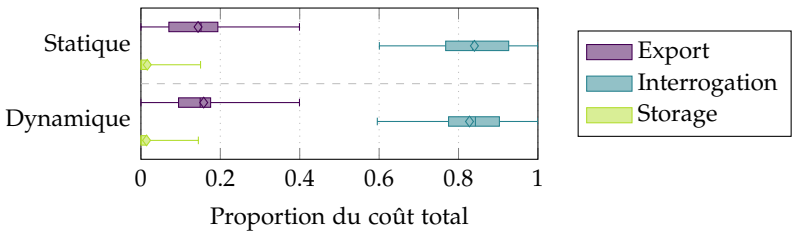
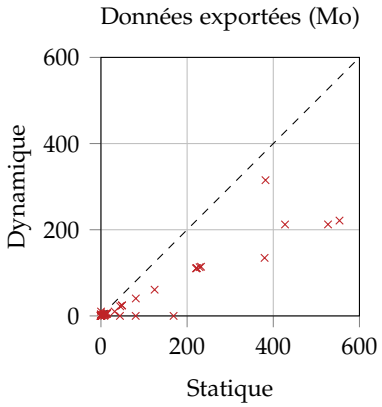


FIGURE 6.12 – Distribution de la décomposition des coûts monétaires des requêtes entre l'export, le stockage, l'interrogation.



Statistique	$S_S^{(E)} / S_D^{(E)}$
Moyenne	2.73
Minimum	0.01
Q_1	0.23
Médiane	0.55
Q_3	1.37
Maximum	66.42

FIGURE 6.13 – Comparaison de la quantité de données exportées lors du traitement de chaque requête multi-cloud entre la méthode statique ($S_S^{(E)}$) et dynamique ($S_D^{(E)}$). Le tableau présente le détail des statistiques sur le rapport $S_S^{(E)} / S_D^{(E)}$.

seraient satisfaites à 65 % pour le temps de réponse et à 78 % pour le coût monétaire.

3.6 *Synthèse*

Les résultats expérimentaux sont encourageants. Ils ont démontré dans deux environnements multi-clouds, l'un à petite échelle et l'autre à grande échelle, en utilisant des fournisseurs cloud simulés, la pertinence des différentes propositions de cette thèse. L'intérêt de la sous-traitance des requêtes multi-clouds a ainsi été mis en lumière, tout comme les bénéfices du passage d'une stratégie de recherche exhaustive à une stratégie aléatoire. L'apport du modèle de coûts et de la méthode d'optimisation dynamique respectivement pour prévenir et pallier à la sous-optimalité des plans d'exécution multi-clouds a également été mise en évidence.

4 *Conclusion*

Les expérimentations de cette thèse ont conduit à l'implémentation d'un prototype de Nebula, utilisant des fournisseurs simulés dans deux environnements différents. Les résultats permettent de valider chacune des propositions formulées. Nos avons montré que la sous-traitance des requêtes multi-clouds se révèle de plus en plus rentable à mesure que les données à interroger sont mises à jour. La stratégie de recherche exhaustive couplée à l'optimisation dynamique et à la ré-estimation des devis permet au middleware de tenir ses engagements vis-à-vis de ses utilisateurs dans la plupart des cas. L'explosion

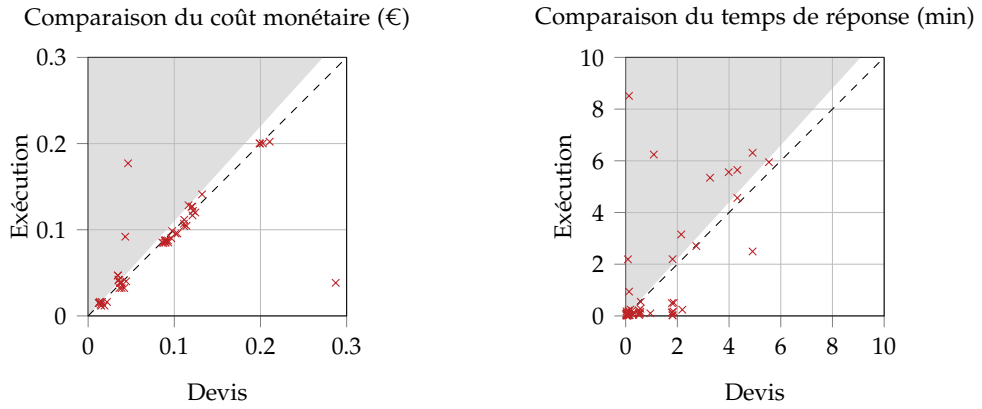


FIGURE 6.14 – Comparaison entre les coûts monétaires et les temps de réponse des devis avec ceux des plans d'exécution sous-jacents optimisés dynamiquement. La zone grisée correspond aux requêtes multi-clouds ne respectant pas un seuil de tolérance utilisateur de $\Lambda = 1{,}1$.

combinatoire qu'elle engendre a également été mise en évidence, et nous avons montré que la stratégie aléatoire, en s'appuyant sur le modèle de coûts multi-cloud dont nous avons illustré sa précision relative, est en mesure de produire des plans d'exécution moins chers et plus performants que la stratégie exhaustive pour des requêtes complexes.

L'intégralité de ces expérimentations repose sur des fournisseurs cloud simulés. Les résultats expérimentaux, même s'ils sont encourageants, traduisent les hypothèses sur la politique tarifaire que nous avons fait sur ces derniers. Il serait intéressant d'évaluer le comportement en cas de fluctuations dans les prix pratiqués par les fournisseurs, ou encore en considérant des politiques tarifaires basées sur le temps de calcul ou un prix négocié plutôt que sur la quantité de données manipulées.

7

Conclusion

1 *Synthèse des contributions*

La mise à disposition par diverses organisation de jeux de données publics en utilisant l'offre DBaaS de fournisseurs cloud est une pratique de plus en plus courante. Celle-ci invite à l'analyse croisée de ces jeux de données en formulant des requêtes multi-clouds, c'est à dire des requêtes SQL formulées sur un schéma multi-cloud agrégeant celui des bases de données sources et dont l'exécution serait sous-traitée auprès des fournisseurs cloud hébergeant les sources. C'est sur ces requêtes que ce sont concentrés ces travaux de thèse. Ils ont aboutit à la conception d'un middleware, nommé Nebula, permettant à ses utilisateurs de formuler de telles requêtes, afin d'orchestrer à sa place leur sous-traitance de la manière la plus optimale possible. Afin de permettre à ses utilisateurs de prévoir leurs dépenses monétaires, de connaître les performances de leurs requêtes et de pouvoir influencer sur les décisions d'optimisation du middleware, celui-ci leur présente des devis pour leurs requêtes multi-clouds.

Les devis sont calculés à partir d'estimations sur la cardinalité en sortie des résultats intermédiaires, le temps de réponse et le coût monétaire des sous-requêtes composant les requêtes multi-clouds. Ces sous-requêtes et leur ordre sont produits par l'optimiseur de Nebula en suivant une stratégie de recherche. Deux stratégies ont été proposées. La première est exhaustive ; elle explore la totalité des plans d'exécution multi-clouds linéaires. Elle permet ainsi de trouver de bons plans d'exécution pour des requêtes multi-clouds faisant intervenir peu de fournisseurs. En effet, sa complexité algorithmique est factorielle en le nombre de fournisseurs impliqués par la requête. Nous avons donc proposé une seconde stratégie de recherche, aléatoire et itérative, permettant d'optimiser des requêtes multi-clouds complexes. Celle-ci permet d'explorer une plus grande variété de plans d'exécution tout en protégeant Nebula du risque d'explosion combinatoire auquel la stratégie exhaustive l'expose.

Ces deux stratégies raisonnent à partir d'estimations produites par le modèle de coûts du SGBD *multi-tenant* sous-jacent à l'offre DBaaS des four-

nisseurs cloud, qui peuvent êtres fausses de plusieurs ordres de grandeur. Nous avons donc proposé un modèle de coûts multi-cloud permettant de corriger ces estimations, afin de permettre aux stratégies de recherche de trouver des plans d'exécution plus optimaux grâce à une meilleure précision de leurs données d'entrée. Ces corrections n'étant pas infaillibles, il faut s'attendre à ce que certains plans d'exécution multi-clouds restent sous-optimaux. Leur sous-optimalité peut être corrigée lorsqu'une de leurs sous-requête est exécutée, en ré-optimisant le plan d'exécution à l'aune de la cardinalité en sortie et des performances constatée. Il s'agit d'une méthode d'optimisation dynamique, qui modifie le plan d'exécution multi-cloud. Ces modifications sont anticipées au moment du calcul du devis par le modèle de coûts, qui est alors recalculé afin d'améliorer sa précision.

Le volet expérimental de cette thèse a conduit à l'implémentation d'un prototype de Nebula. Celui-ci a été évalué en utilisant le JOB, un ensemble de requêtes formulées sur la base de données IMDb. Les fournisseurs cloud ont été simulés à l'aide de PostgreSQL assorti d'une API similaire à celle des fournisseurs DBaaS. En utilisant un premier environnement expérimental à petite échelle, nous avons montré (i) que la sous-traitance des requêtes multi-clouds peut être plus économique que leur traitement local précédé d'un téléchargement, (ii) que la méthode d'optimisation dynamique permet de diminuer la sous-optimalité des plans d'exécution multi-clouds, et (iii) que le modèle de coûts permet d'anticiper l'impact de la ré-optimisation dans les devis. Puis, en utilisant la grille de calcul nationale Grid'5000 comme environnement expérimental à grande échelle, nous avons montré (i) que le modèle de coûts multi-cloud permet d'améliorer la précision des estimations sur les sous-requêtes et *in fine* la précision des devis, (ii) que la stratégie aléatoire d'optimiser des requêtes multi-clouds complexes en un temps acceptable, et (iii) que les plans d'exécution multi-clouds qu'elle trouve sont plus efficaces, tant du point de vue monétaire que des performances, que ceux produits par la stratégie de recherche exhaustive. Les résultats expérimentaux sont encourageants, et suggèrent que la voie vers une étude plus approfondie de l'optimisation de requêtes en environnements multi-clouds, et plus généralement à la conception de middlewares à cet effet, est libre.

2 *Perspectives*

Dans l'optique de continuer l'amélioration des techniques présentées précédemment, il serait intéressant d'adapter plus de techniques d'optimisation de requêtes distribuées. Par exemple, l'ajout de règles à la stratégie de recherche exhaustive permettrait par exemple d'implémenter des jointures à base de semi-jointures [117]. Ensuite, l'adaptation de méthodes comme le *query scrambling* [105] conduirait à la conception d'une méthode d'optimisa-

tion dynamique fonctionnant avec la stratégie de recherche aléatoire tout en améliorant la robustesse du middleware vis-à-vis d'imprévus côté fournisseurs. Enfin, permettre au middleware d'optimiser un ensemble de requêtes tout entier plutôt que chaque requête individuellement permettrait la mise en place de mécanismes de cache en maintenant des résultats intermédiaires chez certains fournisseurs, de sorte à diminuer encore plus les transferts de données, et donc d'améliorer les performances générales des requêtes tout en baissant leur coût monétaire individuel. Quand au modèle de coûts, il semble possible de continuer à complexifier les modèles d'apprentissage automatiques en ligne utilisées afin de continuer à améliorer la précision des devis et des corrections d'estimation, par exemple en transformant les modèles analytiques classiques des SGBD pour qu'ils adaptent la valeur de leurs paramètres en ligne [92].

Enfin, au-delà de l'extension des propositions existantes et de l'amélioration du middleware proposé, la démonstration en environnement simulé de l'intérêt de la sous-traitance des requêtes multi-clouds appelle sa validation en environnement réel. Des applications faisant la part-belle au « *BigData* » et dont l'essence même relève de l'analyse croisée de données, comme les *multi-omics* en biologie [118] ou encore l'analyse de données interdisciplinaires d'observation de la Terre [119], pourraient bénéficier de nos travaux sur l'optimisation de requêtes multi-clouds.

Liste des acronymes

DAAS *Data-as-a-Service*

DBAAS *Database-as-a-Service*

DAG Graphe orienté acyclique

GIL *Global Interpreter Lock*

IAAS *Infrastructure-as-a-Service*

IMDB *Internet Movie Database*

JOB *Join-Order Benchmark*

ORM *Object-Relationnal Mapping*

NOSQL *Not only SQL*

PAAS *Platform-as-a-Service*

PSLA *Personalised Service-Level Agreement*

SAAS *Software-as-a-Service*

SGD Système de Gestion Données

SGF Système de Fichiers

SGBD Système de Gestion de Base de Données

SGBDR SGBD Relationnel

SLA *Service-Level Agreement*

SLO *Service-Level Objective*

SMA Système Multi-Agent

SPJ *Select-Project-Join*

Liste des symboles

$\epsilon_P^{(E)}$	Coefficient de facturation pour l'export de données d'un fournisseur P (en euros par gigaoctets)
$\epsilon_P^{(Q)}$	Coefficient de facturation pour l'interrogation d'un fournisseur P (en euros par gigaoctets)
$\epsilon_P^{(S)}$	Coefficient de facturation pour le stockage de données d'un fournisseur P (en euros par gigaoctets)
Λ	Seuil de tolérance d'un utilisateur par rapport à un devis
$\mathcal{D}_Q$	L'ensemble des devis d'une requête multi-cloud Q
$\mathcal{E}_Q$	Ensemble des plans d'exécution utilisés pour calculer $\mathcal{D}_Q$
$\mathcal{F}$	Frontière de l'espace de recherche
$\mathcal{N}_v^+$	Ensemble des prédécesseurs directs ou indirects d'un sommet v dans un graphe
$\mathcal{N}_v^-$	Ensemble des successeurs directs ou indirects d'un sommet v dans un graphe
$\mathcal{P}$	Ensemble des fournisseurs recensés dans le schéma multi-cloud
$\mathcal{P}_Q$	L'ensemble des fournisseurs impliqués par une requête multi-cloud Q
$\mathcal{R}$	Ensemble des relations du schéma multi-cloud
$\mathcal{R}_Q$	L'ensemble des relations interrogées par une requête multi-cloud Q
$\mathcal{T}_Q$	L'ensemble des plans d'exécution pour une requête multi-cloud Q
Ω	Ensemble de fonctions objectif
ω	Une fonction objectif
τ_i	Type d'un prédicat i
φ	Un prédicat de jointure, de sélection ou de projection
φ_i	Prédicat d'une clause i
ϱ_i	Règle de transformation d'un plan d'exécution
A_i	Ensemble des attributs d'une relation ou d'une clause i
C_Q	Ensemble de clauses d'une requête Q

G_Q	Graphe orienté acyclique modélisant l'ensemble des plans d'exécution linéaires possibles pour Q
L_Q	Graphe de dépendance des clauses d'une requête multi-cloud Q
m_i	Coût monétaire d'un élément i
N_v^+	Ensemble des prédécesseurs directs d'un sommet v dans un graphe
N_v^-	Ensemble des successeurs directs d'un sommet v dans un graphe
P_i	Un fournisseur cloud
Q	Une requête SQL multi-cloud
q_i	Une sous-requête d'une requête multi-cloud Q
r_i	Temps de réponse d'un élément i
s_q	Cardinalité en sortie d'une requête q
T_0	Plan d'exécution multi-cloud initial pour la méthode aléatoire

Bibliographie

1. Stephens, Z. D. *et al.* Big Data : Astronomical or Genomical ? *PLOS Biology* **13,** 1-11 (juill. 2015).
2. Sozzi, M. *et al.* Economic comparison of satellite, plane and UAV-acquired NDVI images for site-specific nitrogen application : Observations from Italy. *Agronomy* **11,** 2098 (2021).
3. Özsu, M. T. & Valduriez, P. *Principles of Distributed Database Systems* 4e éd. (Springer, New York, États-Unis, 2011).
4. Garfinkel, S. *Architects of the information society : 35 years of the Laboratory for Computer Science at MIT* (MIT press, 1999).
5. Surbiryala, J. & Rong, C. *Cloud computing : History and overview* in *2019 IEEE Cloud Summit* (2019), 1-7.
6. Williams, B. *The economics of cloud computing* (Cisco Press, 2012).
7. Bômont, C. & Cattaruzza, A. Le cloud computing : de l'objet technique à l'enjeu géopolitique. Le cas de la France. *Herodote* **177178,** 149-163 (17 juin 2020).
8. Le Crosnier, H. À l'ère de l'"informatique en nuages". *Le Monde diplomatique* **653,** 19 (1er août 2008).
9. Herbst, N. R., Kounev, S. & Reussner, R. *Elasticity in Cloud Computing : What It Is, and What It Is Not* in *10th International Conference on Autonomic Computing (ICAC 13)* (USENIX Association, San Jose, CA, juin 2013), 23-27.
10. Mell, P. M. & Grance, T. *The NIST Definition of Cloud Computing* NIST SP-800-145 (National Institute of Standards et Technology, Gaithersburg, États-Unis, 2011). 3 p.
11. Kouki, Y. *Approche dirigée par les contrats de niveaux de service pour la gestion de l'élasticité du "nuage"* Thèse de doctorat (École nationale supérieure des mines de Nantes, Université Nantes Angers Le Mans, Nantes, France, 9 déc. 2013). 199 p.
12. Bayrak, E., Conley, J. P. & Wilkie, S. The Economics of Cloud Computing. *Economics* **27,** 203-230 (2011).
13. Agrawal, D., Abbadi, A. E., Emekci, F. & Metwally, A. *Database Management as a Service : Challenges and Opportunities* in *2009 IEEE 25th International Conference on Data Engineering* (Shanghai, Chine, 2009), 1709-1716.

14. Da Silveira Segalin, V., Dorneles, C. F. & Dantas, M. A. R. DBaaS Multi-tenancy, Auto-tuning and SLA Maintenance in Cloud Environments : a Brief Survey. *iSys-Brazilian Journal of Information Systems* **11,** 30-42 (2018).

15. Yin, S., Hameurlain, A. & Morvan, F. SLA Definition for Multi-Tenant DBMS and its Impact on Query Optimization. *IEEE Transactions on Knowledge and Data Engineering* **30,** 2213-2226 (nov. 2018).

16. Vuolo, F. *et al.* Data Service Platform for Sentinel-2 Surface Reflectance and Value-Added Products : System Use and Examples. *Remote Sensing* **8,** 938 (nov. 2016).

17. Beaujardière, J. D. L. NOAA Environmental Data Management. *Journal of Map & Geography Libraries* **12,** 5-27 (2 jan. 2016).

18. Koppad, S., Gkoutos, G. V. & Acharjee, A. Cloud computing enabled big multi-omics data analytics. *Bioinformatics and Biology Insights* **15,** 16 (2021).

19. Bates, J. W. & Aldred, M. US Patent 8,762,642 (2014).

20. Rao, R. J. US Patent 10,015,268 (2018).

21. Jayaraman, P. P. *et al.* Analytics-as-a-service in a multi-cloud environment through semantically-enabled hierarchical data processing. *Software : Practice and Experience* **47,** 1139-1156 (2017).

22. Alzain, M. A., Soh, B. & Pardede, E. *MCDB : Using Multi-clouds to Ensure Security in Cloud Computing* in *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing* (IEEE Computer Society, Sydney, Australie, déc. 2011), 784-791.

23. Iordache, A., Morin, C., Parlavantzas, N., Feller, E. & Riteau, P. *Resilin : Elastic MapReduce over Multiple Clouds* in *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing* (mai 2013), 261-268.

24. Wang, L. *et al.* G-Hadoop : MapReduce across distributed data centers for data-intensive computing. *Future Generation Computer Systems. Special Section : Recent Developments in High Performance Computing and Security* **29,** 739-750 (1er mars 2013).

25. Zhang, Q. *et al. Improving Hadoop Service Provisioning in a Geographically Distributed Cloud* in *2014 IEEE 7th International Conference on Cloud Computing* (juin 2014), 432-439.

26. Gouasmi, T., Louati, W. & Kacem, A. H. *Efficient distribution of mapreduce jobs for maximizing profit on federated cloud* in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing* (2018), 207-209.

27. Imai, S., Patterson, S. & Varela, C. A. *Cost-Efficient High-Performance Internet-Scale Data Analytics over Multi-cloud Environments* in *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (mai 2015), 793-796.

28. Tsamoura, E., Gounaris, A. & Tsichlas, K. *Multi-objective optimization of data flows in a multi-cloud environment* in *Proceedings of the Second Workshop on Data Analytics in the Cloud* (ACM, New York, États-Unis, 23 juin 2013), 6-10.

29. Gounaris, A., Karampaglis, Z., Naskos, A. & Manolopoulos, Y. A Bi-Objective Cost Model for Optimizing Database Queries in a Multi-Cloud Environment. *Journal of Innovation in Digital Ecosystems* **1,** 12-25 (1er déc. 2014).

30. Kllapi, H., Sitaridi, E., Tsangaris, M. M. & Ioannidis, Y. *Schedule Optimization for Data Processing Flows on the Cloud* in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data* (ACM, New York, États-Unis, 2011), 289-300.

31. Trummer, I. & Koch, C. *Approximation schemes for many-objective query optimization* in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data* (2014), 1299-1310.

32. Ibaraki, T. & Kameda, T. On the optimal nesting order for computing *N*-relational joins. *ACM Transactions on Database Systems* **9,** 482-502 (1er sept. 1984).

33. Leis, V. *et al.* How good are query optimizers, really ? *Proceedings of the VLDB Endowment* **9,** 204-215 (1er nov. 2015).

34. Halford, M. *Statistical learning for selectivity estimation in relational databases* Thèse de doctorat (Université Toulouse III - Paul Sabatier, 28 sept. 2020).

35. Balouek, D. *et al. Adding Virtualization Capabilities to Grid'5000* RR-8026 (Inria Grenoble – Rhône-Alpes, Montbonnot-Saint-Martin, France, 2012). 20 p.

36. Vukolić, M. The Byzantine Empire in the Intercloud. *SIGACT News* **41,** 105-111 (sept. 2010).

37. Wang, L., Yang, Z. & Song, X. SHAMC : A Secure and highly available database system in multi-cloud environment. *Future Generation Computer Systems* **105,** 873-883 (2020).

38. Hill, Z. & Humphrey, M. *CSAL : A Cloud Storage Abstraction Layer to Enable Portable Cloud Applications* in *2010 IEEE Second International Conference on Cloud Computing Technology and Science* (2010), 504-511.

39. D'Andria, F., Bocconi, S., Cruz, J. G., Ahtes, J. & Zeginis, D. *Cloud4SOA : Multi-Cloud Application Management Across PaaS Offerings* in *Proceedings of the 2012 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing* (IEEE Computer Society, États-Unis, 2012), 407-414.

40. Bessani, A., Correia, M., Quaresma, B., André, F. & Sousa, P. DepSky : Dependable and Secure Storage in a Cloud-of-Clouds. *ACM Transactions on Storage* **9,** 1-33 (1er nov. 2013).

41. Wu, Z., Butkiewicz, M., Perkins, D., Katz-Bassett, E. & Madhyastha, H. V. *SPANStore : Cost-Effective Geo-Replicated Storage Spanning Multiple Cloud Services* in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (ACM, Farminton, Pennsylvania, 3 nov. 2013), 292-308.

42. Rafique, A., Van Landuyt, D., Reniers, V. & Joosen, W. *Towards an Adaptive Middleware for Efficient Multi-Cloud Data Storage* in *Proceedings of the 4th Workshop on CrossCloud Infrastructures & Platforms* (ACM, Belgrade, Serbie, 23 avr. 2017), 1-6.

43. Rafique, A., Van Landuyt, D., Truyen, E., Reniers, V. & Joosen, W. SCOPE : self-adaptive and policy-based data management middleware for federated clouds. *Journal of Internet Services and Applications* **10,** 2 (30 jan. 2019).

44. Papaioannou, T. G., Bonvin, N. & Aberer, K. *Scalia : An adaptive scheme for efficient multi-cloud storage* in *SC'12 : Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (2012), 1-10.

45. Li, M., Qin, C. & Lee, P. P. C. *CDStore : Toward Reliable, Secure, and Cost-Efficient Cloud Storage via Convergent Dispersal* in (2015), 111-124.

46. Dobre, D., Viotti, P. & Vukolić, M. *Hybris : Robust Hybrid Cloud Storage* in *Proceedings of the ACM Symposium on Cloud Computing* (ACM, New York, États-Unis, 3 nov. 2014), 1-14.

47. Fabian, B., Ermakova, T. & Junghanns, P. Collaborative and secure sharing of healthcare data in multi-clouds. *Inf. Syst.* **48,** 132-150 (2015).

48. Han, S. *et al. {MetaSync} : File Synchronization Across Multiple Untrusted Storage Services* in *2015 USENIX Annual Technical Conference (USENIX ATC 15)* (USENIX Association, Santa Clara, CA, juill. 2015), 83-95.

49. Akintoye, S. & Bagula, A. Lightweight Cloud Storage Systems : Analysis and Performance Evaluation. *International Journal of Scientific & Engineering Research (IJSER)* **9,** 1447-1454 (déc. 2018).

50. *BlueSky : A Cloud-Backed File System for the Enterprise* in *10th USENIX Conference on File and Storage Technologies (FAST 12)* (USENIX Association, San Jose, CA, fév. 2012).

51. Selvakumar, C., Rathanam, G. J. & Sumalatha, M. *PDDS-Improving cloud data storage security using data partitioning technique* in *2013 3rd IEEE International Advance Computing Conference (IACC)* (2013), 7-11.

52. Chen, H. C., Hu, Y., Lee, P. P. & Tang, Y. NCCloud : A Network-Coding-Based Storage System in a Cloud-of-Clouds. *IEEE Transactions on Computers* **63,** 31-44 (jan. 2014).

53. Bessani, A. *et al. SCFS : A Shared Cloud-backed File System* in *2014 USENIX Annual Technical Conference (USENIX ATC 14)* (USENIX Association, Philadelphia, PA, juin 2014), 169-180.

54. Cachin, C., Haas, R. & Vukolic, M. *Dependable storage in the intercloud* RZ 3783 (IBM research, Zurich, Suisse, 2010). 6 p.

55. Zhang, Q. *et al.* CHARM : A Cost-Efficient Multi-Cloud Data Hosting Scheme with High Availability. *IEEE Transactions on Cloud Computing* **3,** 372-386 (2015).

56. Bermbach, D., Klems, M., Tai, S. & Menzel, M. *MetaStorage : A Federated Cloud Storage System to Manage Consistency-Latency Tradeoffs* in *2011 IEEE 4th International Conference on Cloud Computing* (IEEE Computer Society, Melbourne, Australie, juill. 2011), 452-459.

57. Dayarathna, M. & Suzumura, T. *Towards Scalable Distributed Graph Database Engine for Hybrid Clouds* in *Proceedings of the 5th International Workshop on Data-Intensive Computing in the Clouds* (IEEE Press, La Nouvelle-Orléans, États-Unis, 2014), 1-8.

58. Angelou, E., Papailiou, N., Konstantinou, I., Tsoumakos, D. & Koziris, N. *Automatic Scaling of Selective SPARQL Joins Using the TIRAMOLA System* in *Proceedings of the 4th International Workshop on Semantic Web Information Management* (Association for Computing Machinery, Scottsdale, États-Unis, 2012).

59. Tsoumakos, D., Konstantinou, I., Boumpouka, C., Sioutas, S. & Koziris, N. *Automated, Elastic Resource Provisioning for NoSQL Clusters Using TIRAMOLA* in *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing* (2013), 34-41.

60. Kassela, E., Boumpouka, C., Konstantinou, I. & Koziris, N. *Automated workload-aware elasticity of NoSQL clusters in the cloud* in *2014 IEEE International Conference on Big Data (Big Data)* (2014), 195-200.

61. Fetai, I., Murezzan, D. & Schuldt, H. *Workload-driven adaptive data partitioning and distribution — The Cumulus approach* in *2015 IEEE International Conference on Big Data (Big Data)* (2015), 1688-1697.

62. Rafique, A., Van Landuyt, D. & Joosen, W. Persist : Policy-based data management middleware for multi-tenant saas leveraging federated cloud storage. *Journal of Grid Computing* **16,** 165-194 (2018).

63. Iordache, A., Morin, C., Parlavantzas, N. & Riteau, P. *Resilin : Elastic MapReduce over Multiple Clouds* RR-8081 (Inria Centre Rennes - Bretagne Atlantique, Rennes, France, 2012). 17 p.

64. Iordache, A., Morin, C., Parlavantzas, N., Feller, E. & Riteau, P. *Resilin : Elastic MapReduce over Multiple Clouds* in *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing* (2013), 261-268.

65. Mandal, A. *et al. Provisioning and Evaluating Multi-domain Networked Clouds for Hadoop-based Applications* in *2011 IEEE Third International Conference on Cloud Computing Technology and Science* (2011), 690-697.

66. Zhang, Q. *et al. Improving Hadoop Service Provisioning in a Geographically Distributed Cloud* in *2014 IEEE 7th International Conference on Cloud Computing* (2014), 432-439.

67. Wang, L. *et al.* G-Hadoop : MapReduce across distributed data centers for data-intensive computing. *Future Generation Computer Systems* **29.** Special Section : Recent Developments in High Performance Computing and Security, 739-750 (2013).

68. Van Aken, D., Pavlo, A., Gordon, G. J. & Zhang, B. *Automatic Database Management System Tuning Through Large-Scale Machine Learning* in *Proceedings of the 2017 ACM International Conference on Management of Data* (Association for Computing Machinery, Chicago, États-Unis, 2017), 1009-1024.

69. Stonebraker, M., Held, G., Wong, E. & Kreps, P. The design and implementation of INGRES. *ACM Transactions on Database Systems* **1,** 189-222 (1er sept. 1976).

70. Astrahan, M. M. *et al.* System R : Relational Approach to Database Management. *ACM Trans. Database Syst.* **1,** 97-137 (juin 1976).

71. Kambayashi, Y., Yoshikawa, M. & Yajima, S. *Query processing for distributed databases using generalized semi-joins* in *Proceedings of the 1982 ACM SIGMOD international conference on Management of data* (ACM, New York, États-Unis, 2 juin 1982), 151-160.

72. Özsu, M. T. & Valduriez, P. *Principles of Distributed Database Systems* 4e éd. (Springer, New York, États-Unis, 2011).

73. Wong, E. & Youssefi, K. Decomposition — a Strategy for Query Processing. *ACM Transactions on Database Systems (TODS)* **1,** 223-241 (1er sept. 1976).

74. Ortiz, J., de Almeida, V. T. & Balazinska, M. *Changing the Face of Database Cloud Services with Personalized Service Level Agreements* in *CIDR* (CIDR, Asilomar, États-Unis, 2015), 13.

75. Ganguly, S., Hasan, W. & Krishnamurthy, R. Query Optimization for Parallel Execution. *SIGMOD Record* **21,** 9-18 (juin 1992).

76. Uspensky, J. V. *Introduction to mathematical probability* (McGraw-Hill Book Company, New York, 1937).

77. Swami, A. *Optimization of large join queries : combining heuristics and combinatorial techniques* in *Proceedings of the 1989 ACM SIGMOD international conference on Management of data* (ACM, New York, États-Unis, 1er juin 1989), 367-376.

78. Ioannidis, Y. E. & Wong, E. *Query optimization by simulated annealing* in *Proceedings of the 1987 ACM SIGMOD international conference on Management of data* (ACM, New York, États-Unis, 1er déc. 1987), 9-22.

79. Hameurlain, A., Basex, P. & Morvan, F. *Traitement parallèle dans les bases de données relationnelles : concepts, méthodes et applications* (Cépaduès, 1996).

80. Bottou, L. & Le Cun, Y. *Large Scale Online Learning* in *NIPS 2003* **16** (Vancouver, Canada, 2003), 217-224.

81. Wolpert, D. H. Stacked generalization. *Neural Networks* **5,** 241-259 (1er jan. 1992).

82. Stillger, M., Lohman, G., Markl, V. & Kandil, M. *LEO – DB2's LEarning Optimizer* in *VLDB* **1** (VLDB Endowment, Rome, Italie, sept. 2001), 19-28.

83. Selinger, P. G., Astrahan, M. M., Chamberlin, D. D., Lorie, R. A. & Price, T. G. *Access path selection in a relational database management system* in *Proceedings of the 1979 ACM SIGMOD international conference on Management of data* (ACM, New York, États-Unis, 30 mai 1979), 23-34.

84. Leis, V. *et al.* Query optimization through the looking glass, and what we found running the join order benchmark. *The VLDB Journal* **27,** 643-668 (2018).

85. Dean, J. *Machine learning for systems and systems for machine learning* in *Presentation at 2017 Conference on Neural Information Processing Systems* (2017).

86. Krishnan, S., Yang, Z., Goldberg, K., Hellerstein, J. & Stoica, I. Learning to Optimize Join Queries With Deep Reinforcement Learning. *arXiv :1808.03196 [cs]* (10 jan. 2019).

87. Marcus, R. *et al.* Neo : A Learned Query Optimizer. *Proceedings of the VLDB Endowment* **12,** 1705-1718 (1er juill. 2019).

88. Zhu, R. *et al. Learned Query Optimizer : At the Forefront of AI-Driven Databases* in *Proceedings of the 25th International Conference on Extending Database Technology, EDBT 2022, Edinburgh, UK, March 29 - April 1, 2022* (OpenProceedings.org, 2022), 1-4.

89. Wu, C. *et al.* Towards a learning optimizer for shared clouds. *Proceedings of the VLDB Endowment* **12,** 210-222 (1er nov. 2018).

90. Sun, J. & Li, G. An End-to-End Learning-based Cost Estimator. *arXiv :1906.02560 [cs]* (6 juin 2019).

91. Wang, Y. *et al. Monotonic Cardinality Estimation of Similarity Selection : A Deep Learning Approach* in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (ACM, New York, États-Unis, 11 juin 2020), 1197-1212.

92. Wu, W. *et al. Predicting query execution time : Are optimizer cost models really unusable ?* in *2013 IEEE 29th International Conference on Data Engineering (ICDE)* (avr. 2013), 1081-1092.

93. Gupta, C., Mehta, A. & Dayal, U. *PQR : Predicting Query Execution Times for Autonomous Workload Management* in *2008 International Conference on Autonomic Computing* (juin 2008), 13-22.

94. Akdere, M., Çetintemel, U., Riondato, M., Upfal, E. & Zdonik, S. B. *Learning-based Query Performance Modeling and Prediction* in *2012 IEEE 28th International Conference on Data Engineering* (avr. 2012), 390-401.

95. Kleerekoper, A., Navaridas, J. & Lujan, M. Can the Optimizer Cost be Used to Predict Query Execution Times ? *arXiv :1905.00774 [cs]* (2 mai 2019).

96. Marcus, R. *et al.* Bao : Making Learned Query Optimization Practical. *SIGMOD Record* **51,** 6-13 (juin 2022).

97. Kiran, M., Murphy, P., Monga, I., Dugan, J. & Baveja, S. S. *Lambda architecture for cost-effective batch and speed big data processing* in *2015 IEEE International Conference on Big Data (Big Data)* (2015), 2785-2792.

98. Sculley, D. *et al. Hidden Technical Debt in Machine Learning Systems* in *Advances in Neural Information Processing Systems* (éd. Cortes, C., Lawrence, N., Lee, D., Sugiyama, M. & Garnett, R.) **28** (Curran Associates, Inc., 2015).

99. Lin, J. The Lambda and the Kappa. *IEEE Internet Computing* **21,** 60-66 (2017).

100. Kipf, A. *et al. Estimating Cardinalities with Deep Sketches* in *Proceedings of the 2019 International Conference on Management of Data* (Association for Computing Machinery, Amsterdam, Pays-Bas, 2019), 1937-1940.

101. Halford, M., Saint-Pierre, P. & Morvan, F. Selectivity correction with online machine learning. *ArXiv* **abs/2009.09884** (2020).

102. Marcus, R. & Papaemmanouil, O. *Deep Reinforcement Learning for Join Order Enumeration* in *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management* (ACM, New York, États-Unis, 10 juin 2018), 1-4.

103. Ortiz, J., Balazinska, M., Gehrke, J. & Keerthi, S. S. *Learning State Representations for Query Optimization with Deep Reinforcement Learning* in *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning* (ACM, Houston, États-Unis, 16 juin 2018), 4.

104. Hellerstein, J. M. *et al.* Adaptive Query Processing : Technology in Evolution. *IEEE Bulletin of the Technical Committee on Data Engineering* **23,** 7-18 (juin 2000).

105. Amsaleg, L., Tomasic, A., Franklin, M. & Urhan, T. *Scrambling query plans to cope with unexpected delays* in *Fourth International Conference on Parallel and Distributed Information Systems* (IEEE Computer Society, Miami Beach, États-Unis, déc. 1996), 208-219.

106. Morvan, F., Hussein, M. & Hameurlain, A. *Mobile agent cooperation methods for large scale distributed dynamic query optimization* in *14th International Workshop on Database and Expert Systems Applications, 2003. Proceedings.* (Prague, Tchéquie, sept. 2003), 542-547.

107. Arpaci-Dusseau, R. H. *et al. Cluster I/O with River : making the fast case common* in *Proceedings of the sixth workshop on I/O in parallel and distributed systems* (ACM, New York, États-Unis, 1er mai 1999), 10-22.

108. Avnur, R. & Hellerstein, J. M. *Eddies : continuously adaptive query processing* in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data* (ACM, Dallas, États-Unis, 16 mai 2000), 261-272.

109. Arcangeli, J.-P., Morvan, F., Hameurlain, A. & Migeon, F. Mobile Agents Based Self-Adaptive Join for Wide-Area Distributed Query Processing. *Journal of Database Management* **15,** 25-44 (2004).

110. Agrawal, D. *et al. RHEEM : Enabling Cross-Platform Data Processing* in *Proceedings of the VLDB Endowment* **11** (VLDB Endowment, Rio de Janeiro, Brésil, août 2018), 14.

111. Hagberg, A., Swart, P. & S Chult, D. *Exploring network structure, dynamics, and function using networkx* LA-UR-08-05495 ; LA-UR-08-5495 (Los Alamos National Lab. (LANL), Los Alamos, NM (United States), 1er jan. 2008).

112. Montiel, J. *et al.* River : machine learning for streaming data in Python. *Journal of Machine Learning Research* **22,** 1-8 (2021).

113. Wojtowicz, D. T., Yin, S., Morvan, F. & Hameurlain, A. *Cost-Effective Dynamic Optimisation for Multi-Cloud Queries* in *CLOUD 2021* (IEEE Computer Society, Chicago, États-Unis, 1er sept. 2021), 387-397.

114. Wojtowicz, D. T., Yin, S., Martinez-Gil, J., Morvan, F. & Hameurlain, A. *Multi-Cloud Query Optimisation with Accurate and Efficient Quoting* in *BigData 2022* (IEEE Computer Society, 2022).

115. Han, Y. *et al.* Cardinality Estimation in DBMS : A Comprehensive Benchmark Evaluation. *arXiv :2109.05877 [cs]* (15 sept. 2021).

116. Moerkotte, G., Neumann, T. & Steidl, G. Preventing bad plans by bounding the impact of cardinality estimation errors. *Proceedings of the VLDB Endowment* **2,** 982-993 (1er août 2009).

117. Valduriez, P. & Gardarin, G. Join and Semijoin Algorithms for a Multiprocessor Database Machine. *ACM Trans. Database Syst.* **9,** 133-161 (mars 1984).

118. Subramanian, I., Verma, S., Kumar, S., Jere, A. & Anamika, K. Multi-omics Data Integration, Interpretation, and Its Application. *Bioinformatics and Biology Insights* **14,** 1177932219899051 (2020).

119. Backeberg, B. *et al.* An open compute and data federation as an alternative to monolithic infrastructures for big Earth data analytics. *Big Earth Data,* 1-19 (2022).

Ce document a été rédigé avec LaTeX en utilisant le type de document `classicthesis`. Les illustrations de ce manuscrit ont été réalisés avec Ti*k*Z, PGFPLOTS et PlantUML.

—

Document produit le 18 avril 2023.