Analyse et traitement de flux de données à large échelle

Yann Busnel

École Doctorale Matisse

Habilitation à diriger des recherches

présentée devant l'École Normale Supérieure de Rennes
sous le sceau de l'Université Bretagne Loire

par

Yann Busnel

Analyse et traitement de flux de données à large échelle

Discipline : Informatique et applications (Section CNU 27)

Soutenue le 5 décembre 2016 devant un jury composé de :

Président	**M. Luc Bougé**, Professeur	(ENS Rennes, France)
Rapporteur	**M. Pascal Felber**, Professeur	(Université de Neuchâtel, Suisse)
Rapporteur	**M. Nicolas Hanusse**, Directeur de recherche	(CNRS, Bordeaux, France)
Rapporteur	**Mme Sandrine Vaton**, Professeure	(Télécom Bretagne, Brest, France)
Examinateur	**M. Claude Jard**, Professeur	(Université de Nantes, France)
Examinateur	**M. Sébastien Tixeuil**, Professeur	(UPMC, Paris, France)




Travaux effectués au sein des laboratoires suivants, classés historiquement

Middleware Laboratory (MIDLAB)

au sein de la Sapienza Università di Roma, Rome, Italie

Laboratoire Informatique de Nantes Atlantique (LINA)

au sein de l'Université de Nantes, France

Centre de Recherche en Statistique et Economie (CREST)

au sein de l'ENSAI, Bruz, France

Centre de Recherche Inria Rennes – Bretagne Atlantique

sur la campus de Beaulieu de Rennes, France

Résumé

Ce manuscrit propose un panorama synthétique d'une partie de mes travaux de recherche effectués depuis ma soutenance de doctorat. Ce document ne prétend pas représenter l'intégralité de ceux-ci, mais se focalise sur l'extraction d'informations pertinentes dans des flux de données massifs et potentiellement infinis.

Dans un premier temps, le contexte général lié aux flux de données à grande échelle est présenté par le prisme de la diversité des problématiques rencontrées. Après avoir présenté les divers modèles de flux et les outils mathématiques et statistiques qui sont communément utilisés dans ce domaine de recherche, un tour d'horizon subjectif de mes résultats est proposé. Celui-ci introduit des solutions aux problématiques de résumé de distribution, d'approximation de métriques, de détection d'éléments surabondants ou d'échantillonnage. Ces propositions sont toujours étayées d'une analyse théorique et de simulations poussées pour mettre en exergue les performances de nos algorithmes. Enfin, l'intégration de ces solutions dans des prototypes fonctionnels et les résultats d'expérimentations afférents démontrent la mise en œuvre pratique de ces contributions dans un environnement réel. Ce document est conclu par une mise en perspective de ces résultats et des propositions de projets de recherche envisagés par la suite.

Mots-clé :

Flux de données à large échelle ; Algorithmes d'approximation ; Modélisation de systèmes répartis ; Analyse statistique ; Agrégation de données ; Échantillonnage ; Big Data ; Systèmes de traitement de flux ; Sûreté de fonctionnement ; Métrologie des réseaux.

Table des matières

Nomenclature

Abréviations et acronymes

AMS Algorithme proposé par Alon *et al.* [1996]

AnKLe Algorithme *Attack-tolerant eNhanced Kullback-Leibler divergence Estimator*

ANR Agence Nationale pour la Recherche

ASSG Algorithme *Apache Storm Shuffle Grouping*

CASE Algorithme *Count-Any Sketch Estimator*

CCM Algorithme proposé par Chakrabarti *et al.* [2007]

CM Sketch Algorithme *Count-Min Sketch* proposé par Cormode et Muthukrishnan [2005]

CREST Centre de Recherche en Économie et Statistiques

DDoS Déni de Service Distribué

DHT Table de Hachage Distribuees

DKG Algorithme *Distribution-aware Key Grouping*

DMS Donnees Massives en Sante

DoS Déni de Service

ECM Sketch Algorithme proposé par Papapetrou *et al.* [2012]

Ensai École Nationale de la Statistique et de l'Analyse de l'Information

HTTP HyperText Transfer Protocol

IP Protocole Internet

IRISA Institut de Recherche en Informatique et Systèmes Aléatoires

KL-divergence Divergence de Kullback-Leibler

LAS Algorithme *Load-Aware Shedding*

LINA Laboratoire d'Informatique de Nantes Atlantique

NASA National Aeronautics and Space Administration

OSG Algorithme *Online Shuffle Grouping*

PSF Probleme de Surveillance Fonctionnelle [Cormode *et al.*, 2008]

SBC Single-Board Computer, tels que des Raspberry Pi

SHA-1 Secure Hash Algorithm

WSN Réseaux de capteurs sans fil

Symboles latins

$\mathcal{A}$ Algorithme d'approximation d'une fonction d'intérêt ϕ

a_t Élément reçu au temps t sur un flux donné

C_i Matrice de projections aléatoires

c_i Taille de la dimension i d'une matrice de projections aléatoires

d_r Distance des matrices de dispersion à la ronde r

$\mathcal{E}_k$ Surestimation de la codéviance agrégée en dimension k

$\mathbf{f}$ Vecteur de fréquences (*i.e.,* nombre d'occurrences) d'un flux donné

F_H Norme de l'entropie d'un flux donné

$\hat{F}$ Matrice de projections du CM Sketch

f_i Fréquence (*i.e.,* nombre d'occurrences) d'un élément i dans un flux donné

F_k $k^{\text{ième}}$ moment de fréquence d'un flux donné

H Entropie d'un flux donné

h Fonction de hachage

$\mathcal{H}$ Famille de fonctions de hachage

H_k $k^{\text{ième}}$ harmonique

m Nombre d'éléments composant un flux de taille finie

$\ddot{m}$ Taille de la fenêtre glissante sur un flux donné

n Cardinal de l'univers Ω

$\mathbf{p}$ Distribution des probabilités d'occurrences d'un flux donné

p_i Probabilité d'occurrences d'un élément i dans un flux donné

$\mathcal{P}_k(\Omega)$ Ensemble des partitions de Ω en k parties non vides et disjointes deux à deux

r Nombre de rondes d'un algorithme de PSF

$\mathcal{S}$ Ensemble des nœuds d'un système réparti, recevant un multi-flux

s Nombre de nœuds d'un système réparti, recevant un multi-flux

s_i Taille de la dimension i du CM Sketch

$S(n, k)$ Nombre de Stirling du deuxieme ordre

t Temps logique, *i.e.,* indice dans la séquence d'un flux donné

$T_{c,n}$ Collecteur de coupons : nombre de tirages pour obtenir c coupons distincts parmi n

Symboles grecs

α Paramètre de la loi de Zipf

β Seuil de fréquence minimale d'un élément surabondant

δ Probabilité de défaillance

δ_m Nombre d'éléments malveillants dans un flux donné

ε Erreur relative tolérée

Γ Mémoire tampon d'une structure de données

Λ File d'attente d'une structure de données

κ Proportion de nœuds byzantins dans un système

λ Paramètre de taille des mémoires tampons

μ Paramètre de temporisation du Diviseur

Ω Univers des éléments composant un flux donné

ϕ Fonction d'intérêt à calculer sur un flux donné

$\widehat{\phi}_k$ Métrique Sketch-ϕ de dimension k

$\sigma^{(i)}$ Flux de données reçu par le nœud $i \in \mathcal{S}$ (ou σ s'il n'y a pas d'ambiguïté selon le contexte)

$\sigma(t)$ Élément reçu au temps t sur un flux σ

τ Minuteur permettant de mesurer des délais de temporisation

Θ Seuil de fréquence minimale d'un iceberg

Ξ Période de temps logique

ξ Paramètre d'agrégation du Diviseur

Symboles mathématiques

$\mathbf{1}$	Fonction indicatrice
$\bar{\cdot}$	Moyenne des scalaires du vecteur donné
cod	Opérateur représentant la codéviance entre deux vecteurs de fréquences
$[\![M]\!]$	Ensemble des entiers naturels de 1 à M, *i.e.,* $[\![M]\!] = \{1, \dots, M\} = \mathbb{N} \cap [1; M]$
$polylog(n)$	Fonction polylogarithmique de n, *i.e.,* polynomiale par rapport au logarithme de n
supp	Support d'un vecteur (*i.e.,* l'ensemble des indices ayant un scalaire non nul)

Avant-propos

« Vita Cartesii est simplicissima. »

Incipit de *Monsieur Teste*, par Paul Valéry (1896).

Ces huit dernières années, depuis ma soutenance de thèse de doctorat, furent incroyablement riches et passionnantes. Ce document ne prétend pas représenter l'intégralité de mes recherches durant cette période, mais se focalise sur l'extraction d'informations pertinentes dans des flux de données massifs et potentiellement infinis. Mon attirance vers le « large échelle » s'est forgée lors mon doctorat, à l'IRISA (Rennes), et mon appétence vers la modélisation de grands systèmes communicants fut renforcée lors de mon séjour post-doctoral à la Sapienza Università di Roma (Italie), en 2008/2009. Les travaux présentés dans ce manuscrit couvrent principalement la période 2009–2016, à la fois au LINA, au sein de l'Université de Nantes, et au CREST, depuis mon arrivée à l'Ensai (Rennes). Cependant, il semble évident que l'évolution de mes thématiques de recherche n'est pas exempte d'une influence forte des différents collaborateurs avec lesquels j'ai eu le plaisir de travailler. Qu'ils en soient sincèrement remerciés ici.

Depuis mon départ de l'IRISA, ma recherche s'est organisée selon deux axes majeurs (*cf.* Figure 1), présentés succinctement ci-après :

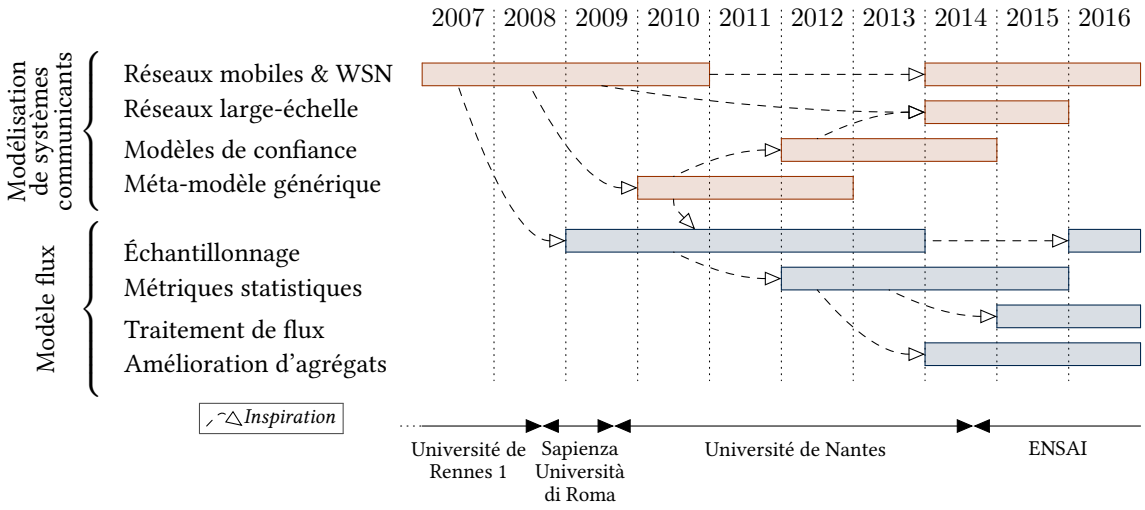


Figure 1 – Organisation thématique de mes recherches sur la période 2007–2016.

1 — Modélisation de systèmes communicants Sur ce premier pan de recherche, je me suis concentré sur l'abstraction de systèmes à large échelle, allant des réseaux de capteurs mobiles (*e.g.*, par l'étude du modèle des protocoles de population) aux systèmes pair-à-pair. Haussant d'un degré d'abstraction supplémentaire, je me suis ensuite intéressé à la définition d'un méta-modèle générique à tout système communicant et à son utilisation dans des modèles de confiance des utilisateurs envers leurs systèmes.

2 — Modèle flux En parallèle, j'ai développé un second axe de recherche, inspiré par les problématiques de diffusion dans les grands réseaux décentralisés. En effet, l'exploitation en temps réel des données massives du monde numérique nécessite l'appropriation de nouvelles méthodes. Dans ce contexte, je me suis intéressé à la caractérisation de problèmes, à

la proposition de méthodes d'approximation et à l'amélioration de structures de données existantes.

Nonobstant que je ne développe dans ce manuscrit qu'une sous-partie de mes travaux, ces deux axes ont toujours été étroitement liés lors de mes recherches. Ils peuvent être considérés comme deux facettes d'un unique problème plus général : le passage à l'échelle dans les grands systèmes communicants, à la fois en terme de contenants (*cf.* Axe 1) et de contenus (*cf.* Axe 2). Les réseaux et systèmes distribués n'ayant de cesse de grandir – simultanément en taille, complexité, contenus, hétérogénéité, dépendances (intra et inter), *etc.* – il est devenu nécessaire de définir des abstractions et des modélisations de plus haut niveau pour comprendre, améliorer et concevoir l'intrication inhérente des différentes facettes de ces systèmes. Ces observations représentent à la fois mon inspiration et les fondations de mes contributions en recherche.

Je vais résumer brièvement ci-après l'évolution chronologique de mes travaux avant introduire les contributions incluses dans ce document. En fin de doctorat à l'IRISA, je me suis principalement intéressé à la modélisation des réseaux de capteurs, d'une part ceux mobiles par le prisme des protocoles de population [Bertier *et al.*, 2009, 2010] et d'autre part, ceux qui, fixes, analysent les trajectoires d'objets mobiles [Busnel *et al.*, 2008b, 2011b]. A mon arrivée à Rome, j'ai poursuivi mes travaux sur les réseaux de capteurs à grande échelle [Bonomi *et al.*, 2009] mais je me suis surtout ouvert à la modélisation de grands systèmes communicants via la problématique d'échantillonnage uniforme, toujours avec une approche décentralisée [Busnel *et al.*, 2009, 2011a]. Dès lors, cet intérêt pour l'échantillonnage de grandes masses de données s'est poursuivi [Anceaume *et al.*, 2010, 2011, 2013a,b].

Mon arrivée à l'Université de Nantes m'a permis de franchir un cap supplémentaire sur la théorie de la modélisation, avec une approche de plus en plus orientée vers l'utilisateur et le contenu de ses données [Busnel *et al.*, 2010]. Nous avons alors proposé SocioPath [Alhadad *et al.*, 2012], un méta-modèle générique à tout système communiquant, permettant d'inférer les dépendances entre participants. Ces travaux ont soulevé de nombreuses questions, notamment en ce qui concerne la confiance des utilisateurs envers les systèmes qu'ils utilisent quotidiennement [Alhadad *et al.*, 2014, 2013, 2015]. L'ensemble de ces travaux s'est poursuivi depuis, principalement dans le cadre des réseaux à grande échelle, en ciblant différents aspects : routage et réputation [Anceaume *et al.*, 2015a], qualité de service et d'expérience [Samba *et al.*, 2016], caractérisation d'anomalies [Anceaume *et al.*, 2014], calcul réparti en environnement mobile [Busnel, 2014, Busnel *et al.*, 2013, Mocquard *et al.*, 2015], *etc.*

En parallèle, ma rencontre avec Emmanuelle Anceaume et son équipe a donné lieu à une très forte collaboration, encore vivement active à ce jour, sur la thématique porteuse du modèle de flux. L'intérêt d'estimer des métriques ou d'identifier des motifs spécifiques entre différents flux de données est important dans de nombreuses applications traitant des masses de données, incluant l'apprentissage, la fouille de données, les bases de données, la collecte d'informations, ou la surveillance réseau. Nous avons notamment proposé des algorithmes d'approximation de

métriques statistiques [Anceaume et Busnel, 2012, 2013, 2014a,b, Anceaume *et al.*, 2012], de détection d'éléments surabondants [Anceaume *et al.*, 2015c, 2016, 2015d], de structure de données permettant de résumer efficacement un flux [Anceaume *et al.*, 2015b, Rivetti *et al.*, 2015a], ou d'équilibrage de charge pour le traitement temps réel dans les centres de données [Rivetti *et al.*, 2016b, 2015b].

Cette collaboration s'est accrue depuis mon arrivée à l'Ensai à la faveur d'un rapprochement géographique. Outre les contributions sus-citées, j'ai eu à cœur d'appliquer ces dernières à des cas d'étude concrets, comme la métrologie et la sûreté de fonctionnement dans les réseaux de terminaux mono-carte (ou SBC pour *single-board computer*, tels que des Raspberry Pi) [DeSceNt, 2014, SocioPlug, 2013], ou le traitement efficace des données massives en santé, avec le Centre Hospitalier Universitaire de Rennes notamment [BigClin, 2016, INSHARE, 2016]. Ces derniers projets viennent d'être acceptés pour financement. Leurs impacts seront traités en partie *Perspectives* de ce manuscrit, spécifiquement dans la section 5.2.

Le reste de ce manuscrit porte essentiellement sur mon second axe de recherche. Quoique mes recherches sur la modélisation et le calcul réparti ont joué un rôle important, en façonnant les travaux présentés ci-après, je me suis permis de les laisser hors de la portée de ce document dans un souci de clarté et de concision.

Dernier point, mais non des moindres, l'ensemble des travaux présentés céans sont le fruit d'étroites collaborations avec des collègues et étudiants, lesquels sont mentionnés par les publications qui seront décrites dans les chapitres suivants. Cependant, toutes erreurs ou manques de clarté restent de ma responsabilité.

CHAPITRE 1

Introduction

« Ça a débuté comme ça. »

Incipit de *Voyage au bout de la nuit*, par Louis-Ferdinand Céline (1932).

La démocratisation et la miniaturisation des réseaux, en partie dues à la croissance exponentielle de l'Internet, ont permis d'envisager des systèmes à très grande échelle, visant à exploiter en commun une multitude de ressources à travers un nombre d'entités toujours croissant. Les systèmes distribués ont donc nécessité une migration de l'approche classique client-serveur vers une informatique répartie à très grande échelle. L'émergence récente du concept de *Cloud Computing* se place dans ce cadre et permet de démocratiser l'accès à ces infrastructures (initialement réservées à des applications de simulation "lourde"), de les rendre accessibles à un spectre de plus en plus large d'applications et donc à un nombre croissant d'acteurs du monde économique. Cette démocratisation est à l'origine du nombre grandissant de données transitant sur nos réseaux, nécessitant des approches novatrices pour le traitement des grandes masses de données.

Par exemple, l'apparition du web a bouleversé les habitudes et la vie de l'humanité et pas seulement des professionnels de l'informatique et de l'information [Mayer-Schönberger et Cukier, 2013]. Le web est lui-même en constante évolution. D'outil de communication et de dissémination, il est devenu éditable et réactif, pour à présent devenir social. Il suffit de voir l'ampleur prise par les réseaux sociaux pour s'en convaincre. Le web est ainsi devenu la première source d'information de tout un chacun.

1.1 Le défi du « Big Data »

Dans le cadre de l'*Internet du futur*, il est ainsi souhaitable de proposer des modèles de systèmes pour la gestion de grande masse de données distribuées à très large échelle par des communautés d'utilisateurs. Le défi est de construire des systèmes et réseaux performants et fiables mais aussi de permettre l'accès à l'information, la personnalisation et l'appropriation des contenus par les utilisateurs. L'émergence du *Big Data* ces dernières années se révèle comme incontournable et particulièrement pertinent pour le développement de ces systèmes [Lynch, 2008].

Au cours de la dernière décennie, nous avons donc assisté à l'émergence d'applications numériques nécessitant de faire face à de gigantesques quantités de données, générées de plus en plus rapidement. Ces applications diverses (métrologie des réseaux, biologie et médecine, applications financières, réseaux sociaux, *etc.*) nécessitent un besoin grandissant de techniques capables d'analyser et de traiter ces grandes masses d'information, avec précision et efficacité, en temps contraint. La statistique rejoint ici les sciences du numérique, et plus précisément l'informatique répartie, pour proposer de nouvelles approches, relatives au Big Data. Les techniques et les modèles doivent prendre en compte le volume pléthorique de ces données, mais également leur génération rapide en continu (vélocité) ainsi que l'hétérogénéité de leur format (variété). Il devient inconcevable de stocker l'ensemble des données concernées pour y effectuer ces traitements *a posteriori*. La diversité de ces contenus nécessite donc la proposition de solutions en temps réel innovantes et à faible complexité, à la fois en terme de mémoire, de temps de traitement, de

communication, *etc.* [Muthukrishnan, 2005].

Dans toutes ces applications, il est ainsi nécessaire de traiter rapidement et précisément un nombre considérable de données. Par exemple, dans la gestion d'un réseau IP, l'analyse des flux d'entrée permet de détecter rapidement la présence d'anomalies ou de tentatives d'intrusion quand des changements de motifs de communication apparaissent. Concernant la sécurisation de systèmes communicants à grande échelle, nous voudrions être capables de détecter des attaques par surveillance passive des messages transitant sur chaque nœud. S'il était possible de détecter une divergence entre un flux attendu et le flux observé sur un nœud, il serait possible d'alerter le propriétaire de celui-ci. Les faibles capacités de mémoire mises à disposition pour ces opérations de surveillance et métrologie rendent cependant la détection de ces attaques particulièrement complexe.

1.2 L'intérêt du modèle flux

Le problème d'extraire de l'information pertinente dans un (ou des) flux (répartis) de données est similaire au problème d'identifier des motifs qui ne sont pas conformes au comportement attendu. Cette thématique a été un champ de recherche très productif ces dernières décennies. Par exemple, en fonction des spécificités du domaine et du type d'aberrations considérées, différentes méthodes ont été conçues : classification, groupement, plus proche voisin, statistique spectrale et théorie de l'information. Un état de l'art intéressant sur ces techniques, décrivant leurs avantages et inconvénients, est proposé par Chandola *et al.* [2009]. Une fonctionnalité commune de ces techniques est leur grande complexité en espace et en coût de calcul, car elles reposent sur des algorithmes « à mémoire non bornée », *i.e.,* nécessitant la connaissance et l'accès complets aux données analysées.

Reposer sur des algorithmes de ce type n'est pas faisable dans un contexte d'extraction d'information en temps réel, avec une très faible capacité en terme de stockage et de traitement. Cependant, deux principales approches existent pour le traitement en temps réel de flux de données massifs, potentiellement répartis.

La première consiste à régulièrement échantillonner le flux, permettant de limiter la quantité de données conservées en mémoire [Alon *et al.*, 1996, Karamcheti *et al.*, 2005, Krishnamurthy *et al.*, 2003, Lakhina *et al.*, 2005]. Cette méthode permet de calculer exactement des fonctions sur ces échantillons. Cependant, la fiabilité de ce calcul, en comparaison avec le résultat exact calculé sur le flux entier, dépend fortement du volume de données qui a été échantillonné, et de la position de ces données sur le flux. Pire encore, un adversaire pourrait facilement tirer avantage de la politique d'échantillonnage pour masquer ses attaques parmi les paquets qui n'ont pas été échantillonnés, ou de ce fait, éviter que ses paquets « malveillants » puissent être corrélés.

A l'inverse, la seconde approche consiste à traiter à la volée toutes les données du flux, et de ne conserver localement que des résumés, ou *agrégats*, dans lesquels seule l'information essentielle

à propos des données est conservée [Chandola *et al.*, 2009]. Cette approche extrait des statistiques sur les flux de données traités, avec des probabilités d'erreurs bornées, sans présupposer aucune contrainte sur l'ordre dans lequel les données sont reçues sur les nœuds (*i.e.*, l'ordre des données pourrait être manipulé par un adversaire omnipotent [Anceaume *et al.*, 2010]). La plupart des recherches existantes utilisant cette approche se sont concentrées sur le calcul de fonctions ou de mesures statistiques avec une erreur ε donnée, utilisant une quantité mémoire poly-logarithmique en la taille du flux et du domaine des données entrantes. Ces fonctions estiment par exemple le nombre de données distinctes présentes dans un flux donné [Bar-Yossef *et al.*, 2002, Flajolet et Martin, 1985, Kane *et al.*, 2010], les moments de fréquence [Alon *et al.*, 1996], les éléments les plus fréquents [Alon *et al.*, 1996, Charikar *et al.*, 2004, Metwally *et al.*, 2005, Misra et Gries, 1982], l'entropie d'un flux [Chakrabarti *et al.*, 2007, Lall *et al.*, 2006], la détection d'attaque par inondation [Salem *et al.*, 2010], la covariance entre plusieurs flux [Anceaume et Busnel, 2014a] ou l'entropie relative entre un flux biaisé et un flux uniforme [Anceaume et Busnel, 2014b, Anceaume *et al.*, 2012].

1.3 Un modèle, des modèles

Malheureusement, calculer des mesures de théorie de l'information dans le modèle de flux est extrêmement complexe, essentiellement en raison du traitement séquentiel, à la volée, d'une incommensurable quantité de données en utilisant un espace mémoire très restreint (par rapport à la taille du flux) [Muthukrishnan, 2005].

De surcroît, l'analyse doit être temporellement robuste pour détecter les changements soudains sur les flux observés (potentiellement liés à la manifestation d'une attaque par déni de service ou la propagation d'un ver par exemple). L'un des modèles dérivés le plus naturel pour prendre en compte ces aspects est le modèle dit à *fenêtre glissante*, introduit par Datar *et al.* [2002], lequel considère uniquement une portion récente du flux considéré. Dans de nombreux contextes applicatifs, seuls les éléments reçus le plus récemment sont pertinents au regards de ceux reçus au démarrage du flux. La majorité des problèmes sus-cités trouvent une réponse dans ce modèle comme l'estimation de la variance [Zhang et Guan, 2007], des quantiles [Arasu et Manku, 2004], des éléments surabondants [Cormode et Yi, 2012, Golab *et al.*, 2003], la surveillance fonctionnelle [Chan *et al.*, 2012], ou la définition d'agrégats [Rivetti *et al.*, 2015a].

D'un autre côté, peu de travaux ont pris en compte le modèle de flux répartis, également appelé *problème de surveillance fonctionnelle* (PSF – « functional monitoring ») introduit par Cormode *et al.* [2008]. Celui-ci combine les caractéristiques des modèles de flux et de communication. Comme dans le modèle de flux, les données en entrée sont lues à la volée et traitées en un minimum d'espace et de temps. Cependant, la majorité des travaux proposant une analyse pour un flux unique ne sont pas adaptables au modèle réparti du PSF. Ici, chaque nœud distant reçoit un flux en entrée, effectue quelques traitements en local, et communique avec un coordinateur qui

souhaite calculer, ou estimer, de manière continue, une fonction donnée, sur l'union de tous les flux entrants. Le défi de ce modèle est, pour le coordinateur, de calculer cette fonction en minimisant le nombre de bits transmis [Anceaume et Busnel, 2014a,b, Anceaume *et al.*, 2015c, Gibbons et Tirthapura, 2001]. Cormode *et al.* [2008] innovent en proposant une étude formelle de fonctions réalisées dans ce modèle, se concentrant dans un premier temps sur les trois premiers moments de fréquence, introduits par Alon *et al.* [1996]. Arackaparambil *et al.* [2009] considèrent l'estimation répartie et empirique de l'entropie [Alon *et al.*, 1996] et améliorent les travaux de Cormode *et al.* en fournissant une borne inférieure sur la complexité d'estimation des moments de fréquence. Enfin, ils proposent des algorithmes répartis pour compter, à tout temps, le nombre de données qui a été reçu par un noeud depuis le début de leur flux respectif [Haung *et al.*, 2012, Liu *et al.*, 2012].

Enfin, d'autres modèles ont été proposés ces dernières années, lesquels relâchent certaines hypothèses comme le modèle multi-flux de Gibbons et Tirthapura [2001], ou mixent plusieurs modèles comme le PSF en fenêtre glissante de Cormode et Yi [2012] ou Gibbons et Tirthapura [2004].

1.4 Feuille de route

Dans la suite de ce manuscrit, je propose ainsi de revisiter plus en avant ces résultats et de combiner des méthodes statistiques de théorie de l'information avec des techniques d'échantillonnage, ou de résumé, pour extraire de l'information pertinente à partir d'une collection potentiellement grande de flux de données (métriques, agrégats, reconnaissance d'éléments surabondants, *etc.*).

Dans un premier temps, j'offre un tour d'horizon parfaitement subjectif des différents formalismes des modèles existants, de même qu'un aperçu des méthodes et outils usuels (*cf.* Chapitre 2),

L'objectif suivant est la conception et l'implémentation de prototypes d'algorithmes, répartis ou non, efficaces en une seule passe, dans un contexte où l'échange de masse des données est la norme (*cf.* Chapitre 3).

Ces algorithmes y ont déjà trouvé une utilité pertinente et croissante dans de nombreux contextes, comme la métrologie des réseaux, la sûreté de fonctionnement, les centres de calculs temps réel, *etc.* (*cf.* Chapitre 4).

Le champ des possibles usages de ces méthodes est considérable. Peu nombreux sont ceux qui peuvent prédire avec certitude ce que seront les besoins de l'Internet dans une décennie. Cependant, les possibilités de décentralisation de ces algorithmes, l'optimisation par l'apport d'une information sémantique ou l'ouverture à d'autres modèles de flux dessineront, en guise de conclusion, les perspectives concrètes de ce document (*cf.* Chapitre 5).

CHAPITRE 2

Modèles de flux de données et contraintes inhérentes

« Toute abstraction est si dure à accepter que notre premier réflexe est de la refuser, d'autant plus si elle s'inscrit à contre-courant de ce que nous avons toujours pensé. »

Dracula, par Bram Stoker (1897).

Ce chapitre a pour vocation de présenter, dans un premier temps, les différents modèles existant dans la littérature, lesquels formalisent tout à tour les flux de données, les systèmes répartis et les adversaires (*cf.* Section 2.1). Dans un second temps, il y est introduit les méthodes, notations et métriques usuelles, dans un objectif d'indépendance des sources (*cf.* Section 2.2).

2.1 Modèles de flux de données

Dans la communauté de recherche des flux de données, de nombreux modèles ont émergé afin de concorder avec les scénarii d'applications de ces approches temps réel. Cette section discute et formalise les modèles qui ont attiré davantage l'attention et qui sont utilisés dans ce manuscrit.

2.1.1 Modèle de flux unique

Le modèle fondateur considère les flux massifs comme une séquence d'éléments (ou *n-uplets*, ou encore *tuples*) potentiellement infinie $\langle a_1, a_2, \ldots, a_m \rangle$ que nous dénoterons σ par la suite. Les éléments sont tirés d'un univers Ω hypothétiquement très grand, que nous abstrairons, sans perdre en généralité $\{1, 2, \ldots, n\}$, par souci de clarté. Nous avons ainsi défini implicitement par $m = |\sigma|$ la taille (ou longueur) du flux considéré (dans le cas fini), et par $n = |\Omega|$, le cardinal de l'univers considéré. Il est important de noter que ce dernier est habituellement trop grand pour en espérer un stockage en mémoire efficace (*e.g.,* $n = 2^{128}$ pour les adresses IPv6 ou 2^{160} pour la fonction de hachage standard SHA-1).

Les éléments arrivent ainsi rapidement, sans contrainte d'ordre ni d'unicité. Les accès aux éléments suivent donc la séquence initiale (sans possibilité d'accès direct ou aléatoire). Les fonctions d'intérêt, que nous noterons ϕ par la suite, doivent ainsi être calculées en une seule passe (*i.e.,* en ligne et en temps réel).

L'enjeu principal est d'accomplir le calcul de ϕ en complexité mémoire sous-linéaire en m et n. Formellement, cette dernière doit respecter au minimum $o(\min\{m, n\})$, mais l'objectif principal reste d'assurer $O(\log(m \cdot n))$. Cependant, cet objectif est difficile (voire impossible) à atteindre dans la majorité des problèmes et la garantie conventionnelle est $polylog(m, n)$ [1]. Il convient d'établir ici que la notation « $\log$ » utilisée dans ce manuscrit correspond au logarithme en base 2. De plus, la complexité de traitement de chaque élément doit être minimale pour suivre le rythme imposé par le débit du flux.

Dans ce modèle, les solutions calculant de façon continue une fonction donnée sont souvent différenciées de celles effectuant un calcul ponctuel. Muthukrishnan [2005] propose un état de l'art pertinent et relativement exhaustif des recherches dans ce modèle, à l'horizon de 2005, mais

1. Par abus de langage, sachant que $f(x) = polylog(g(x)) \Leftrightarrow \exists c > 0, f(x) = O(\log^c g(n))$, la notation $polylog(m, n)$ est définie par $\exists c, c' > 0$ tels que $f(x) = O(\log^c(m) + \log^{c'}(n))$.

toujours d'actualité.

2.1.2 Modèle à fenêtre glissante

Dans de nombreuses applications, il est plus pertinent de ne prendre en considération que les éléments récemment observés, plutôt que l'ensemble du flux [Chan *et al.*, 2012]. Par exemple, dans une application de métrologie ou de détection d'attaque par déni de service, calculer le nombre d'occurrences d'une adresse destination dans un flux IP sur plusieurs mois n'a pas de sens. Ce constat a motivé la proposition d'un modèle à fenêtre glissante, formalisé par Datar *et al.* [2002].

Dans ce modèle, les éléments arrivent continuellement mais expirent après $\ddot{m}$ étapes. Les données pertinentes représentent donc les éléments reçus durant les $\ddot{m}$ dernières étapes. Une étape est définie soit :

- par l'arrivée d'un nouvel élément. La fenêtre glissante contient alors exactement $\ddot{m}$ éléments et la fenêtre est dite à taille fixe ;
- par un tic d'horloge (logique ou physique). La fenêtre est alors dite temporelle.

Formellement, en supposant qu'au plus un élément est reçu par étape, le flux $\sigma(t)$ considéré à l'étape t (ou au temps t) est la séquence $\langle a_1, \ldots, a_i, a_j, \ldots, a_{t'} \rangle$ où $i \leq t - \ddot{m}$, $j > t - \ddot{m}$ et $t' \leq t$ (lequel correspond au temps d'arrivée du dernier élément reçu). La fonction considérée devra ainsi être évaluée uniquement sur le sous-flux $\sigma_{\ddot{m}}(t) = \langle a_i \rangle_{i>t-\ddot{m}}$. La fenêtre glissante se réfère donc à la séquence des éléments valides.

Deux autres modèles simplifiés de la littérature méritent également d'être cités, dits à fenêtre à jalon et à fenêtre sautante. Dans le premier, une étape particulière est choisie comme point de repère et la fonction est calculée sur le sous-flux reçu à partir de celui-ci. A l'inverse, dans le modèle à fenêtre sautante, la fenêtre courante est divisée en plusieurs sous-fenêtres (habituellement de taille fixe), et progresse uniquement quand la sous-fenêtre courante est pleine, représentant un « saut en avant ».

L'ensemble de ces modèles peut être utilisé dans le cadre d'un calcul ponctuel ou continu. Ils héritent par ailleurs des contraintes de mémoire sous-linéaire et d'accès séquentiel du modèle précédent (*cf.* Section 2.1.1).

2.1.3 Modèle à flux multiples

Le modèle à flux multiples a été formalisé par Gibbons et Tirthapura [2001]. Nous considérons ici un ensemble de nœuds recevant chacun un flux de données distinct et coopérant par l'intermédiaire d'un coordinateur central.

Formellement, nous considérons un ensemble de nœuds $\mathcal{S} = \{1, \ldots, s\}$, recevant chacun un flux massif $\sigma^{(i)}$ ($i \in \mathcal{S}$) composé d'éléments tirés dans un univers commun Ω. Chaque sous-flux étant potentiellement non-bornés, les nœuds ne peuvent stocker localement qu'une quantité minime

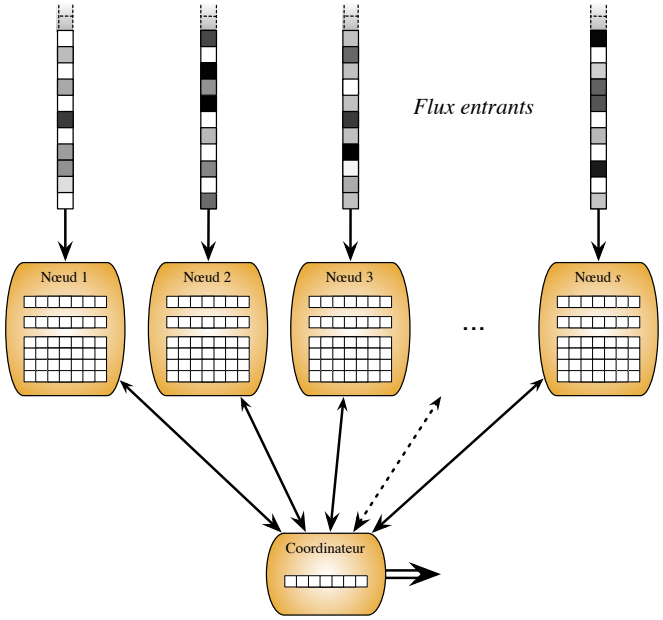


FIGURE 2.1 – Illustration des modèles à flux répartis

d'informations, au regard de m et n.

Dans ce modèle, les nœuds ne peuvent communiquer directement entre eux, mais doivent passer par l'intermédiaire d'un coordinateur. Ce dernier peut être vu comme une station de base dans un réseau de capteurs, ou un leader choisi dans un contexte décentralisé. En fonction de l'application, le mode de communication peut ainsi être modélisé comme uni- ou bi-directionnel. La figure 2.1 illustre le fonctionnement de ce modèle.

A nouveau, nous partageons les contraintes du modèle pionnier de la Section 2.1.1 avec comme objectif de minimiser la complexité de communication (en nombre de bits échangés), pour calculer ponctuellement une fonction d'intérêt ϕ sur l'union de ces flux :

$$\phi(\sigma) = \phi(\sigma^{(1)} \cup \ldots \cup \sigma^{(s)}).$$

2.1.4 Modèle de flux répartis continus

Cormode *et al.* [2008] ont récemment proposé le modèle de flux répartis continus, également connu sous le nom de problème de surveillance fonctionnelle, dont ils dressent un état de l'art des possibilités dans [Cormode, 2011, Cormode *et al.*, 2011].

Ce modèle est similaire au précédent à flux multiples, mais plutôt qu'une évaluation ponctuelle et globale, celui-ci considère une évaluation en continu de la fonction d'intérêt ϕ sur l'union des

flux. Formellement, l'objectif est de calculer à tout instant t,

$$\phi(\sigma(t)) = \phi(\sigma^{(1)}(t) \cup \ldots \cup \sigma^{(s)}(t)).$$

Cormode *et al.* [2011] bornent cependant ϕ à être monotone et distinguent deux classes de problèmes :

- la supervision à seuil, dont l'objectif est de lever une alerte si $\phi(\sigma(t))$ dépasse une certaine valeur limite donnée ;
- la supervision de valeur, qui calcule une approximation de la valeur exacte de $\phi(\sigma(t))$.

Les auteurs affirment néanmoins que la seconde classe de problèmes peut être réduite à la première, avec un facteur $O(\frac{1}{\varepsilon} \log R)$ bits de communication, où R représente la plus grande valeur atteinte de ϕ. Toutefois, la question reste ouverte pour le calcul de fonctions non monotones, telles que l'entropie par exemple.

2.1.5 Combinaison de modèles

Comme introduit dans la Section 1.3, un intérêt significatif existe pour un modèle permettant de combiner les aspects multi-flux et fenêtre glissante. Gibbons et Tirthapura [2004] proposent une formalisation de celui-ci, tout en soulevant le problème de la définition d'une fenêtre commune dans un environnement synchrone. Les auteurs proposent ainsi trois définitions d'une fenêtre de $\ddot{m}$ éléments sur s flux répartis :

1. Une fenêtre est composée des $\ddot{m}$ derniers éléments de chacun des s flux. La fenêtre représente donc une taille globale de $s\ddot{m}$ éléments.

2. Les flux locaux sont extraits d'un flux global unique arrivant à l'entrée du système et répartis arbitrairement sur les différents nœuds participant. Chaque élément reçu localement est donc enrichi d'un numéro de séquence global. Ainsi, une fenêtre représente $\ddot{m}$ éléments consécutifs dans le flux entrant unique.

3. Les $\ddot{m}$ derniers éléments de chacun des s flux locaux sont fusionnés pour produire un unique flux sortant global, également de taille $\ddot{m}$. Les éléments à l'étape t de ce flux combiné résultent du *ou* logique appliqué sur les s éléments des flux locaux à l'étape t (*i.e.*, $a_t = a_t^{(1)} \vee \ldots \vee a_t^{(s)}$).

Si les deux premières définitions diffèrent formellement, elles peuvent être résolues par adaptation d'algorithmes existants, à la fois dans les flux uniques ou multiples. Cependant, la dernière ouvre un champ de recherche stimulant mais ambitieux qui reste à explorer, au même titre que leur adaptation au modèle asynchrone.

2.1.6 Modèles d'adversaires

De manière transversale, dans n'importe quel modèle cité ci-dessus, nous pouvons supposer la présence de nœuds malveillants (ou Byzantins) qui cherchent collectivement à subvertir le système, en manipulant les algorithmes considérés.

Ce comportement malveillant est modélisé par un adversaire qui contrôle et manipule intégralement ces nœuds. Nous supposons que l'adversaire est adaptatif, en ce sens qu'il peut manipuler, autant qu'il le souhaite, le flux entrant de n'importe quel nœud dans $\mathcal{S}$, en observant, insérant, ou réorganisant les éléments de ce flux.

Afin d'éviter toute forme de sécurité par obscurité, l'algorithme utilisé est considéré comme une connaissance publique. Cependant, l'adversaire ne peut avoir connaissance des valeurs aléatoires générées localement par l'algorithme (*e.g.,* le choix aléatoire des fonctions de hachage à partir d'une famille de fonctions 2-universelles – *cf.* Section 2.2.2.2).

Généralement, la résilience face à un adversaire est révélée par l'effort requis par ce dernier pour renverser l'algorithme considéré (*i.e.,* pour enfreindre ses garanties de précision). D'autre part, dans le cas des flux répartis, l'adversaire peut également mettre en œuvre une stratégie pour faire du coordinateur une cible d'attaque par déni de service distribué (DDoS), en maximisant les interactions entre lui et les s nœuds du système. Il peut y parvenir en réorganisant les éléments des flux et/ou en injectant stratégiquement des éléments à des positions pertinentes.

Un nœud du système qui n'est pas malveillant est dit *correct*. Cependant, aucun nœud correct ne peut distinguer *a priori* les autres nœuds corrects des malveillants. Traditionnellement, un adversaire n'est pas en mesure de supprimer ou de modifier le contenu d'un message échangé entre deux nœuds corrects sans être détecté [Bortnikov *et al.*, 2009]. L'existence d'un mécanisme d'authentification est ainsi supposée pour assurer l'authenticité et l'intégrité des messages, correspondant au modèle de défaillance Byzantine authentifiée [Lynch, 1996]. De même, le contrôle de multiples nœuds est très coûteux pour l'adversaire, qui doit ainsi interagir avec une autorité centrale pour recevoir les certificats adéquats, *i.e.,* le modèle d'attaque Sybil [Douceur, 2002].

2.2 Contraintes, méthodes et boite à outils

L'ensemble des contraintes de ces modèles a été précédemment introduit, tel que la nécessité de fonctionner à mémoire restreinte, en ligne, avec un accès séquentiel, une faible complexité de calcul et de communication, *etc.* Dans ces conditions, la plupart du temps, il est impossible d'obtenir les valeurs exactes de la fonction d'intérêt ϕ considérée. Par conséquent, il est d'usage de recourir à des algorithmes d'approximation, permettant de garantir un résultat estimé avec une marge d'erreur relative bornée.

Cette section présente les techniques courantes d'estimation, puis expose les briques de base utilisées dans les solutions présentées dans les chapitres suivants.

2.2.1 Méthodes d'approximation classiques

Comme introduit dans le chapitre précédent, deux techniques d'approximation, que nous présentons succinctement ci-après, sont principalement utilisées dans les solutions existantes.

2.2.1.1 Échantillonage

L'approche par échantillonnage consiste à prélever un sous-ensemble restreint d'éléments reçus sur le flux, et d'appliquer la fonction d'intérêt ϕ à ce sous-ensemble. Conventionnellement, l'enjeu persiste à restreindre l'espace de stockage à une taille poly-logarithmique en la longueur du flux. Couramment, des méta-données sont aussi associées aux éléments échantillonnés [Alon *et al.*, 1996, Karamcheti *et al.*, 2005, Krishnamurthy *et al.*, 2003, Lakhina *et al.*, 2005, Misra et Gries, 1982].

Il existe de nombreuses méthodes pour choisir quel élément doit être échantillonné, lesquelles peuvent être déterministes ou aléatoires [Muthukrishnan, 2005]. Cependant, la présence d'un adversaire contraint fortement la politique d'échantillonnage, celui-là pouvant tirer profit de cette dernière pour augmenter ses chances de réussir une attaque. Afin de limiter l'impact de ces attaques, une méthode classique pour augmenter l'entropie d'un plan de sondage est de réordonner aléatoirement le flux avant d'échantillonner [Chauvet, 2012]. Malheureusement, ce processus est incompatible avec les modèles présentés ci-dessus.

2.2.1.2 Projections aléatoires et agrégats

A l'inverse, les techniques de projection et d'agrégat examinent chaque élément du flux d'entrée à la volée et conservent localement un résumé de l'information pertinente pour la fonction ϕ à estimer.

Les méthodes usuelles reposent sur des *projections (pseudo-)aléatoires* permettant de réduire significativement la taille des données. L'empilement de projections aléatoires permet souvent d'affiner le résultat final. Les vecteurs de projection reposent principalement sur les fonctions de hachage utilisées pour y extraire les statistiques désirées et sont générés en utilisant des calculs à faible mémoire, sur des variables aléatoires indépendantes [Charikar *et al.*, 2004, Cormode et Muthukrishnan, 2005].

Un *agrégat* est défini comme une structure de données (telle que les projections aléatoires) qu'il est possible de fusionner sans perte d'information supplémentaire, en comparaison à l'élaboration de celle-ci sur les données globales. Formellement, une structure de données `SD` est un agrégat s'il existe un algorithme de fusion `COMB` tel que, pour tout couple de flux (σ_1, σ_2),

$$\texttt{COMB}(\texttt{DS}(\sigma_1), \texttt{DS}(\sigma_2)) = \texttt{DS}(\sigma_1 \cup \sigma_2),$$

où $\sigma_1 \cup \sigma_2$ représente l'union des éléments de σ_1 et σ_2 selon n'importe quelle intrication. De

plus, une structure de données est un *agrégat linéaire* si elle prend ses valeurs dans un espace vectoriel de dimension finie, souvent dépendant de n, et que l'algorithme d'agrégation est une fonction linéaire de ϕ.

Dans le cadre des modèles à fenêtre glissante, les définitions d'agrégat sont bien entendu limitées à l'intrication des éléments des flux σ_1 et σ_2 sur les $\ddot{m}$ derniers éléments seulement.

2.2.2 Boîte à outils

Dans un souci d'exhaustivité, cette seconde partie présente les formalismes et différentes métriques qui seront utilisés dans la suite de ce manuscrit.

2.2.2.1 Algorithmes d'approximation

Un algorithme $\mathcal{A}$ est une *ε-approximation* d'une fonction d'intérêt ϕ sur un flux σ si, pour toute séquence d'éléments de σ, l'algorithme $\mathcal{A}$ retourne une estimation $\hat{\phi}$ telle que

$$\mid \hat{\phi}(\sigma) - \phi(\sigma) \mid \leq \varepsilon\phi(\sigma),$$

où $\varepsilon > 0$ est un paramètre de précision donnée de $\mathcal{A}$.

De même, un algorithme aléatoire $\mathcal{A}$ est une *(ε, δ)-approximation (relative)* d'une fonction d'intérêt ϕ sur un flux σ si, pour toute séquence d'éléments de σ, l'algorithme $\mathcal{A}$ retourne une estimation $\hat{\phi}(\sigma)$ telle que

$$\mathbb{P}\{\mid \hat{\phi}(\sigma) - \phi(\sigma) \mid > \varepsilon\phi(\sigma)\} < \delta,$$

où $\varepsilon, \delta > 0$ représentent respectivement les paramètres de précision et de sensibilité de $\mathcal{A}$.

Enfin, dans certains cas, cette condition est trop forte, par exemple lorsque les valeurs de ϕ sont proches de ou égales à 0. Dans les hypothèses ci-dessus, un algorithme aléatoire $\mathcal{A}$ est une (ε, δ)-*approximation absolue* d'une fonction d'intérêt ϕ sur un flux σ si

$$\mathbb{P}\{\mid \hat{\phi}(\sigma) - \phi(\sigma) \mid > \varepsilon\} < \delta.$$

2.2.2.2 Fonctions de hachage

Une famille $\mathcal{H}$ de fonctions de hachage $h : [\![M]\!] \to [\![N]\!]$ est dite *ℓ-indépendante* si pour tout ℓ éléments distincts $x_1, \ldots, x_\ell \in [\![M]\!]$ et pour tout ℓ hachés (non nécessairement distincts) $y_1, \ldots, y_\ell \in [\![N]\!]$,

$$\mathbb{P}_{h\in\mathcal{H}}\{h(x_1) = y_1 \wedge \ldots \wedge h(x_\ell) = y_\ell\} \leq \frac{1}{N^\ell}$$

Néanmoins, bien qu'elles soient robustes et performantes, ces fonctions de hachage sont particulièrement difficiles à construire, *i.e.,* il est presque impossible de constituer un très grand ensemble de fonctions de hachage ℓ-indépendantes.

En pratique, la propriété d'universalité suffit à garantir la projection correcte des éléments. Une famille $\mathcal{H}$ de fonctions de hachage $h : [\![M]\!] \to [\![N]\!]$ est dite *universelle* si pour tout couple d'éléments distincts $x, y \in [\![M]\!]$,

$$\mathbb{P}_{h\in\mathcal{H}}\{h(x) = h(y)\} \leq \frac{1}{N}.$$

En d'autres termes, cela garantit que la probabilité de collision de deux éléments distincts de l'univers de départ est au plus $1/N$, lorsque la fonction de hachage est choisie aléatoirement parmi $\mathcal{H}$. Carter et Wegman [1979] ont introduit ce concept et proposé une méthode simple de construction de très grandes familles universelles. En effet, à partir du choix d'un entier premier $p > M$, la famille suivante est universelle :

$$\{h_{a,b}(x) = ((ax + b) \mod p) \mod N\}_{a\in[\![p]\!],b\in[\![p]\!]\cup\{0\}}.$$

Les fonctions ainsi construites sont dites 2-universelles, attendu qu'elles sont indépendantes deux à deux (par le tirage aléatoire indépendant de a et b).

2.2.2.3 Métriques statistiques

Vecteur de fréquences et distribution empirique Un flux $\sigma = \langle a_1, a_2, \ldots, a_m\rangle$ définit implicitement un vecteur de fréquences $\mathbf{f} = (f_1, \ldots, f_n)$, où f_u représente le nombre d'occurrences d'un élément u dans σ.

Ce vecteur de fréquences peut aussi être assimilé à la distribution de probabilité empirique des éléments de Ω sur le flux

$$\mathbf{p} = (p_u)_{u\in\Omega} \quad \text{où} \quad p_u = \frac{f_u}{\sum_{v\in\Omega} f_v} = \frac{f_u}{m}.$$

Moments de fréquence Les moments de fréquence d'un flux sont des mesures statistiques importantes, introduites par Alon *et al.* [1996]. La valeur des moments de fréquences F_k fournit une quantification du taux de biais de la distribution du flux. Pour tout $k \geq 0$, le $k^{\text{ième}}$ moment de fréquence F_k du flux σ est défini par

$$F_k(\sigma) = \sum_{u\in\Omega} f_u^k.$$

Parmi les moments marquants, il est intéressant de relever que

- $F_0(\sigma)$ représente le nombre d'éléments distincts de σ (où par convention $0^0 = 0$ [Jacquelin, 2007]) ;
- $F_1(\sigma)$ représente la longueur m de σ ;
- $F_2(\sigma)$ représente « l'asymétrie » de la distribution de σ (*i.e.,* sa divergence par rapport à

la loi uniforme) ;
- $F_\infty(\sigma)$ représente l'élément le plus fréquent dans σ [Flajolet et Martin, 1985].

Entropie Intuitivement, l'entropie est une mesure du caractère aléatoire d'un flux σ [Basseville et Cardoso, 1995]. L'entropie $H(\sigma)$ est minimale (*i.e.,* égale à 0) quand tous les éléments du flux sont identiques, et elle atteint son maximum quand tous les éléments du flux sont distincts. Formellement, l'entropie du flux σ est définie par

$$H(\sigma) = -\sum_{u \in \Omega} p_u \log p_u. \tag{2.1}$$

Par convention, il est établi ici que $0 \log 0 = 0$.

Norme de l'entropie Par analogie, la norme de l'entropie d'un flux σ est définie par

$$F_H(\sigma) = \sum_{u \in \Omega} f_u \log f_u. \tag{2.2}$$

Entropie relative L'entropie relative, aussi dénommée la divergence de Kullback-Leibler (KL-divergence) [Kullback et Leibler, 1951], est une métrique robuste qui estime la différence statistique entre deux flux de données. La KL-divergence fait partie de la classe des distances de Ali-Silvey [Ali et Silvey, 1966] et possède la particularité intéressante de mettre en exergue un faible nombre de faux positifs quand deux flux de données sont significativement différents.

Etant donnés deux flux σ_1 et σ_2 ayant pour distribution de probabilité respectivement $\mathbf{p}$ et $\mathbf{q}$ sur Ω, la KL-divergence de σ_1 vers σ_2 est définie comme valeur attendue du rapport de vraisemblance de $\mathbf{p}$ par rapport à $\mathbf{q}$:

$$\mathcal{D}(\sigma_1 || \sigma_2) = \sum_{u \in \Omega} p_u \log \frac{p_u}{q_u} = H(\sigma_1, \sigma_2) - H(\sigma_1),$$

où $H(\sigma_1, \sigma_2) = -\sum p_u \log q_u$ est l'entropie croisée de p vers q. La KL-divergence est donc nulle si et seulement si $\mathbf{p} = \mathbf{q}$.

CHAPITRE 3

Problèmes classiques et solutions algorithmiques

« La science n'a jamais tout à fait raison, mais elle a rarement tout à fait tort,
et, en général, elle a plus de chance d'avoir raison que les théories non scientifiques.
Il est donc rationnel de l'accepter à titre d'hypothèse. »

Histoire de mes idées philosophiques, par Bertrand Russell (1961).

Sans prétendre à une quelconque exhaustivité, ce chapitre a pour vocation de présenter un tour d'horizon des problématiques sur lesquelles j'ai contribué ces dernières années, avec une approche plus orientée « algorithmique ». Le chapitre suivant mettra en perspective ces contributions par leur intégration à des cas d'utilisations pratiques.

Ce chapitre présentera ainsi successivement les problématiques de résumé de distribution et de leur utilisation dans les calculs de métriques statistiques (*cf.* Section 3.1), de détection d'éléments surabondants dans les environnements multi-flux (*cf.* Section 3.2), d'estimation de fréquence d'occurrence de tout élément d'un flux (*cf.* Section 3.3) et enfin, d'échantillonnage uniforme de flux (*cf.* Section 3.4).

Dans un souci de clarté et de concision, les détails des preuves des résultats obtenus sont disponibles dans les articles référencés correspondants.

3.1 Résumés de distribution et calculs de métriques

Les travaux sur l'analyse des flux de données ont initialement porté sur des méthodes (algorithmes et structures de données) capables de répondre à différents types de requêtes. Principalement, ces méthodes consistaient à dériver des estimateurs statistiques sur des flux massifs, en créant des représentations résumées (comme des histogrammes, ondelettes ou quantiles) et en les comparant entre eux. De nombreux travaux ont proposé des estimateurs de moments de fréquences [Alon *et al.*, 1996], de l'entropie et notions dérivées [Cover et Thomas, 1991], ou autres quantificateurs du degré d'aléa [Anceaume et Busnel, 2012, Chakrabarti *et al.*, 2006, Guha *et al.*, 2006, Haung *et al.*, 2012, Lall *et al.*, 2006, Liu *et al.*, 2012]. L'utilisation de projection aléatoire est alors apparue comme intéressante dans ce contexte [Charikar *et al.*, 2004, Cormode et Garofalakis, 2007]. En effet, Guha *et al.* [2008] proposent une caractérisation des mesures de divergences qui sont « agrégeables ». Ils ont notamment prouvé que n'importe quelle métrique qui ne dispose pas des propriétés d'une norme ne peut être agrégeable.

3.1.1 Codéviance : estimer la corrélation entre flux

Dans ce cadre, nous avons proposé une nouvelle métrique permettant d'évaluer la corrélation entre plusieurs flux distribués [Anceaume et Busnel, 2014a], par l'étude de la matrice de dispersion.

Spécifiquement, nous introduisons le concept de *codéviance* sur les vecteurs de fréquences des flux, inspiré par la classique covariance des distributions en statistique. Une telle métrique nous permet de qualifier la dépendance ou la corrélation entre deux vecteurs en comparant leurs variations. Cette métrique capture notamment les changements de comportement d'un trafic réseau lors d'une attaque malveillante (*cf.* Section 4.2.1). Formellement, la codéviance entre deux vecteurs de fréquences $\mathbf{f} = (f_1, f_2, \ldots, f_n)$ et $\mathbf{f}' = (f'_1, f'_2, \ldots, f'_n)$ est définie par la valeur réelle

$\mathrm{cod}(\mathbf{f}, \mathbf{f}')$ suivante :

$$\mathrm{cod}(\mathbf{f}, \mathbf{f}') = \frac{1}{n} \sum_{i \in \Omega} (f_i - \overline{\mathbf{f}})(f'_i - \overline{\mathbf{f}'}) = \frac{1}{n} \sum_{i \in \Omega} f_i f'_i - \overline{\mathbf{f}}\,\overline{\mathbf{f}'} \tag{3.1}$$

où $\overline{\mathbf{f}} = \frac{1}{n} \sum_{i \in \Omega} f_i$ et $\overline{\mathbf{f}'} = \frac{1}{n} \sum_{i \in \Omega} f'_i$.

Cette métrique reposant sur la connaissance complète des vecteurs de fréquences des flux considérés, il est impossible de l'appliquer directement dans le cadre de nos modèles. Nous avons donc proposé une méthodologie, dénommée *codéviance agrégée*, permettant d'estimer la valeur de codéviance de deux flux. Celle-ci consiste à calculer la valeur de codéviance sur toutes les projections possibles dans un espace vectoriel de dimension sous-linéaire à Ω. Formellement, étant donné un paramètre de précision $k \in \mathbb{N}^*$, la codéviance agrégée sur deux vecteurs de fréquences $\mathbf{f} = (f_1, f_2, \ldots, f_n)$ et $\mathbf{f}' = (f'_1, f'_2, \ldots, f'_n)$, est définie par

$$\begin{aligned} \widehat{\mathrm{cod}}_k(\mathbf{f}, \mathbf{f}') &= \min_{\rho \in \mathcal{P}_k(\Omega)} \mathrm{cod}\left(\widehat{\mathbf{f}}_\rho, \widehat{\mathbf{f}'}_\rho\right) \\ &= \min_{\rho \in \mathcal{P}_k(\Omega)} \left(\frac{1}{n} \sum_{a \in \rho} \widehat{\mathbf{f}}_\rho(a) \widehat{\mathbf{f}}'_\rho(a) - \left(\frac{1}{n} \sum_{a \in \rho} \widehat{\mathbf{f}}_\rho(a) \right) \left(\frac{1}{n} \sum_{a \in \rho} \widehat{\mathbf{f}}'_\rho(a) \right) \right) \end{aligned} \tag{3.2}$$

où $\forall a \in \rho, \widehat{\mathbf{f}}_\rho(a) = \sum_{i \in a} f_i$, et $\mathcal{P}_k(\Omega)$ est l'ensemble des partitions de Ω en k parties non vides et disjointes deux à deux.

Nous démontrons ensuite quelques résultats intéressants permettant de relier la codéviance avec sa version agrégée. En effet, la seconde est fonction de la première. Par exemple, étant donnés deux vecteurs de fréquences $\mathbf{f}$ et $\mathbf{f}'$, nous avons :

$$\forall k \geq n, \widehat{\mathrm{cod}}_k(\mathbf{f}, \mathbf{f}') = \mathrm{cod}(\mathbf{f}, \mathbf{f}') \tag{3.3}$$

$$\forall k < n, \widehat{\mathrm{cod}}_k(\mathbf{f}, \mathbf{f}') = \mathrm{cod}(\mathbf{f}, \mathbf{f}') + \mathcal{E}_k(\mathbf{f}, \mathbf{f}') \tag{3.4}$$

$$\text{où } \mathcal{E}_k(\mathbf{f}, \mathbf{f}') = \min_{\rho \in \mathcal{P}_k(\Omega)} \frac{1}{n} \sum_{a \in \rho} \sum_{i \in a} \sum_{j \in a \setminus \{i\}} f_i f'_j$$

De plus, nous avons montré que la surestimation est également nulle (*i.e.,* $\mathcal{E}_k(\mathbf{f}, \mathbf{f}') = 0$) et donc $\widehat{\mathrm{cod}}_k(\mathbf{f}, \mathbf{f}') = \mathrm{cod}(\mathbf{f}, \mathbf{f}')$ si $k \geq |\ \mathrm{supp}(X) \cap \mathrm{supp}(Y)\ | + \mathbf{1}_{\mathrm{supp}(X) \setminus \mathrm{supp}(Y)} + \mathbf{1}_{\mathrm{supp}(Y) \setminus \mathrm{supp}(X)}$, où $\mathrm{supp}(\mathbf{f})$ représente le support du vecteur $\mathbf{f}$ (*i.e.,* l'ensemble des éléments de Ω de fréquence non nulle, soit $\mathrm{supp}(\mathbf{f}) = \{i \in \Omega \mid f_i \neq 0\}$), et la notation $\mathbf{1}_A$ dénote la fonction indicatrice, qui est égale à 1 si l'ensemble A n'est pas vide et 0 sinon.

Nous avons finalement caractérisé la borne supérieure de la surestimation, lorsque k est stricte-

Algorithme 3.1 : Algorithme d'approximation de la codéviance agrégée

Entrées : Deux flux de données σ_1 and σ_2 ; Deux paramètres de précision ε et de sensibilité δ;
Sortie : L'estimation de la codéviance agrégée sur σ_1 et σ_2 : $\widehat{\mathrm{cod}}_k(\sigma_1, \sigma_2)$

$c_1 \leftarrow \lceil \ln \frac{1}{\delta} \rceil$; $c_2 \leftarrow \lceil \frac{e}{\varepsilon} \rceil$;
$C_{\sigma_1}[1..c_1][1..c_2] \leftarrow 0_{c_1,c_2}$; ▷ Matrice nulle de dimension $c_1 \times c_2$ ◁
$C_{\sigma_2}[1..c_1][1..c_2] \leftarrow 0_{c_1,c_2}$;
Choisir c_1 fonctions de hachage $h_\ell : \Omega \to [\![c_2]\!]$ parmi une famille 2-universelle;
pour $u \in \sigma_1$ **faire**
    pour $\ell = 1$ **à** c_1 **faire**
        $C_{\sigma_1}[\ell][h_\ell(u)] \leftarrow C_{\sigma_1}[\ell][h_\ell(u)] + 1$;
pour $v \in \sigma_2$ **faire**
    pour $\ell = 1$ **à** c_1 **faire**
        $C_{\sigma_2}[\ell][h_\ell(v)] \leftarrow C_{\sigma_2}[\ell][h_\ell(v)] + 1$;
à la requête $\widehat{\mathrm{cod}}(\sigma_1, \sigma_2)$ **retourner** $\min_{1 \le \ell \le c_1} \mathrm{cod}(C_{\sigma_1}[\ell], C_{\sigma_2}[\ell])$

ment inférieur à cette borne :

$$\mathcal{E}_k = \max_{\mathbf{f},\mathbf{f}' \in \mathbb{N}^n} \mathcal{E}_k(\mathbf{f}, \mathbf{f}') = \begin{cases} \dfrac{F_1(\mathbf{f})F_1(\mathbf{f}')}{n} & \text{si } k = 1, \\ \dfrac{F_1(\mathbf{f})F_1(\mathbf{f}')}{n}\left(\dfrac{1}{k} - \dfrac{1}{n}\right) & \text{si } k > 1. \end{cases} \tag{3.5}$$

Nous avons ensuite proposé un algorithme simple et très peu coûteux en espace permettant d'estimer à la volée la codéviance agrégée sur deux flux massifs σ_1 et σ_2. Par définition (*cf.* Relation 3.2), il est nécessaire de générer toutes les partitions possibles de dimension k. Le nombre de ces partitions est équivalent au nombre de Stirling du deuxième ordre, *i.e.,*

$$S(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n,$$

lequel est exponentiellement croissant. Cela n'est évidement pas envisageable dans notre environnement contraint. En pratique, il suffit de générer seulement $c_1 = \lceil \log(1/\delta) \rceil$ partitions tirées aléatoirement (où δ est la probabilité de défaillance tolérée), pour garantir de bonnes performances globales de notre algorithme. Le pseudo-code de ce dernier est présenté dans l'Algorithme 3.1.

Il est important de noter qu'aucune hypothèse n'est faite sur la longueur respectives des deux flux (m_1 et m_2 sont finis mais inconnus et peuvent être différents). Inspirées par les travaux sur les agrégats de Cormode et Muthukrishnan [2005], deux séries de projections aléatoires sont

construites à la volée $C_{\sigma 1}$ et C_{σ_2}, reposant sur une collection de fonctions aléatoires, identiques pour les deux agrégats. À chaque élément u (resp. v) lu sur σ_1 (resp. σ_2), son compteur correspondant est incrémenté sur chacune des c_1 projections, *i.e.,* $C_{\sigma_1}[\ell][h_\ell(u)]$ est incrémenté pour toutes les valeurs de ℓ dans $[\![c_1]\!]$. Ainsi, chaque ligne $\ell \in [\![c_1]\!]$ de $C_{\sigma 1}$ et C_{σ_2} correspond à une même partition ρ_ℓ de Ω, et chaque scalaire a d'une ligne ℓ correspond à $\widehat{\mathbf{f}}_{\rho_\ell}(a)$ (*cf.* relation 3.2). Par conséquent, à la réception d'une requête de codéviance agrégée $\widehat{\mathrm{cod}}$, la codéviance entre chacune des c_1 lignes de $C_{\sigma 1}$ et C_{σ_2} est calculée et la valeur minimale est retournée.

Nous avons prouvé que cet algorithme est une (ε, δ)-approximation additive vérifiant

$$\mathbb{P}\left\{\left|\widehat{\mathrm{cod}}_{\lceil \frac{e}{\varepsilon} \rceil}(\mathbf{f}_1, \mathbf{f}_2) - \mathrm{cod}(\mathbf{f}_1, \mathbf{f}_2)\right| \geq \frac{\varepsilon}{n}\left(F_1(\mathbf{f}_1)F_1(\mathbf{f}_2) - F_1(\mathbf{f}_1 \cdot \mathbf{f}_2)\right)\right\} \leq \delta$$

De plus, il possède les propriétés d'agrégat linéaire et vérifie une faible complexité mémoire par l'utilisation de $O\left(\frac{1}{\varepsilon}\log\frac{1}{\delta}(\log n + \log m)\right)$ bits (avec $m = \max(m_1, m_2)$).

Nous avons également proposé une extension de cet algorithme pour le modèle de flux répartis continus, permettant de calculer la codéviance entre un ensemble de s flux répartis. L'objectif pour le coordinateur est de maintenir à jour la matrice de codéviance, représentant la matrice de dispersion des s flux :

$$\widehat{\mathrm{COD}} = \left[\widehat{\mathrm{cod}}(\mathbf{f}_i, \mathbf{f}_j)\right]_{i,j \in \mathcal{S}}.$$

Cet algorithme fonctionne en rondes, de longueurs géométriquement croissantes, au cours de desquelles, chaque nœud va transmettre au coordinateur l'état courant de sa matrice de projection aléatoire C_{σ_i} ($i \in \mathcal{S}$). À chaque fin de ronde, le coordinateur est donc en mesure de construire la matrice de dispersion du système, en appliquant la méthode présentée ci-dessus à chaque couple de matrice $(C_{\sigma_i}, C_{\sigma_j})$ ($i, j \in \mathcal{S}$).

Nous avons montré que les bornes théoriques de précision et de complexité en espaces restent valides, tout en communiquant au plus $O\left(s\left(1 + (1/\varepsilon)\log(1/\delta)\log^2(m)\right)\right)$ bits.

Enfin, nous avons implémenté notre algorithme et procédé à une série d'expérimentations sur différentes distributions de flux. La figure 3.1 représente la courbe de niveaux de la matrice de dispersion pour 13 flux, chacun de longueur $m = 100\,000$, contenant au plus $n = 1\,000$ éléments distincts. Chaque flux, identifié de 0 à 12 sur cette figure, suit respectivement des distributions suivantes :

Flux 0 **:** Loi uniforme ;

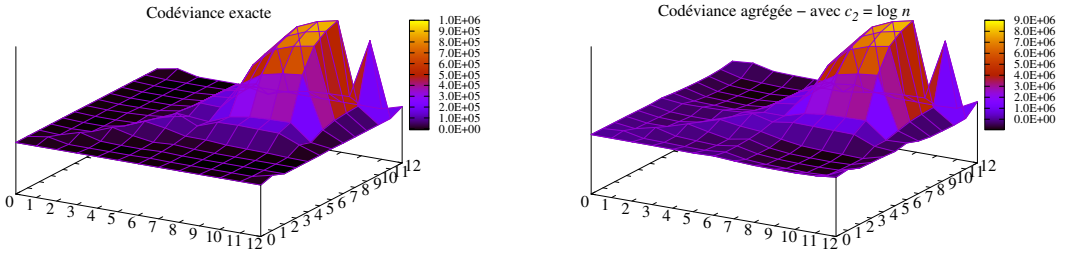
Flux $1 \ldots 5$ **:** Loi de Zipf avec respectivement un paramètre α allant de 1 à 5 ;

Flux $6 \ldots 11$ **:** Loi de Poisson avec respectivement un paramètre λ allant de 2^1 à 2^5 ;

Flux 12 **:** Loi binomiale tronquée ;

Flux 13 **:** Loi de Pascal (ou loi binomiale négative tronquée).

La partie gauche de la figure 3.1 représente la matrice de dispersion exacte, obtenue à partir des vecteurs de fréquences complets. La partie droite représente la matrice calculée par le coordi-



(a) Matrice de codéviance exacte

(b) Matrice de codéviance agrégée obtenue à partir de notre algorithme réparti avec $c_2 = \log n$

FIGURE 3.1 – Courbes de niveaux représentant les matrices de dispersion.

nateur à la fin de l'exécution de notre algorithme réparti d'approximation, avec un paramètre $c_2 = \log n$. L'axe des z représente la valeur de chaque cellule de la matrice correspondante, pour chacun des deux graphes. Ces résultats montrent que notre algorithme permet bien de quantifier fidèlement et efficacement comment les données observées sur les flux évoluent les unes par rapport aux autres, et en quelle proportion, quelle que soit la forme du flux d'entrée. La forme des isoplèthes sont similaires pour les deux courbes. Il faut noter que la différence d'ordre de grandeur des valeurs entre les deux courbes est due au facteur de surestimation, mais reste proportionnelle à l'exact.

3.1.2 Métrique Sketch-$\star$: estimer n'importe quelle distance entre flux

Les distances et divergences sont des mesures clés en inférence statistique et traitement des données [Basseville et Cardoso, 1995]. Il existe deux classes de mesure largement utilisées, à savoir les f-divergences (étonnamment introduites indépendamment par Ali et Silvey [1966], Csiszár [1963], Morimoto [1963]) et les divergences de Bregman [1967]. Elles permettent en effet de quantifier la quantité d'information qui sépare deux distributions.

Nous avons ainsi généralisé la précédente approche à tout type de distance et à tout membre de ces deux classes de divergences [Anceaume et Busnel, 2013]. Au même titre que la codéviance, l'idée est ici d'étendre n'importe quelle distance ou divergence, notée ϕ, par une version agrégée dont la valeur est définie, pour tout paramètre de précision k, par

$$\widehat{\phi}_k(\mathbf{f}||\mathbf{f}') = \max_{\rho \in \mathcal{P}_k(\Omega)} \phi(\widehat{\mathbf{f}}_\rho||\widehat{\mathbf{f}'}_\rho)$$

Cette version agrégée, que nous dénoterons *métrique Sketch-ϕ*, est à nouveau indépendante de la distribution d'entrée des flux et ne présuppose aucune connaissance sur la taille de ceux-ci. Elle est ainsi capable de comparer tout couple de flux, d'identifier des corrélations le cas échéant, et

Algorithme 3.2 : Algorithme d'approximation d'une métrique Sketch-$\star$

Entrées : Deux flux de données σ_1 and σ_2 ; Une métrique ϕ ; Deux paramètres de précision ε et de sensibilité δ;

Sortie : L'estimation de la métrique Sketch-ϕ sur σ_1 et σ_2 : $\widehat{\phi}_k(\sigma_1, \sigma_2)$

1 $c_1 \leftarrow \lceil \ln \frac{1}{\delta} \rceil$; $c_2 \leftarrow \lceil \frac{e}{\varepsilon} \rceil$;

2 $C_{\sigma_1}[1..c_1][1..c_2] \leftarrow 0_{c_1,c_2}$; ▷ Matrice nulle de dimension $c_1 \times c_2$ ◁

3 $C_{\sigma_2}[1..c_1][1..c_2] \leftarrow 0_{c_1,c_2}$;

4 Choisir c_1 fonctions de hachage $h_\ell : \Omega \to [\![c_2]\!]$ parmi une famille 2-universelle;

5 **pour** $u \in \sigma_1$ **faire**

6 **pour** $\ell = 1$ **à** c_1 **faire**

7 $C_{\sigma_1}[\ell][h_\ell(u)] \leftarrow C_{\sigma_1}[\ell][h_\ell(u)] + 1$;

8 **pour** $v \in \sigma_2$ **faire**

9 **pour** $\ell = 1$ **à** c_1 **faire**

10 $C_{\sigma_2}[\ell][h_\ell(v)] \leftarrow C_{\sigma_2}[\ell][h_\ell(v)] + 1$;

11 **à la requête** $\widehat{\phi}(\sigma_1, \sigma_2)$ **retourner** $\max_{1 \le \ell \le c_1} \phi(C_{\sigma_1}[\ell], C_{\sigma_2}[\ell])$

plus généralement, d'acquérir une connaissance approfondie de la structure de ceux-ci.

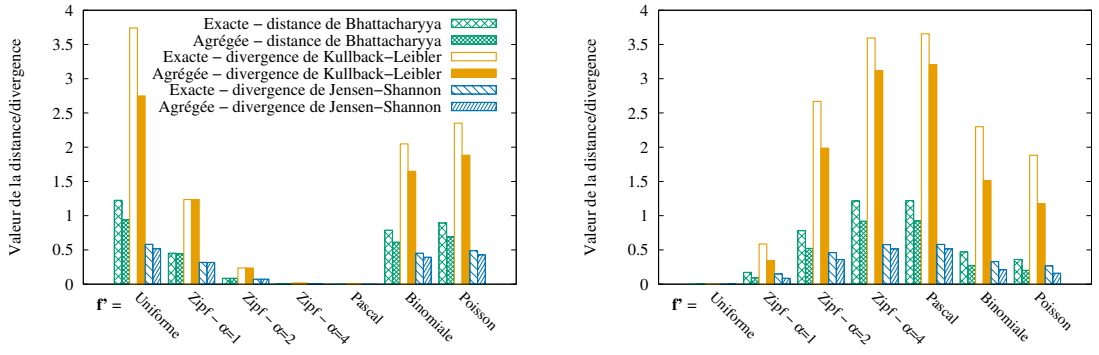
Il est à noter que, la définition d'une distance, ou d'une divergence étant contraire à celle de la codéviance, toute métrique Sketch-$\star$ est une sous-estimation de la valeur exacte de celle-ci, à l'inverse de la codéviance agrégée.

Nous avons montré que les métriques Sketch-$\star$ conservaient toutes les propriétés des distances (non-négativité, identité des indiscernables, symétrie et inégalité triangulaire), et des divergences (monotonie, convexité et linéarité) [Anceaume et Busnel, 2013]. Ainsi, par exemple, si ϕ est une f-divergence, alors la métrique Sketch-ϕ associée est également une f-divergence. Ce résultat est aussi valable pour les distances et les divergences de Bregman.

L'algorithme associé (*cf.* Algorithme 3.2) est identique à celui de la codéviance à l'exception de la ligne 11 pour laquelle il retourne la valeur maximale sur les c_1 projections aléatoires. Les propriétés d'agrégat linéaire et les résultats de complexité de cet algorithme restent ainsi valides dans le cadre des métriques Sketch-$\star$.

À nouveau, nous avons évalué expérimentalement cet algorithme afin d'étudier son comportement face à différentes distances et divergences, pour plusieurs types de flux. Nous avons notamment généré des flux suivant des distributions de loi uniforme, de Zipf, de Pascal, binomiales et de Poisson. Les flux générés étaient tous de taille $m = 200\,000$, et le nombre d'éléments distincts était au plus $n = 4\,000$. Les matrices de projections avaient pour paramètres $c_1 = 4$ et $c_2 = 200$. Nous avons ensuite appliqué à tout couple de distribution, trois métriques différentes :

Divergence de Kullback-Leibler Aussi appelée entropie relative (*cf.* Section 2.2.2.3).



(a) Vecteur de fréquences $\mathbf{f}$ de distribution de loi binomiale négative NB(3; 0.99) (ou de Pascal)

(b) Vecteur de fréquences $\mathbf{f}$ de distribution de loi uniforme

FIGURE 3.2 – Comparaison entre les métriques Sketch-$\star$ et les valeurs exactes en fonction des lois de distribution des flux entrants.

Divergence de Jensen-Shannon Elle est une version symétrisée de la KL-divergence. Elle est aussi connue sous le nom de Radius d'Information (IRad) ou divergence totale à la moyenne. Elle est définie par

$$\mathcal{D}_{JS}(\mathbf{f}||\mathbf{f}') = \frac{1}{2}D_{KL}(\mathbf{f}||\mathbf{p}) + \frac{1}{2}D_{KL}(\mathbf{f}'||\mathbf{p}), \text{ où } \mathbf{p} = \frac{1}{2}(\mathbf{f}+\mathbf{f}') \tag{3.6}$$

La racine carrée de cette divergence est par ailleurs une distance.

Distance de Bhattacharyya Elle est dérivée de la mesure de similarité entre deux distributions de loi multinomiale, aussi connue comme le coefficient de Bhattacharya (BC) [Bhattacharyya, 1943]. C'est une semi-distance car elle ne vérifie pas l'inégalité triangulaire. Elle est définie par

$$\mathcal{D}_B(\mathbf{f}||\mathbf{f}') = -\log(BC(\mathbf{f},\mathbf{f}')) \text{ où } BC(\mathbf{f},\mathbf{f}') = \sum_{i\in\Omega}\sqrt{f_i f'_i}. \tag{3.7}$$

La figure 3.2 illustre la justesse de notre algorithme par la comparaison de $\phi(\mathbf{f}||\mathbf{f}')$ avec $\widehat{\phi}_k(\mathbf{f}||\mathbf{f}')$, pour les trois métriques ci-dessus. En effet, les métriques Sketch-$\star$ estiment parfaitement lorsque les deux flux suivent la même distribution, quelle que soit la métrique ϕ (*e.g.*, *Pascal* dans la figure 3.2a, et *Uniforme* dans la figure 3.2b). Cette tendance se constate également lorsque les deux distributions sont « proches » (*e.g.*, entre *Zipf*-α=4 et *Pascal*, ainsi que pour α=2). Notre algorithme est donc un bon candidat comme méthode pour de l'inférence de paramètres de distribution par exemple.

Des évaluations plus approfondies, notamment sur des jeux de données réelles, sont disponibles dans [Anceaume et Busnel, 2013]. De plus, une extension de ces métriques Sketch-$\star$ dans le modèle de flux répartis continus est directe, à l'instar de la méthode utilisée pour la codéviance.

3.1.3 AnKLe : optimisation pour l'entropie relative

Finalement, nous nous sommes concentrés plus spécifiquement sur la divergence de Kullback-Leibler, afin d'en optimiser le calcul dans des conditions pratiques spécifiques. En effet, dans le cadre de l'inférence de distribution dont nous avons parlé dans la section précédente, il est pertinent de comparer la divergence du flux reçu localement avec le vecteur de fréquences uniforme.

Dans [Anceaume et Busnel, 2012, Anceaume *et al.*, 2012], nous proposons d'optimiser l'algorithme précédent afin d'améliorer l'approximation de la KL-divergence entre un flux σ (de vecteur de fréquences $\mathbf{f}$) et un autre flux avec les mêmes caractéristiques dimensionnelles (F_0 et F_1 spécifiquement), mais de loi uniforme, dont nous noterons le vecteur de fréquences $\mathbf{u}$. Pour cela, notre point de départ a été la réécriture de la KL-divergence comme suit

$$\begin{aligned}\mathcal{D}(\mathbf{f}||\mathbf{u}) &= \frac{1}{m}\left(\sum_{i\in\Omega} f_i \log\left(\frac{f_i}{m}\right) - \sum_{i\in\Omega} f_i \log\left(\frac{1}{n}\right)\right) \\ &= \log(n) - \log(m) + \frac{1}{m}\sum_{i\in\Omega} f_i \log\left(f_i\right). \end{aligned} \tag{3.8}$$

Pour simplifier l'écriture, nous supposons ici sans perdre en généralité que $F_0(\sigma) = n$.

Ainsi, l'approximation optimisée de la KL-divergence passera par l'estimation *(i)* du nombre d'éléments distincts du flux (*i.e.,* F_0) ainsi que *(ii)* de la norme de l'entropie de ce flux (*i.e.,* F_H). S'il existe des algorithmes d'estimation de F_0 efficaces tels que celui proposé par [Kane *et al.*, 2010] (fondé sur l'algorithme LogLog de [Durand et Flajolet, 2003]), tel n'était pas le cas pour la norme de l'entropie. Nous avons ainsi proposé une extension de l'approche introduite par Alon *et al.* [1996].

Spécifiquement, nous proposons un estimateur non-paramétrique $X = S_{i,j}$, conçu pour avoir une espérance égale à la norme de l'entropie et une variance faible. Plus précisément, nous avons

$$X = m(g\log g - (g-1)\log(g-1)), \tag{3.9}$$

où g est la variable aléatoire représentant le nombre d'occurrences d'un élément u pioché au hasard sur le flux. Ce tirage s'effectue par le choix aléatoire d'une position j sur le flux. La variable aléatoire g compte le nombre d'occurrences de $u = a_j$ à partir de la position j. Formellement, g est défini par $g = |\{i \in \mathbb{N} \mid i \geq j, a_i = a_j\}|$.

Nous pouvons montrer, à l'instar de [Alon *et al.*, 1996, Lall *et al.*, 2006], que l'estimateur X est non-biaisé, *i.e.,*

$$\mathbb{E}[X] = \frac{1}{m}\sum_{u\in\Omega}\sum_{j=1}^{f_u} m(j\log j - (j-1)\log(j-1)) = \frac{m}{m}\sum_{u\in\Omega} f_u \log f_u = F_H. \tag{3.10}$$

Pour améliorer la fiabilité de l'estimation, $s_1 \times s_2$ estimateurs sont utilisés et réunis dans la

Algorithme 3.3 : AnKLe : estimateur de la KL-divergence

Entrées : Un flux de données σ de taille m ; Deux paramètres de précision ε et de sensibilité δ

Sortie : L'estimation de la KL-divergence $\mathcal{D}(\mathbf{f}_\sigma||\mathbf{u})$

Choisir aléatoirement $s_1 \times s_2$ entiers parmi $[\![m]\!]$;

pour tous les $i \in [\![s_1]\!], j \in [\![s_2]\!]$ **faire** $S[i,j] \leftarrow (\perp, \perp)$;

pour tous les $u \in \sigma$ **faire**

  Tâche T_1 *(Estimation de n)* **:**

    $\hat{F}_0 \leftarrow$ Algorithme proposé par Kane *et al.* [2010] appliqué à u;

  Tâche T_2 *(Estimation des fréquences des éléments surabondants)* **:**

    $\hat{B} \leftarrow$ Algorithme proposé par Metwally *et al.* [2005] appliqué à u;

  Tâche T_3 *(Estimation de la norme de l'entropie)* **:**

    pour tous les éléments i de la matrice S tels que $(s_i, r_i) \neq (\perp, \perp)$ **faire**

      si $s_i = u$ **alors** $r_i \leftarrow r_i + 1$;

    si j est l'un des $s_1 \times s_2$ entiers aléatoires choisis **alors**

      Assigner $(u, 1)$ au premier élément non encore initialisé de S;

$\hat{B} \leftarrow$ l'ensemble des (s_i, r_i) des k éléments les plus fréquents de $\hat{B}$ tels que $r_i > e$;

pour tous les éléments i de la matrice S **faire**

  si $(s_i, -) \in \hat{B}$ **alors**

    $(s_i, r_i) \leftarrow (\perp, \perp)$; ▷ s_i est un des éléments surabondants, identifié en tâche T_2 ◁

  sinon

    $(s_i, r_i) \leftarrow (s_i, m\,(r_i \log r_i - (r_i - 1)\log(r_i - 1)))$ ▷ Calcul de l'estimateur par Rel. 3.9 ◁

$Y_{\hat{B}} \leftarrow \sum_{(s_i, r_i) \in \hat{B}} r_i \log r_i$; ▷ Norme de l'entropie des éléments surabondants ◁

$Y_S \leftarrow \text{médiane}_{1 \leq i \leq s_1} \left(\frac{1}{s_2} \sum_{j=1}^{s_2} S_{ij} \right)$; ▷ Norme de l'entropie des autres éléments ◁

$p \leftarrow 1 - \max\left(0, \dfrac{\min\left(Y_S, Y_{\hat{B}}\right) - m}{10 \cdot m}\right)$;

retourner $\hat{D} = \log \hat{F}_0 - \log m + \frac{p}{m}\left(Y_S + Y_{\hat{B}}\right)$;

matrice $S_{i,j}$ (avec $1 \leq i \leq s_1$ et $1 \leq j \leq s_2$), chacun échantillonnant une position aléatoire distincte du flux.

Cependant, pour estimer correctement la KL-divergence de n'importe quel flux, il est essentiel de prendre en considération les éléments surabondants (dont le nombre d'occurrences dans les flux est très grand par rapport à m). En cas d'existence de tels éléments, l'estimateur précédent est incapable de calculer la norme de l'entropie en mémoire bornée [Chakrabarti *et al.*, 2007]. En effet, par analogie avec les travaux de Alon *et al.* [1996], la variance de l'estimateur explose littéralement.

L'algorithme 3.3 présente le pseudo-code de notre approche. Notons que dans un souci de simplification, nous supposons ici la connaissance de la taille du flux m. Cependant, cet algorithme peut parfaitement s'affranchir de cette hypothèse par l'utilisation de la méthode proposée par [Chakrabarti *et al.*, 2007]. Cet algorithme est donc divisé en trois phases parallèles estimant respectivement *(1)* la valeur de F_0, *(2)* la fréquence des éléments surabondants (en utilisant l'algorithme SpaceSaving proposé par Metwally *et al.* [2005]) et *(3)* la norme de l'entropie sur les autres éléments. Finalement, afin d'éviter de comptabiliser deux fois un même élément dans les deux derniers estimateurs, nous pondérons la contribution des deux derniers termes (*cf.* ligne 22).

Dans [Anceaume et Busnel, 2014b], nous avons démontré que cet algorithme est une (ε, δ)-approximation relative de la KL-divergence, utilisant

$$\mathcal{O}\left(\log n + \frac{1}{\varepsilon^2} + \left(\frac{1}{\varepsilon} + \frac{\mu}{\varepsilon^2}\log\frac{1}{\delta}\right)(\log n + \log m)\right)$$

bits de mémoire où $\mu = (\log m + \log e - 1)$ si $F_H^e \geq \frac{m^2}{\Delta m_e}$ (sinon $\mu = (n - \frac{1}{\varepsilon} - 1)$), Δ est une constante strictement positive, m_e représente le nombre d'occurrences d'éléments non-surabondants dans le flux (*i.e.*, $m_e = m - \sum_{i \in \hat{B}} f_i$) et F_H^e est la norme de l'entropie pour ces éléments uniquement. Nous proposons également une extension de cet algorithme au modèle de flux répartis continus, ayant une complexité en communication de $O(s \log m \log n)$ bits (avec m représentant la longueur maximale des s flux entrants).

À nouveau, nous avons évalué les performances de notre algorithme par des expérimentations, sur des flux de longueur $m = 200\,000$ contenant au plus $n = 1\,000$ éléments distincts, suivant différentes distributions (uniforme, Zipf, Pascal et Poisson). La figure 3.3 présente la moyenne et l'écart-type pour des approximations de la KL-divergence obtenues avec trois algorithmes différents (chacun exécuté 1 000 fois) : *(i)* AnKLe présenté ci-dessus, *(ii)* AMS correspondant aux travaux de Alon *et al.* [1996] avec l'estimateur de la relation 3.9 et *(iii)* CCM adapté de la proposition de Chakrabarti *et al.* [2007]. Pour des raisons de lisibilité, la courbe a été recadrée : la valeur retournée par CCM pour la loi de Zipf avec $\alpha = 1$ atteignait *8,3*. Ces résultats montrent que AnKLe surpasse les deux autres algorithmes en terme de précision pour tous les scénarii (même ceux pour lesquels CCM a été conçu, *i.e.,* des lois fortement biaisées telles que Zipf avec $\alpha = 4$). Un exception peut être remarquée avec la loi de Zipf avec $\alpha = 2$ pour laquelle AMS

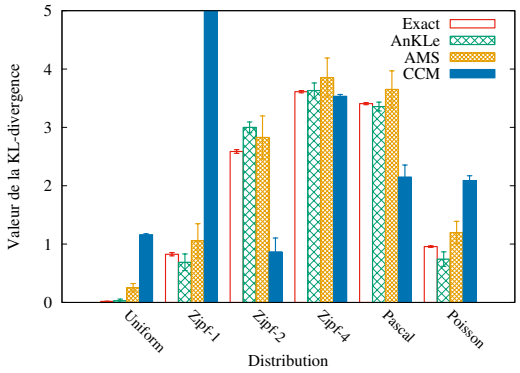


Figure 3.3 – Comparaison entre la valeur exacte de la KL-divergence et différents estimateurs

obtient une meilleure moyenne qu'AnKLe. Cependant, les estimations d'AnKLe sont plus stables et robustes, avec un écart type 4 fois plus petit que celui d'AMS dans ce scénario.

3.2 Détection d'éléments surabondants

Nous avons touché du doigt la problématique des éléments surabondants dans la section précédente. La détection de ces éléments très fréquents est d'autant plus fondamentale dans des domaines d'application tels que la sûreté de fonctionnement (notamment pour les attaques DDoS, *cf.* Section 4.2.1 et [Manjhi *et al.*, 2005, Yi et Zhang, 2013]), ou les bases de données réparties (pour l'estimation de charge impliquée par une jointure entre clés surabondantes [Alon *et al.*, 1999, Poosala et Ioannidis, 1996, Swami et Schiefer, 1994, Vengerov *et al.*, 2015]).

De nombreux travaux connexes ont été proposés dans le cadre du modèle de flux unique [Charikar *et al.*, 2004, Cormode et Muthukrishnan, 2005, Demaine *et al.*, 2003, Karp *et al.*, 2003, Metwally *et al.*, 2005, Misra et Gries, 1982], ou en fenêtre glissante [Rivetti *et al.*, 2015a, Golab *et al.*, 2003]. Ces travaux relèvent d'un intérêt particulier dans les modèles à flux répartis, où la complexité de communication doit être minimisée, sans compromettre la latence de détection ni introduire de faux négatifs (la non détection d'éléments surabondants) [Anceaume *et al.*, 2015c, Manjhi *et al.*, 2005, Yi et Zhang, 2013, Zhao *et al.*, 2010, 2006].

Nous avons proposé une solution au problème de détection et d'identification d'éléments surabondants spécifiques, dénotés *icebergs*, tout en garantissant les deux propriétés ci-dessus [Anceaume *et al.*, 2015c]. Plusieurs structures de données de faible taille sont utilisées à cette fin, pour *(i)* estimer la fréquence locale de chaque élément d'un flux entrant, *(ii)* stocker temporairement les éléments surabondants potentiels et *(iii)* mémoriser ceux qui ont été réellement identifiés. De plus, nous considérons la présence d'un adversaire *adaptatif*, qui cherche à éviter la détection de ces éléments en les répartissant judicieusement parmi les s flux. Ainsi, il est ainsi capable d'insérer autant d'éléments que nécessaire pour augmenter ou réduire le nombre d'éléments surabondants, par rapport à l'état courant du coordinateur et des s nœuds, ou de réordonner les flux

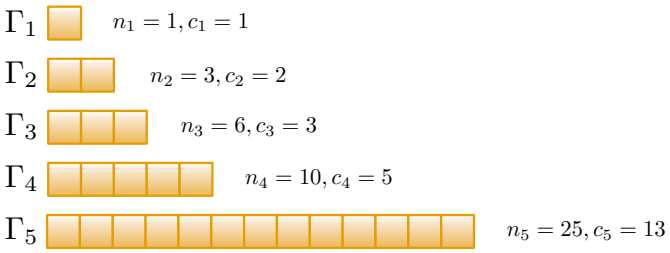


FIGURE 3.4 – Structure de données des Γ sur un nœud $i \in \mathcal{S}$, avec $\Theta = 0,04$, $r = 0,5$ et $s = 20$.

à sa guise (*cf.* Section 2.1.6).

Le problème de détection (et d'identification) d'icebergs a été formalisé par [Estand et Varghese, 2003], puis adapté dans différents contextes [Manjhi *et al.*, 2005, Zhao *et al.*, 2010]. Ici, nous étendons ce problème au cadre du modèle de flux répartis continus. Ce problème consiste, pour le coordinateur, à identifier rapidement tout élément u excédant globalement une proportion Θ donnée du flux.

Formellement, nous noterons σ l'union des s flux $\sigma^{(i)}$ ($i \in \mathcal{S}$) depuis l'origine. Dénotons alors par m le nombre d'éléments de σ. Nous avons donc $m = \sum_{i \in \mathcal{S}} m_i$. Enfin, f_u représentera le nombre total d'occurrences d'un élément u dans σ. Ainsi, pour tout seuil $\Theta \in (0, 1]$, tout paramètre d'approximation $\varepsilon \in [0, 1]$ et toute probabilité d'erreur $\delta \leq 1/2$, le problème de détection et d'identification d'icebergs consiste, pour le coordinateur :

- à retourner tout élément u si $f_u \geq \Theta m$, et
- à ne jamais retourner un élément u si $f_u < (1 - \varepsilon)\Theta m$.

Notre solution œuvre comme suit. Supposons dans un premier temps que pour tout élément u reçu sur un flux $\sigma^{(i)}$, celui-ci est étiqueté par sa probabilité $p_u^{(i)}$. Chaque nœud $i \in \mathcal{S}$ reçoit donc un flux, mais ne garde trace que des éléments très fréquents, *i.e.,* ceux qui représentent localement une proportion au moins égale à Θ du flux $\sigma^{(i)}$ (les autres étant simplement ignorés). Afin de traiter proportionnellement l'urgence des éléments extrêmement fréquents par rapport aux simplement fréquents, chaque nœud stocke ces éléments dans différents tampons. L'intervalle $[\Theta, 1]$ est découpé en ℓ sous-intervalles de taille géométrique, dénotée c_k, et chacun des tampons Γ_k ($k \in [\![\ell - 1]\!]$) est associé aux éléments ayant une probabilité d'occurrence correspondant au $k^{\text{ème}}$ intervalle : $]\Theta + (1 - \Theta)/2^k, \Theta + (1 - \Theta)/2^{k-1}]$. Dans un souci d'exhaustivité, Γ_ℓ contient les éléments u ayant une probabilité d'occurrence $p_u^{(i)}$ telle que $\Theta \leq p_u^{(i)} \leq \Theta + (1 - \Theta)/2^{\ell-1}$. La taille c_k de chaque tampon Γ_k est fixée à $\lceil \lambda n_k \rceil$, où n_k est le nombre maximum d'éléments distincts pouvant intégrer le $k^{\text{ème}}$ tampon, et $0 < \lambda \leq 1$ est un paramètre utilisateur. Étant donné que Γ_k ($k \in [\![\ell - 1]\!]$) contient les éléments ayant une probabilité d'occurrence $p_u^{(i)}$ vérifiant $\Theta + (1 - \Theta)/2^k < p_u^{(i)} \leq \Theta + (1 - \Theta)/2^{k-1}$, le nombre d'éléments possibles est donc

$$n_k = \left\lfloor 1/(\Theta + (1 - \Theta)/2^k) \right\rfloor \text{ et } n_\ell = \lfloor 1/\Theta \rfloor .$$

La figure 3.4 illustre cette structure de données. Ainsi, dès qu'un tampon est rempli, son contenu est alors envoyé au coordinateur. Ce dernier interroge ensuite les autres nœuds afin de déterminer si un iceberg est effectivement présent.

De plus, la distribution des éléments dans chaque flux $\sigma^{(i)}$ étant inconnue, il est probable que ces tampons ne soient jamais remplis intégralement. Afin d'assurer que tout iceberg potentiel soit transmis au coordinateur, un minuteur τ_k est démarré dès qu'un élément est ajouté à Γ_k, et incrémenté à chaque réception d'un nouvel élément sur le flux $\sigma^{(i)}$. Le délai de temporisation de τ_k est initialisé à $H_{\lfloor 1/\Theta \rfloor}/\Theta$, où H_n représente le $n^{\text{ème}}$ harmonique défini par $H_0 = 0$ et $H_n = 1 + 1/2 + \cdots + 1/n$.

Enfin, à chaque détection d'un iceberg réel, le coordinateur en informe à la fois l'application et les s nœuds. Ces derniers l'ajoutent alors à une file spécifique Λ_i, de taille inférieure à $\lceil 1/\Theta \rceil$. Cela évite de retransmettre au coordinateur à maintes reprises un potentiel iceberg déjà détecté. Les algorithmes 3.4 et 3.5 présentent le pseudo-code exécuté respectivement sur les s nœuds de $\mathcal{S}$ d'une part et sur le coordinateur d'autre part.

Revenons à présent sur l'hypothèse précédente. Connaître la probabilité d'occurrence d'un élément u reçu sur $\sigma^{(i)}$ est irréaliste. Cependant, nous pouvons l'estimer en temps réel, en utilisant une structure de données utilisant un faible espace mémoire, dénommée *Count-Min (CM) Sketch* [Cormode et Muthukrishnan, 2005]. Le CM Sketch estime la fréquence $f_u^{(i)}$ de l'élément u avec une erreur bornée par un facteur ε avec une probabilité δ. Cet estimateur utilise une matrice $\hat{F}_i$ de $s_1 \times s_2$ compteurs avec $s_1 = \lceil \log(1/\delta) \rceil$ et $s_2 = \lceil e/\varepsilon \rceil$, ainsi que s_1 fonctions de hachage 2-universelles $h_1, \ldots, h_{s_1}$. À la réception de chaque élément u sur le flux, un compteur pour chaque ligne est incrémenté, *i.e.*, $\hat{F}_i[j][h_j(u)]$ est incrémenté pour tout $j \in [\![s_1]\!]$. Lors d'une requête d'estimation de fréquence de u, le minimum des valeurs parmi $\{\hat{F}_i[j][h_j(u)]\}_{j \in [\![s_1]\!]}$ est retourné. La taille mémoire requise par un CM Sketch est donc proportionnelle à $\frac{1}{\varepsilon} \log \frac{1}{\delta}$ et permet de garantir la fiabilité suivante

$$\forall u \in \Omega, \mathbb{P}\{\mid \hat{f}_u - f_u \mid > \varepsilon(m - f_u)\} < \delta. \tag{3.11}$$

Son pseudo-code est présenté dans l'algorithme 3.6.

Nous montrons ainsi dans Anceaume *et al.* [2015c] que notre solution est une (ε, δ)-approximation permettant de résoudre le problème de la détection d'icebergs, pour tout $\varepsilon \in [0, 1]$ et $\delta \leq 1/2$, avec une faible complexité mémoire. De plus, nous fournissons une borne supérieure précise sur le nombre de bits échangés par notre solution, la solution présentée ne transmettant en moyenne pas plus de

$$2mS(\log m + \log n) \sum_{k=1}^{\ell} \frac{c_k}{\sum_{i=0}^{\tau_k - 1} \mathbb{P}\{T_{c_k, n_k}(v) > i\}} \text{ bits.} \tag{3.12}$$

Ici, la somme des probabilités au dénominateur de la relation 3.12 représente le temps d'atteinte moyen de l'émission d'un message, qu'il soit dû au remplissage du tampon k ou à la fin du délai de

Algorithme 3.4 : Détection omnisciente d'icebergs avec mémoire bornée, sur un nœud $i \in \mathcal{S}$

Entrées : Un flux de données $\sigma^{(i)}$; Un seuil Θ qualifiant un élément d'iceberg ;
Un paramètre λ définissant proportionnellement la taille des ℓ tampons Γ;

pour chaque $k \in \{1, \ell - 1\}$ **faire**
  $\Gamma_k \leftarrow \emptyset$; $n_k \leftarrow \lfloor 1/(\Theta + (1-\Theta)/2^k) \rfloor$; $c_k \leftarrow \lceil \lambda n_k \rceil$; $\tau_k \leftarrow$ (desactivé, 0);
$\Gamma_\ell \leftarrow \emptyset$; $n_\ell \leftarrow \lfloor 1/\Theta \rfloor$; $c_\ell \leftarrow \lceil \lambda n_\ell \rceil$; $\tau_\ell \leftarrow$ (desactivé, 0) ;
$m_i \leftarrow 0$; *Freq* $\leftarrow \emptyset$; $\Lambda_i \leftarrow \emptyset$;
pour tous les $u \in \sigma^{(i)}$ **faire**
  $m_i \leftarrow m_i + 1$;
  $F[u] \leftarrow F[u] + 1$;
  si $p_u^{(i)} \geq \Theta$ **alors**
    si $p_u^{(i)} \leq \Theta + (1-\Theta)/2^{\ell-1}$ **alors**
      $k \leftarrow \ell$;
    sinon
      $k \leftarrow \min\{h \in \{1, \ell-1\} \mid \Theta + (1-\Theta)/2^h < p_u^{(i)}\}$;
    si $u \notin \Gamma_k$ **alors**
      si $\Gamma_k = \emptyset$ **alors** $\tau_k \leftarrow$ (activé, 0);
      $\Gamma_k \leftarrow \Gamma_k \cup \{u\}$;
      si $|\Gamma_k| = c_k$ **alors**
        pour chaque $u \in \Gamma_k$ **faire** *Freq* $\leftarrow$ *Freq* $\cup \{(u, F[u])\}$;
        envoi d'un message ("détection", m_i, *Freq*) au coordinateur;
        $\Gamma_k \leftarrow \emptyset$; *Freq* $\leftarrow \emptyset$; $\tau_k \leftarrow$ (desactivé, 0);
  pour chaque minuteur actif τ_k, $k \in \{1, \ell\}$ **faire**
    $\tau_k \leftarrow$ (activé, $\tau_k + 1$);
    si $\tau_k > H_{\lfloor 1/\Theta \rfloor}/\Theta$ **et** $\Gamma_k \neq \emptyset$ **alors**
      pour chaque $u \in \Gamma_k$ **faire** *Freq* $\leftarrow$ *Freq* $\cup \{(u, F[u])\}$;
      envoi d'un message ("détection", m_i, *Freq*) au coordinateur;
      $\Gamma_k \leftarrow \emptyset$; *Freq* $\leftarrow \emptyset$; $\tau_k \leftarrow$ (desactivé, 0);
à la réception d'un message ("freq ?", *Freq*) venant du coordinateur **faire**
  pour chaque $u \in Freq$ **faire**
    $Freq \leftarrow (u, F[u])$;
    si $\exists k \in \{1, \ell\}, u \in \Gamma_k$ **alors** $\Gamma_k \leftarrow \Gamma_k \setminus \{u\}$;
  envoi d'un message ("freq", m_i, *Freq*) au coordinateur;
à la réception d'un message ("iceberg", F) venant du coordinateur **faire**
  ajouter F à la file Λ_i;

Algorithme 3.5 : Détection d'icebergs sur le coordinateur

Entrées : Un seuil Θ qualifiant un élément d'iceberg ;

Sortie : L'ensemble F des icebergs détectés ;

à la réception d'un message ("détection", m', *Freq*) venant de $i \in \mathcal{S}$ **faire**
  $m \leftarrow m'$; $F \leftarrow$ *Freq*;
  pour chaque $(k, f_k) \in$ Freq **faire**
    $(k, f_k) \leftarrow (k, \perp)$;
  pour chaque $j \neq i \in \mathcal{S}$ **faire**
    envoi d'un message ("freq ?", *Freq*) à j;
  pour chaque message ("freq", m', *Freq*) reçu de $j, j \neq i$ **faire**
    $m \leftarrow m + m'$;
    pour chaque $(k, f_k) \in F$ **faire**
      mise à jour de F : $f_k \leftarrow f_k + f'_k$ pour chaque $(k, f'_k) \in$ *Freq*;
  pour chaque $(k, f_k) \in F$ **faire**
    si $f_k < \Theta m$ **alors**
      $F \leftarrow F \setminus \{(k, f_k)\}$;
  si $F \neq \emptyset$ **alors**
    envoi d'un message ("iceberg", F) à chaque $i \in \mathcal{S}$;
    retourner F;

Algorithme 3.6 : Count-Min Sketch proposé par Cormode et Muthukrishnan [2005]

Entrées : Un flux de données σ ; Deux paramètres de précision ε et de sensibilité δ;

Sortie : L'estimation de la fréquence d'occurrence $\hat{f}_u$ de n'importe quel élément u lu sur le flux;

$s_1 \leftarrow \lceil \log(1/\delta) \rceil$;
$s_2 \leftarrow \lceil e/\varepsilon \rceil$;
$\hat{F}[1..s_1][1..s_2] \leftarrow 0_{s_1,s_2}$;
choisir s_1 fonctions de hachage $h_1, \ldots, h_{s_1} : \Omega \rightarrow [\![s_2]\!]$ parmi une famille 2-universelle;
pour tous les $u \in \sigma$ **faire**
  pour $j = 1$ **à** s_1 **faire**
    $\hat{F}[j][h_j(u)] \leftarrow \hat{F}[j][h_j(u)] + 1$;
à la requête $\hat{f}_u$ **retourner** $\min_{1 \leq j \leq s_1} \hat{F}[j][h_j(u)]$;

Algorithme 3.7 : Détection non-omnisciente d'icebergs avec mémoire bornée, sur un nœud $i \in \mathcal{S}$

Entrées : Un flux de données $\sigma^{(i)}$; Un seuil Θ qualifiant un élément d'iceberg ;
Un paramètre λ définissant proportionnellement la taille des ℓ tampons Γ ;
Deux paramètres de précision ε et de sensibilité δ;

1 $s_1 \leftarrow \lceil \log(1/\delta) \rceil$;
2 $s_2 \leftarrow \lceil 2(1-\Theta)/\varepsilon\Theta \rceil$;
3 $\hat{F}[1..s_1][1..s_2] \leftarrow 0_{s_1,s_2}$;
4 choisir s_1 fonctions de hachage $h_1, \ldots, h_{s_1} : \Omega \to [\![s_2]\!]$ parmi une famille 2-universelle;
5 **pour chaque** $k \in \{1, \ell - 1\}$ **faire**
6 $\quad \Gamma_k \leftarrow \emptyset$; $n_k \leftarrow \lfloor 1/(\Theta + (1-\Theta)/2^k) \rfloor$; $c_k \leftarrow \lceil \lambda n_k \rceil$; $\tau_k \leftarrow$ (desactivé, 0);
7 $\Gamma_\ell \leftarrow \emptyset$; $n_\ell \leftarrow \lfloor 1/\Theta \rfloor$; $c_\ell \leftarrow \lceil \lambda n_\ell \rceil$; $\tau_\ell \leftarrow$ (desactivé, 0) ;
8 $m_i \leftarrow 0$; *Freq* $\leftarrow \emptyset$; $\Lambda_i \leftarrow \emptyset$;
9 **pour tous les** $u \in \sigma^{(i)}$ **faire**
10 $\quad$ **Tâche** T_1 *(Estimation des fréquences par le Count-Min Sketch)* **:**
11 $\quad\quad$ Algorithme 3.6 proposé par Cormode et Muthukrishnan [2005] appliqué à u;
12 $\quad$ **Tâche** T_2 *(Détection des éléments surabondants locaux au nœud i)* **:**
$\quad\quad$ ▷ Algorithme 3.4 tel que : ◁
$\quad\quad$ ▷ les lignes 8–12 sont modifiées comme suit : ◁
13 $\quad\quad$ **si** $\hat{f}_u^{(i)}/m_i \geq \Theta$ **alors**
14 $\quad\quad\quad$ **si** $\hat{f}_u^{(i)}/m_i \leq \Theta + (1-\Theta)/2^{\ell-1}$ **alors**
15 $\quad\quad\quad\quad k \leftarrow \ell$;
16 $\quad\quad\quad$ **sinon**
17 $\quad\quad\quad\quad k \leftarrow \min\{h \in \{1, \ell-1\} \mid \Theta + (1-\Theta)/2^h < \hat{f}_u^{(i)}/m_i\}$;
$\quad\quad\quad$ (...)
$\quad\quad$ (...)
$\quad\quad$ ▷ les lignes 17 et 23 sont modifiées comme suit : ◁
18 $\quad\quad$ **pour chaque** $u \in \Gamma_k$ **faire** *Freq* $\leftarrow$ *Freq* $\cup (u, \hat{f}_u^{(i)})$;
$\quad\quad$ (...)
$\quad\quad$ ▷ les lignes 27–28 sont modifiées comme suit : ◁
19 $\quad\quad$ **pour chaque** $u \in Freq$ **faire**
20 $\quad\quad\quad Freq \leftarrow (u, \hat{f}_u^{(i)})$;
$\quad\quad\quad$ (...)
$\quad\quad$ (...)
(...)

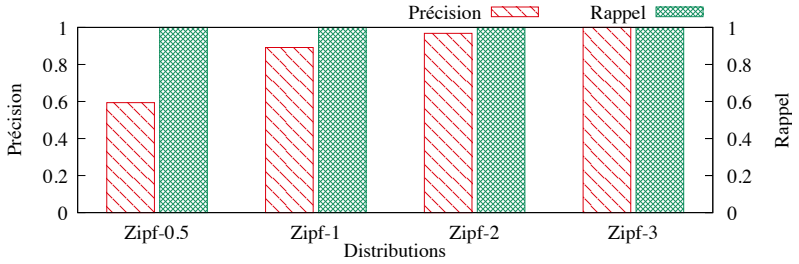


FIGURE 3.5 – Précision et rappel en fonction du type de flux entrant pour $\Theta = 0.005$, où $m = 2\,000\,000$ et $n = 10\,000$.

temporisation τ. Pour déterminer cette valeur, nous nous sommes intéressés à une généralisation du problème du *collecteur de coupons*. Nous avons en effet besoin de déterminer $T_{c,n}$, le nombre de coupons qui doivent être tirés avec remise pour obtenir $c < n$ coupons distincts à partir de l'ensemble $[\![n]\!]$. Nous avons étudié le comportement stochastique de ce nombre, puis extrait sa loi et son espérance [Anceaume *et al.*, 2015d]. Nous noterons dans la suite $S_{i,n}$, l'ensemble des ensembles d'éléments de $[\![n]\!]$ de taille exactement égale à i, *i.e.*, $S_{i,n} = \{J \subseteq \{1,\ldots,n\} \mid |J| = i\}$. Nous avons ainsi $S_{0,n} = \{\emptyset\}$. De plus, pour tout $J \in S_{i,n}$, nous définissons $P_J = \sum_{j\in J} p_j$, avec $P_\emptyset = 0$. Enfin, étant donné que le nombre $T_{c,n}$ dépend du vecteur de probabilité $\mathbf{p} = (p_1,\ldots,p_n)$, nous utiliserons la notation $T_{c,n}(\mathbf{p})$ plutôt que $T_{c,n}$, impliquant de fait que la dimension du vecteur $\mathbf{p}$ est n. Nous avons donc les égalités suivantes, pour tout $n \geq 1$, c dans $[\![n]\!]$ et $h \geq 1$:

$$\mathbb{P}\{T_{c,n}(\mathbf{p}) > h\} = \sum_{i=0}^{c-1} (-1)^{c-1-i} \binom{n-i-1}{n-c} \sum_{J\in S_{i,n}} (p_0 + P_J)^h, \tag{3.13}$$

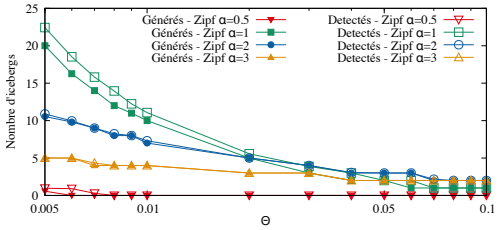
$$\mathbb{E}[T_{c,n}(\mathbf{p})] = \sum_{i=0}^{c-1} (-1)^{c-1-i} \binom{n-i-1}{n-c} \sum_{J\in S_{i,n}} \frac{1}{1-(p_0+P_J)}, \tag{3.14}$$

De plus, par [Anceaume *et al.*, 2016], pour tout $n \geq 1$, tout $c \in [\![n]\!]$, et $0 < \Theta \leq p_i$ pour $i \in [\![n]\!]$, nous avons

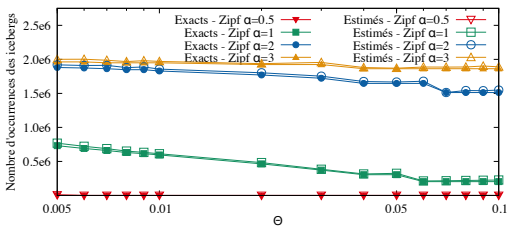
$$\mathbb{E}[T_{c,n}(\mathbf{p})] \leq H_{\lfloor 1/\Theta \rfloor}/\Theta. \tag{3.15}$$

Enfin, nous avons prouvé que notre solution omnisciente (algorithmes 3.4 et 3.5) résout le problème de détection d'icebergs et son adaptation de l'algorithme 3.7 est une (ε,δ)-approximation relative de cette solution utilisant $\mathcal{O}((\log n + \log m)\log(1/\delta)(1/\Theta - 1)/\varepsilon + (\log s \log n)/\Theta)$ bits de mémoire seulement sur chaque nœud.

Les figures 3.5 et 3.6a présentent à la fois la précision et le rappel de notre solution, en fonction de différentes distributions de flux et différentes valeurs de Θ. La figure 3.6b illustre l'estimation de fréquence globale (nombre d'occurrences totales sur l'union des flux) des icebergs. Par précision, nous définissons le nombre global d'icebergs réels détectés par notre solution divisé par le nombre total d'éléments détectés comme tels. Pour rappel, nous définissons le nombre total d'icebergs



(a) Nombre d'icebergs générés et détectés.



(b) Fréquence globale des icebergs (générées et estimées).

FIGURE 3.6 – Efficacité et précision de notre solution, en fonction de Θ, où $m = 2\ 000\ 000$ et $n = 10\ 000$.

réels détectés par notre solution divisé par le nombre total d'icebergs réellement présents.

Les résultats numériques confirment clairement que pour toute valeur de Θ, le rappel est toujours égal à 1 : tous les icebergs ont été détectés (*i.e.*, pas de faux négatifs). Les figures 3.5 et 3.6a illustrent également que la précision est très haute pour des distributions de loi de Zipf de paramètre $\alpha \geq 1$, mais diminue ensuite. En effet, pour des petites valeurs de α, il existe peu d'éléments fréquents, et la différence de fréquence entre des véritables icebergs et ceux dont la probabilité d'occurrence est inférieurement proche à Θ est toute petite (de l'ordre de $1,5 \times 10^{-3}$ pour une valeur de $\Theta = 5 \times 10^{-3}$). La figure 3.6b illustre bien la fiabilité de notre algorithme pour déterminer la taille globale des icebergs.

3.3 Approximation de fréquence d'occurrence

Nous avons vu dans les sections précédentes que l'utilisation de projections aléatoires permettait de réduire significativement l'empreinte mémoire de nos algorithmes, mais qu'elle possède deux limitations : l'influence sur la précision des estimateurs est manifeste et leur extension dans le modèle à fenêtre glissante n'est pas triviale.

Dans cette section, nous nous concentrons sur l'algorithme *Count-Min sketch* proposé par Cormode et Muthukrishnan [2005] et présenté en algorithme 3.6. Cette structure de données a prédominé jusqu'à présent les autres solutions, en terme de complexité d'espace et de temps, pour garantir une (ε, δ)-approximation additive sur l'estimation des fréquences d'éléments. Cependant, étant donné que s_2 est généralement bien plus petit que F_0, le nombre de collisions de hachés est très important (notamment pour un très grand flux, *cf.* relation 3.11). Ceci affecte principalement les estimations de fréquence des éléments qui ne sont pas fortement fréquents dans σ. Pour atténuer cet effet, deux approches ont été majoritairement proposées par le passé. Dimitropoulos *et al.* [2008] suggèrent de ne conserver que les éléments récents, permettant de réduire ainsi le nombre d'éléments distincts apparaissant dans le flux. Cependant, si cette approche réduit effectivement le nombre de collisions, elle ouvre la porte à des comportements malveillants, tels que des attaques latentes [Frye et Southerington, 2012], où un adversaire inonde le flux de

temps à autre, pour éclipser la majeure partie des éléments corrects dans l'agrégat. Une seconde approche, proposée par Estand et Varghese [2003], consiste à limiter le nombre de mises à jour de l'agrégat, en incrémentant uniquement le plus petit compteur des hachés, et en affectant aux autres compteurs correspondants, le maximum entre leur valeur actuelle et celle du plus petit compteur incrémenté. Comme nous le verrons dans les résultats expérimentaux, cette solution a tendance à uniformiser les distributions et ne s'applique qu'à des flux composés de valeurs positives.

3.3.1 Réduction de l'erreur d'approximation

Nous avons proposé CASE (*Count-Any Sketch Estimator*), une approche alternative pour réduire l'impact des collisions sur l'estimation de fréquence des éléments [Anceaume *et al.*, 2015b]. L'intuition de notre solution repose sur le repérage des éléments surabondants du flux, puis à retrancher leur poids dans l'agrégat. Ainsi, les éléments en collision avec ceux-ci voient la surestimation de leur approximation de fréquence drastiquement diminuer.

De plus, les éléments qui biaisent majoritairement la distribution du flux ayant été traités à part et retirés de l'agrégat, la variance des compteurs de chaque projection aléatoire est significativement diminuée. La propriété d'universalité des fonctions de hachage choisies nous permet d'estimer le nombre d'éléments en collision sur un compteur quelconque à F_0/s_2 [Carter et Wegman, 1979]. Nous pouvons ainsi proposer une estimation plus fiable des éléments non surabondants. L'algorithme 3.8 présente le pseudo-code de notre approche.

Nous avons prouvé que, pour tout paramètre de précision et sensibilité ε et δ, pour tout seuil d'éléments surabondants $\varepsilon^3/4 \leq \beta < 1$, CASE est une (ε, δ)-approximation relative dans la majorité des cas :

$$\mathbb{P}\left\{|\hat{f}_u - f_u| \geq \varepsilon f_u\right\} \leq \delta \quad \text{si } \forall v \in N, f_v \leq \frac{\varepsilon^2 m}{2s_1} \tag{3.16}$$

ou

$$\begin{cases} \mathbb{P}\left\{|\hat{f}_u - f_u| \geq \varepsilon f_u\right\} \leq \delta & \text{si } f_u \geq \beta m \\ \mathbb{P}\left\{|\hat{f}_u - f_u| \geq \varepsilon (m - f_u)\right\} \leq \delta & \text{sinon} \end{cases} \tag{3.17}$$

utilisant $O\left((1/\beta + (\log(1/\delta)/\varepsilon)(1/\beta + 1/\varepsilon))\,(\log m + \log n)\right)$ bits de mémoire. La distinction des deux cas permet d'identifier les flux dans lesquels certains éléments sont extrêmement rares (quelques occurrences seulement). Pour ces éléments particuliers, l'obtention d'une estimation relative est impossible sans utiliser une quantité mémoire de l'ordre la taille du flux (et/ou de l'univers Ω) [Anceaume *et al.*, 2015b].

Enfin, nous avons étendu CASE dans le cadre du modèle à flux répartis, qui offre une (ε, δ)-approximation de la fréquence globale de tout élément sur l'union des flux entrants, avec des bornes identiques à la version à flux unique. Tirant parti de la propriété d'agrégat linéaire de la structure de données utilisée, les nœuds du système $\mathcal{S}$ sont répartis dans une structure auto-organisante célèbre : les tables de hachage distribuées (DHT), qui ont reçu un intérêt croissant

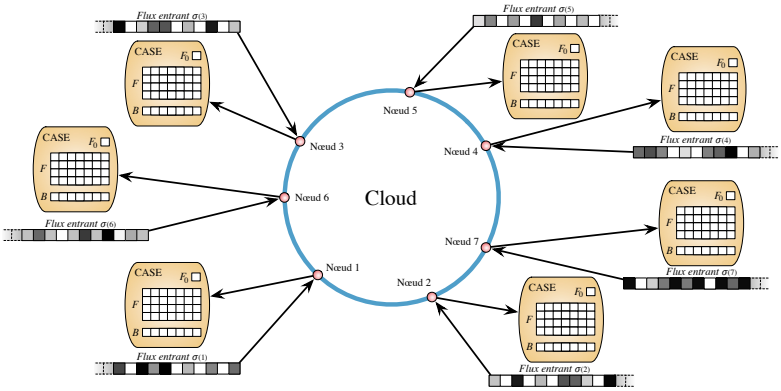


FIGURE 3.7 – Structuration répartie, où chaque nœud exécute une instance de CASE sur une DHT.

Algorithme 3.8 : CASE (Count-Any Sketch Estimator)

Entrées : Un flux de données σ ; Paramètres de précision ε et β, et de sensibilité δ;

Sortie : L'estimation de la fréquence d'occurrence $\hat{f}_u$ de n'importe quel élément u lu sur le flux;

$s_1 \leftarrow \lceil \log(1/\delta) \rceil$;

$s_2 \leftarrow \lceil 2(1-\beta)/\beta\varepsilon \rceil$;

$m = 0$;

$\hat{F}[1..s_1][1..s_2] \leftarrow 0_{s_1,s_2}$;

choisir s_1 fonctions de hachage $h_1, \ldots, h_{s_1} : \Omega \to [\![s_2]\!]$ parmi une famille 2-universelle;

pour tous les $u \in \sigma$ **faire**

  $m \leftarrow m + 1$;

  Tâche T_1 *(Mise à jour de l'agrégat principal)* **:**

    pour $i = j$ **à** s_1 **faire**

      $\hat{F}[j][h_j(u)] \leftarrow \hat{F}[j][h_j(u)] + 1$;

  Tâche T_2 *(Estimation des fréquences des éléments surabondants)* **:**

    $\hat{B} \leftarrow$ Algorithme proposé par Metwally *et al.* [2005] appliqué à u, avec des paramètres de précision ε et $1/\beta$;

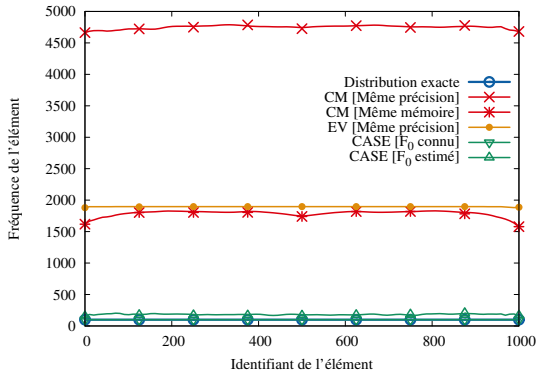
  Tâche T_3 *(Estimation de F_0)* **:**

    $\hat{F}_0 \leftarrow$ Algorithme proposé par Kane *et al.* [2010] appliqué à u;

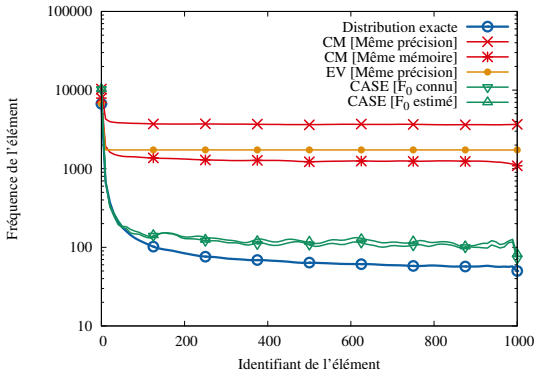
à la requête $\hat{f}_u$ $\hat{f}_u \leftarrow \min_{1 \le j \le s_1} \left(\hat{F}[j][h_j(u)] \right)$;

si $u \in \hat{B}$ **et** $\hat{f}_u \geq m\varepsilon$ **alors retourner** $\hat{f}_u$;

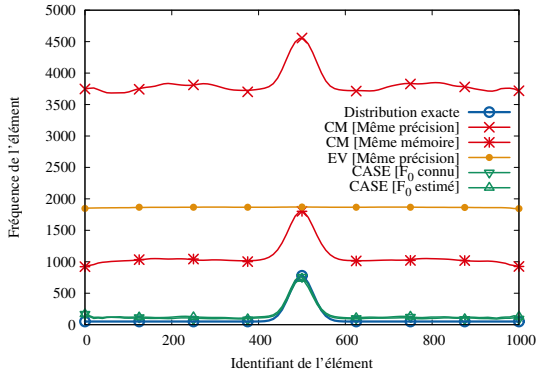
sinon retourner $\hat{f}_u \cdot s_2/\hat{F}_0$;



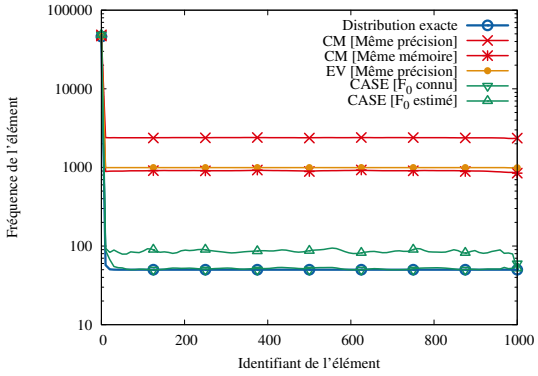
(a) Distribution uniforme.



(b) Distribution de Zipf avec $\alpha = 1$.



(c) Distribution de Poisson tronquée.



(d) Distribution de Zipf avec $\alpha = 4$.

FIGURE 3.8 – Fréquence estimée de tous les éléments de l'univers Ω, avec $\varepsilon = 0,1$ et $\delta = 0,01$.

depuis le début des années 2000 [Druschel et Rowstron, 2001, Ratnasamy *et al.*, 2001, Rowstron et Druschel, 2001, Stoica *et al.*, 2003, Zhao *et al.*, 2004]. Chaque nœud est alors responsable de l'estimation de fréquence de tous les éléments dont la « clé » est la plus proche de leur identifiant unique, généré selon l'espace de nommage de la DHT (*cf.* figure 3.7).

Nous avons également évalué CASE sur des flux de données générés à partir de différentes distributions, de taille $m = 100\,000$, et de nombre d'éléments distincts $n = 1\,000$. Un aperçu de ces résultats est présenté en figure 3.8. Nous dénotons ici :

CM [Même précision] Résultats obtenus en utilisant le CM Sketch initialisé avec les mêmes valeurs ε et δ que CASE.

CM [Même mémoire] Résultats obtenus en utilisant le CM Sketch initialisé avec des valeurs ε et δ plus petites que celles de CASE, permettant d'atteindre la même empreinte mémoire que ce dernier.

EV [Même précision] Résultats obtenus en utilisant l'algorithme de Estand et Varghese [2003] avec les mêmes valeurs ε et δ que CASE.

CASE [F_0 connu] Résultats obtenus en utilisant CASE, lequel connait précisément F_0 (*i.e.*, sans la tâche T_3).

CASE [F_0 estimé] Résultats obtenus en utilisant CASE, avec l'estimation de F_0 par l'algorithme de Kane *et al.* [2010].

D'un point de vue global, CASE réalise de meilleures performances que les autres algorithmes existants. Dans le cas de flux uniformément distribués (*cf.* figure 3.8a) ou de loi de Poisson (*cf.* figure 3.8c), CASE estime parfaitement la fréquence des éléments contrairement au CM Sketch. En effet, dans ces deux scénarii, tous les éléments (ou la majorité dans le cas de Poisson) possèdent une fréquence identique, permettant d'exhiber les performances optimales de CASE (*cf.* relation 3.16). Il est à noter que fournir plus d'espace au CM Sketch ne permet pas d'atteindre la même précision que CASE. Concernant les distributions (fortement) biaisées (*cf.* figures 3.8b et 3.8d), CASE surestime légèrement les fréquences des éléments rares (par un facteur $1,2$), alors que CM Sketch surestime par un facteur 100 dans le meilleurs des cas. Ceci s'explique par la relation 3.17, *i.e.,* le passage en approximation additive en raison de la distribution d'entrée. Cependant, comme il faut s'y attendre, tous les algorithmes estiment parfaitement les éléments surabondants.

Des expérimentations plus poussées, notamment sur des jeux de données réelles et sur l'impact des paramètres dans la précision de l'estimation, sont disponibles dans [Anceaume *et al.*, 2015b].

3.3.2 Extension dans le modèle de fenêtre glissante

Nous nous sommes également intéressés à l'extension du CM Sketch de Cormode et Muthukrishnan [2005] dans le modèle à fenêtre glissante. Dans ce contexte, la difficulté principale est de pouvoir retirer de l'agrégat les contributions liées à des éléments qui ne sont plus valides (*i.e.,* sont sortis de la fenêtre glissante, *cf.* Section 2.1.2).

Nous avons ainsi proposé deux extensions : les algorithmes Proportionnel et Diviseur [Rivetti *et al.*, 2015a]. Dans la suite de cette section, la notation f_u représente désormais la fréquence de l'élément u dans la fenêtre courante et non la fréquence totale sur le flux σ. Néanmoins, nous proposons tout d'abord deux autres algorithme naïfs, qui jouissent des bornes optimales par rapport à la précision (algorithme Parfait) ou à l'empreinte mémoire (algorithme Simple).

Algorithme Parfait Cet algorithme assure la meilleure précision en relâchant la contrainte de complexité mémoire faible. Il stocke ainsi intégralement la fenêtre active en mémoire. Lorsqu'un élément est reçu, il est placé dans la file et ses compteurs, dans l'agrégat, sont incrémentés. Une fois que la file mémoire atteint la taille $\ddot{m}$, l'élément expiré est retiré de celle-ci et ses compteurs sont décrémentés.

Algorithme Simple À l'inverse, cet algorithme est le plus direct qui soit, tout en conservant la meilleure empreinte mémoire, par rapport à l'algorithme pionnier. Il se comporte comme le CM Sketch, excepté pour la matrice $\hat{F}$ qui est réinitialisée au début de chaque fenêtre. Logiquement,

Algorithme 3.9 : PROPORTIONNEL pour l'estimation de fréquences en fenêtre glissante

Entrées : Un flux de données σ ; Deux paramètres de précision ε et de sensibilité δ;
Sortie : L'estimation de la fréquence d'occurrence $\hat{f}_u$ de n'importe quel élément u lu sur le flux;

$s_1 \leftarrow \lceil \log(1/\delta) \rceil$; $s_2 \leftarrow \lceil e/\varepsilon \rceil$; $m' = 0$;
$\hat{F}[1..s_1][1..s_2] \leftarrow 0_{s_1,s_2}$;
$\hat{I}[1..s_1][1..s_2] \leftarrow 0_{s_1,s_2}$;
choisir s_1 fonctions de hachage $h_1, \ldots, h_{s_1} : \Omega \to [\![s_2]\!]$ parmi une famille 2-universelle;
pour tous les $u \in \sigma$ **faire**
    si $m' = 0$ **alors**
        pour tous les $i_1 = 1$ **à** s_1 **et** $i_2 = 1$ **à** s_2 **faire** $\hat{I}[i_1, i_2] \leftarrow \hat{F}[i_1, i_2]/\ddot{m}$;
    pour tous les $i_1 = 1$ **à** s_1 **et** $i_2 = 1$ **à** s_2 **faire**
        si $h_{i_1}(u) = i_2$ **alors** $\hat{F}[i_1, i_2] \leftarrow \hat{F}[i_1, i_2] + 1$;
        $\hat{F}[i_1, i_2] \leftarrow \hat{F}[i_1, i_2] - \hat{I}[i_1, i_2]$;
    $m' \leftarrow m' + 1 \mod \ddot{m}$;
à la requête $\hat{f}_u$ **retourner** $\lfloor \min_{i \in [\![s_1]\!]} \{\hat{F}[i][h_i(u)]\} \rfloor$

la précision de cet algorithme est sommaire, étant donné qu'il détruit toute information passée à chaque nouvelle fenêtre.

Algorithme PROPORTIONNEL L'intuition derrière ce premier algorithme « intéressant » est la suivante. À la fin de chaque fenêtre, la matrice $\hat{F}$ est recopiée dans une matrice de même dimension $\hat{I}$, permettant de mémoriser la distribution agrégée de la fenêtre précédente :

$$\forall i_1, i_2 \in [\![s_1]\!] \times [\![s_2]\!],\ \hat{I}[i_1, i_2] \leftarrow \hat{F}[i_1, i_2]/\ddot{m}.$$

À chaque nouvel élément reçu, les compteurs associés à cet élément sont toujours incrémentés, mais l'ensemble de la matrice $\hat{F}$ courante est réduite proportionnellement à l'instantané mémorisé :

$$\forall i_1, i_2 \in [\![s_1]\!] \times [\![s_2]\!],\ \hat{F}[i_1, i_2] \leftarrow \hat{F}[i_1, i_2] - \hat{I}[i_1, i_2]$$

Cela permet de lisser l'impact du réajustement sur l'ensemble de la fenêtre courante. L'algorithme 3.9 présente le pseudo-code de cette approche.

Algorithme DIVISEUR Le précédent algorithme retranche donc à la fenêtre en cours la distribution de fréquences « moyennes » sur la fenêtre précédente. Par conséquent, il ne capture pas les changements brusques et soudains internes à une fenêtre. Pour faire face à ce défaut, il serait possible de suivre ces changements critiques via plusieurs instantanés. Cependant, chaque projection vectorielle étant associée à une fonction de hachage 2-universelle, ces variations n'affectent pas chaque ligne de $\hat{F}$ de la même manière.

Ainsi, l'algorithme DIVISEUR propose une approche à granularité plus fine en analysant le taux

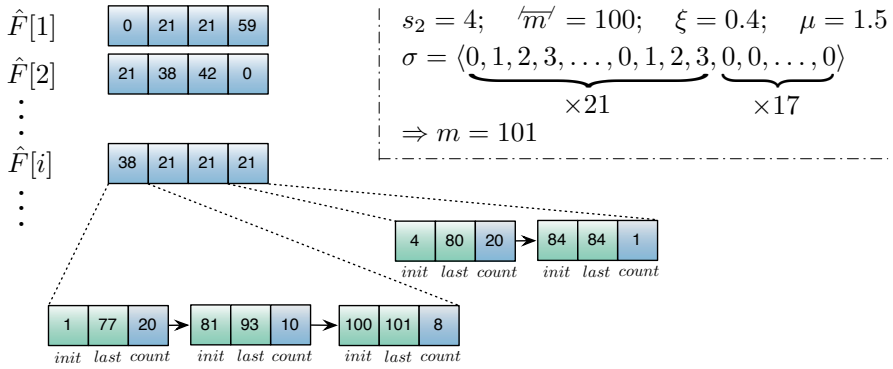


FIGURE 3.9 – Structure de données du DIVISEUR

de mises à jour de chaque cellule de $\hat{F}$. Nous ajoutons une file de sous-cellules à chaque entrée de $\hat{F}$. Lorsqu'une variation importante est détectée, une nouvelle sous-cellule est créée et ajoutée à la file. Cette dernière s'attache donc à suivre le taux de mises à jour actuel de la cellule, alors que la précédente garde trace de l'ancien taux.

Chaque sous-cellule possède un compteur de fréquence (*count*) et deux horodatages : *init*, qui mémorise le temps (logique) où cette sous-cellule est devenue active et *last* qui stocke celui du dernier incrément. Après un court auto-amorçage, chaque cellule contient au moins deux sous-cellules : la courante qui représente l'historique récent et une prédécesseur retraçant le passé. La figure 3.9 illustre un possible état de la structure de données du DIVISEUR, après avoir reçu un préfixe de 101 éléments de σ. Chaque nouvelle sous-cellule est ainsi engendrée pour saisir un changement dans la distribution du flux. La décision de création d'une nouvelle sous-cellule est paramétrée via ξ, μ, et une fonction d'erreur : ERROR. De manière informelle, la fonction ERROR évalue la quantité d'information potentielle qui serait perdue en fusionnant deux sous-cellules, alors que μ représente la quantité raisonnable d'information pouvant être perdue. Effectuer ce test à chaque élément reçu pourrait donner lieu à des comportements erratiques. Le paramètre ξ ($0 < \xi \leq 1$) permet d'éviter cette situation, en fixant la durée de vie minimale d'une sous-cellule avant de l'intégrer dans le processus de décision. L'algorithme 3.10 présente le pseudo-code du DIVISEUR.

Nous avons prouvé que, étant donné $0 < \xi \leq 1$, le nombre maximum de sous-cellules existantes en parallèle est $O(\frac{1}{\varepsilon\xi} \log \frac{1}{\delta})$ (*i.e.*, pas plus de $1/\xi$ sous-cellules par entrée de $\hat{F}$). Nous avons aussi démontré que chacun des deux derniers algorithmes sont des (ε, δ)-approximations additives, PROPORTIONNEL ayant une meilleure complexité mémoire mais une précision plus faible que le DIVISEUR [Rivetti *et al.*, 2015a].

Nous avons comparé nos algorithmes avec la plus proche étude relative, les ECM Sketch, proposés par Papapetrou *et al.* [2012]. Ceux-ci consistent en une structure compacte combinant des méthodes d'agrégation fondées sur des histogrammes, avec un résumé de la fenêtre active. Le tableau 3.1 résume les complexités en mémoire, mise à jour et requête de tous les algorithmes

Algorithme 3.10 : DIVISEUR pour l'estimation de fréquences en fenêtre glissante

Entrées : Un flux de données σ ; Deux paramètres de précision ε et β, et de sensibilité δ ; Paramètres de scission/fusion des sous-cellules ξ et μ;

Sortie : L'estimation de la fréquence d'occurrence $\hat{f}_u$ de n'importe quel élément u lu sur le flux;

$s_1 \leftarrow \lceil \log(1/\delta) \rceil$;
$s_2 \leftarrow \lceil e/\varepsilon \rceil$;
$m' = 0$;
$\hat{F}[1..s_1][1..s_2] \leftarrow \overrightarrow{\langle \emptyset, 0 \rangle}$; ▷ L'ensemble est une file ◁
choisir s_1 fonctions de hachage $h_1, \ldots, h_{s_1} : \Omega \to [\![s_2]\!]$ parmi une famille 2-universelle;
pour tous les $u \in \sigma$ **faire**
  pour tous les $i_1 = 1$ **à** s_1 **et** $i_2 = 1$ **à** s_2 **faire**
    $\langle file, v \rangle \leftarrow \hat{F}[i_1, i_2]$;
    $first \leftarrow$ Tête de $file$; ▷ Récupération de la plus ancienne sous-cellule ◁
    si $\exists first \wedge first_{init} = m' - \ddot{m}$ **alors** ▷ La sous-cellule était active ◁
      $v' \leftarrow \frac{first_{count}}{first_{last} - first_{init} + 1}$;
      $v \leftarrow v - v'$;
      $first_{count} \leftarrow first_{count} - v'$;
      $first_{init} \leftarrow first_{init} + 1$;
      si $first_{init} > first_{last}$ **alors** Retirer $first$ de la file ;
    si $h_{i_1}(u) = i_2$ **alors** ▷ Traitement d'une cellule associée à l'élément u ◁
      $v \leftarrow v + 1$;
      $last \leftarrow$ Queue de $file$; ▷ Récupération de la sous-cellule la plus récente ◁
      si $\nexists last$ **alors** Créer et ajouter la nouvelle sous-cellule dans $file$;
      sinon si $last_{count} < \frac{\xi \ddot{m}}{s_2}$ **alors**
        Mettre à jour la sous-cellule $last$
      sinon
        $pred \leftarrow$ Prédécesseur de $last$ dans $file$;
        si $\exists pred \wedge$ ERROR$(pred, last) \leq \mu$ **alors**
          Fusionner $last$ avec $pred$ et remplacer $last$ dans $file$;
        sinon
          Créer et ajouter la nouvelle sous-cellule dans $file$; ▷ *Diviser* la cellule ◁
    $\hat{F}[i_1, i_2] \leftarrow \langle file, v \rangle$;
  $m' \leftarrow m' + 1$
à la requête $\hat{f}_u$ **faire**
  $\langle file, v \rangle \leftarrow \min_{i \in [\![s_1]\!]} \{\hat{F}[i][h_i(u)]\}$;
  retourner $\lfloor v \rfloor$;

Table 3.1 – Comparaison des complexités

Algorithme	Mémoire (bits)	Mise à jour	Requête
CM [Cormode et Muthukrishnan, 2005]	$O(\frac{1}{\varepsilon}\log\frac{1}{\delta}(\log m+\log n))$	$O(\log\frac{1}{\delta})$	$O(\log\frac{1}{\delta})$
Parfait	$O(\ddot{m})$	$O(\log\frac{1}{\delta})$	$O(\log\frac{1}{\delta})$
Simple	$O(\frac{1}{\varepsilon}\log\frac{1}{\delta}(\log\ddot{m}+\log n))$	$O(\log\frac{1}{\delta})$	$O(\log\frac{1}{\delta})$
Proportionnel	$O(\frac{1}{\varepsilon}\log\frac{1}{\delta}(\log\ddot{m}+\log n))$	$O(\frac{1}{\varepsilon}\log\frac{1}{\delta})$	$O(\log\frac{1}{\delta})$
Diviseur	$O(\frac{1}{\xi\varepsilon}\log\frac{1}{\delta}(\log\ddot{m}+\log n))$	$O(\log\frac{1}{\delta})$	$O(\log\frac{1}{\delta})$
ECM [Papapetrou *et al.*, 2012]	$O\left(\frac{1}{\varepsilon^2}\log\frac{1}{\delta}\left(\log^2\varepsilon\ddot{m}+\log n\right)\right)$	$O(\log\frac{1}{\delta})$	$O(\log\frac{1}{\delta})$

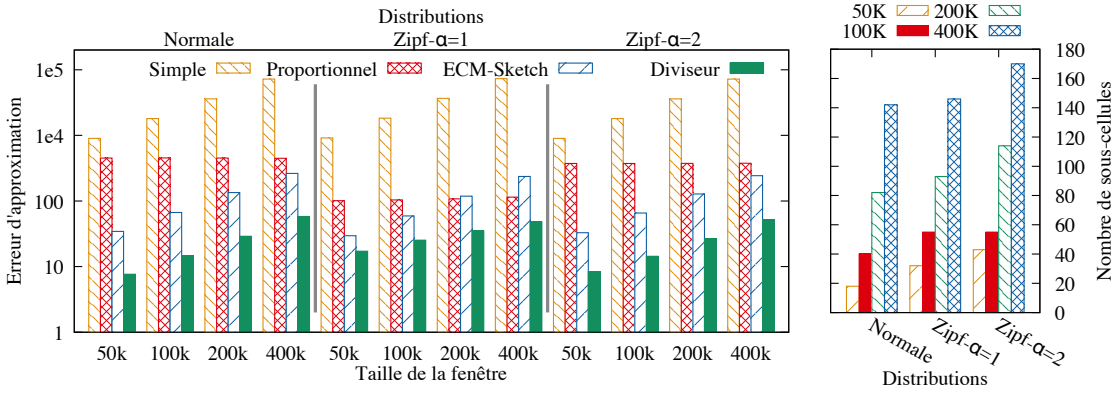
sus-cités.

Nous avons ensuite conduit une série d'expérimentations avec différents flux et paramètres, afin de comparer l'efficacité respective des algorithmes présentés et de les comparer avec les ECM Sketch. La métrique utilisée dans nos évaluations, dénommée *erreur d'approximation*, correspond à la moyenne, sur tous les éléments de Ω, des erreurs absolues des estimations de l'algorithme considéré par rapport à celles retournées par Parfait, *i.e.,*

$$\frac{1}{n}\left(\sum_{u\in\Omega}\left|\hat{f}_u^{(\text{Algorithme\_Évalué})}-\hat{f}_u^{(\text{Parfait})}\right|\right).$$

La figure 3.10a présente les erreurs d'approximation, sur différentes distributions de flux de taille $m = 150\,000 = 3\ddot{m}$, contenant $n = 1\,000$ éléments distincts. Il apparait clairement que Simple retourne toujours des résultats pires que, dans l'ordre, Proportionnel, ECM et Diviseur. Il est à noter que, en moyenne sur ces expérimentations, l'erreur de Diviseur est 4 fois plus petite que celle d'ECM, tout en utilisant 4 fois moins de mémoire. En parallèle, la figure 3.10b illustre le nombre moyen de sous-cellules générées par Diviseur pour conserver les changements de distribution. Ce nombre de divisions croît en moyenne d'un facteur de $1,7$ pour chaque doublement de $\ddot{m}$. En effet, le ratio ξ étant fixé, la durée minimale de chaque sous-cellule augmente avec $\ddot{m}$, ainsi que l'erreur par conséquent.

La figure 3.11 présente l'erreur d'approximation de chaque algorithme sur un flux variant drastiquement périodiquement. Ici, les probabilités d'occurrences des éléments sont permutées tous les $15\,000$ éléments, et la distribution change tous les $60\,000$ éléments, comme indiqué au dessus de la courbe. Afin d'éviter les effets de bord, les périodes de permutations et de changements de distribution ne sont pas synchronisées avec la taille de la fenêtre ($\ddot{m} = 50\,000$). À nouveau, la précision de Diviseur surpasse toutes celles des autres algorithmes. L'aspect périodique de Simple est clairement observable : l'erreur explose en début de fenêtre (correspondant à la réinitialisation de $\hat{F}$) et diminue continuellement jusqu'à la fin. Dans les $1^{\text{ère}}$ et $6^{\text{ème}}$ fenêtres (respectivement de



(a) Erreur d'approximation moyenne. **(b)** Nombre de sous-cellules.

FIGURE 3.10 – Erreur d'approximation avec plusieurs distributions consécutives.

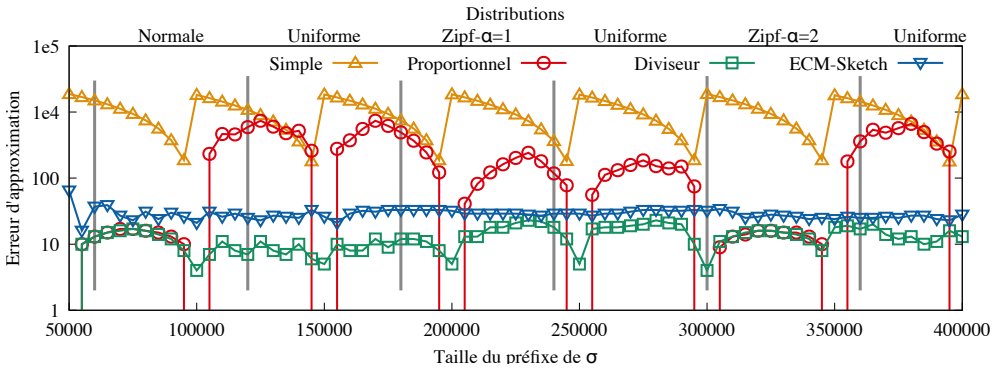


FIGURE 3.11 – Erreur d'approximation avec plusieurs distributions consécutives.

0 à 50 000, puis de 250 000 à 300 000), la distribution ne change pas (permuter les probabilités d'une distribution uniforme n'a pas d'effet). C'est la raison pour laquelle DIVISEUR ne capture pas plus d'information que PROPORTIONNEL, et ainsi, ils produisent les mêmes estimations sur les $2^{\text{ème}}$ et $7^{\text{ème}}$ fenêtres.

L'étude expérimentale complète de ces algorithmes, disponible dans [Rivetti *et al.*, 2015a], renseigne sur l'efficacité et la robustesse de nos algorithmes, à la fois sur des traces réelles et d'autres traces synthétiques conçues pour exhiber les comportements aux limites. Les erreurs d'estimation sont toujours plus faibles que celles produites par les solutions de l'état de l'art et l'empreinte mémoire est au moins quatre fois moindre.

Enfin, nous avons également proposé deux extensions de nos algorithmes, respectivement pour les modèles à flux répartis et pour les fenêtres temporelles [Rivetti *et al.*, 2015a]. L'intégration de ces techniques de fenêtrage dans CASE (*cf.* section 3.3.1) fait par ailleurs l'objet de travaux en cours.

3.4 Échantillonnage

Nous nous sommes intéressés au problème de l'échantillonnage uniforme, dans les modèles de flux répartis avec adversaire. L'échantillonnage uniforme est une primitive fondamentale qui garantit que tout individu d'une population a la même probabilité d'être sélectionné dans l'échantillon. Cette propriété est particulièrement importante dans les systèmes où la population évolue continuellement et où il est impossible de saisir globalement la complexité du système. En recueillant un sous-ensemble aléatoire d'informations sur le système, il devient possible d'inférer des caractéristiques globales à moindre coût sur l'ensemble de la population (*e.g.,* sa taille, son organisation topologique, ses ressources, *etc.*). Ainsi, l'échantillonnage uniforme est essentiel dans de nombreux problèmes tels que la collecte de données, la dissémination, l'équilibrage de charge, la métrologie ou les caches répartis [Bertier *et al.*, 2009, Jelasity *et al.*, 2007, Karger et Ruhl, 2004, Lv *et al.*, 2002]. Fournir un échantillonnage non biaisé (*i.e.,* uniforme dans notre cas) dans les systèmes à large échelle est particulièrement complexe [Busnel *et al.*, 2011a].

Plusieurs approches ont été proposées pour résoudre le problème de l'échantillonnage uniforme en présence d'adversaire dans les très grands systèmes [Jesi *et al.*, 2010, Liu *et al.*, 2005, Singh *et al.*, 2006]. Ces techniques proposées pour détecter et exclure les éléments malveillants reposent sur l'observation que les malveillants cherchent à être sur-représentés dans les échantillons, pour éclipser les éléments corrects. Par exemple, tirant partie des propriétés des DHT et de leurs structures inhérentes, Anceaume *et al.* [2011b], Awerbuch et Scheideler [2004, 2007] ont montré qu'avec une forte probabilité, tout nœud possédait la même probabilité d'apparition dans la vue du système de tout nœud correct, après un nombre de rondes polynomial en la taille du système.

Dans ce contexte, nous avons tout d'abord proposé une définition formelle du problème d'échantillonnage uniforme tolérant aux attaques, par l'introduction de deux propriétés [Anceaume *et al.*, 2013a] :

Uniformité Tout élément de la population doit posséder la même probabilité d'occurrence dans l'échantillon d'un nœud correct du système.

Fraîcheur Tout élément qui apparaît une infinité de fois dans le flux doit posséder une probabilité non nulle d'apparaître infiniment souvent dans l'échantillon d'un nœud correct du système.

Notre première étude a été dédiée à l'analyse formelle des conditions pour lesquelles un échantillonnage uniforme et frais était réalisable de manière déterministe [Anceaume *et al.*, 2010]. Plus précisément, nous avons établi les conditions nécessaires et suffisantes à la mise en œuvre de ce service en présence d'une large proportion de nœuds byzantins, dénotée $\kappa < 1$. Soient δ_m le nombre d'éléments malveillants (non nécessairement uniques) reçus sur le flux durant un intervalle de temps logique Ξ et Γ la mémoire locale d'un nœud correct $j \in \mathcal{S}$. Dans ce contexte, nous avons démontré les assertions suivantes :

- Si le nombre δ_m d'éléments malveillants (non uniques) reçus par i durant la période Ξ donnée est strictement plus grand que $\Xi - |\Omega|(1-\kappa)$ alors, aucun échantillonnage uniforme et/ou frais ne peut être réalisé.
- Si $\delta_m \leq \Xi - |\Omega|(1-\kappa)$ et que la taille de la mémoire Γ est supérieure ou égale à $|\Omega|$ alors, un échantillonnage uniforme et frais est réalisable [1].
- Si $\delta_m \leq \Xi - |\Omega|(1-\kappa)$, et $|\Gamma| < |\Omega|$ alors, un échantillonnage uniforme et frais n'est pas réalisable.

En résumé, ces conditions montrent que si le système ne peut prétendre à limiter le nombre d'éléments inoculés par un adversaire dans le flux, alors réaliser un échantillonnage uniforme ou frais est impossible. D'autre part, si cette condition est vérifiée et que tous les nœuds corrects de $\mathcal{S}$ ont accès à une immense mémoire (de la taille de l'univers) alors le problème devient trivialement résoluble. Malheureusement, comme nous l'avons montré dans [Anceaume *et al.*, 2010], ces conditions sont nécessaires et suffisantes pour résoudre le problème de l'échantillonnage uniforme et frais. Ces conditions excessives illustrent clairement les dommages que peut générer un comportement malveillant dans un système à grande échelle [Anceaume *et al.*, 2013a].

Nous avons ensuite conduit une étude déterminant une caractérisation de la puissance d'un adversaire visant à biaiser un échantillonnage uniforme et frais [Anceaume *et al.*, 2011]. En adoptant un point de vue statistique des flux d'entrée et en comparant leurs distributions par des métriques de divergences (*cf.* section 3.1), nous avons dérivé des bornes inférieures de l'effort nécessaire à exercer par l'adversaire pour biaiser suffisamment le flux d'entrée entraînant l'impossibilité de réaliser cet échantillonnage. Nous avons ainsi étudié deux modèles d'adversaire : *(i) omniscient* ayant la capacité d'observer tous les flux et messages échangés sur le système et *(ii) aveugle* ne pouvant écouter que les échanges impliquant au moins un nœud malveillant.

Enfin, nous avons franchi une étape supplémentaire en proposant un algorithme d'échantillonnage probabiliste, permettant de garantir avec une erreur bornée les conditions d'uniformité et de fraicheur [Anceaume *et al.*, 2013b].

Dans un premier temps, à l'image de la méthode proposée en section 3.2, nous supposons que notre algorithme dispose d'un oracle lui indiquant exactement la probabilité d'occurrence de tout élément reçu sur le flux. Cette solution dite « *omnisciente* », n'a cependant accès qu'à une petite mémoire Γ de taille constante. L'intuition de cet algorithme repose sur la sélection d'un élément pour apparaître dans l'échantillon avec une probabilité inversement proportionnelle à sa fréquence d'occurrence. Nous avons analysé le comportement de cet algorithme par l'analyse de chaînes de Markov, et nous avons montré que celui-ci est capable de tolérer tout type de biais sur le flux d'entrée, introduit par l'adversaire. L'étude des comportements transitoires et stationnaires de notre algorithme montre que ce dernier résout le problème d'échantillonnage uniforme et frais.

1. Il est à noter que ce cas de figure est peu intéressant en pratique, l'empreinte mémoire qui permet de mémoriser tout l'univers étant irréalisable (*cf.* section 2.1).

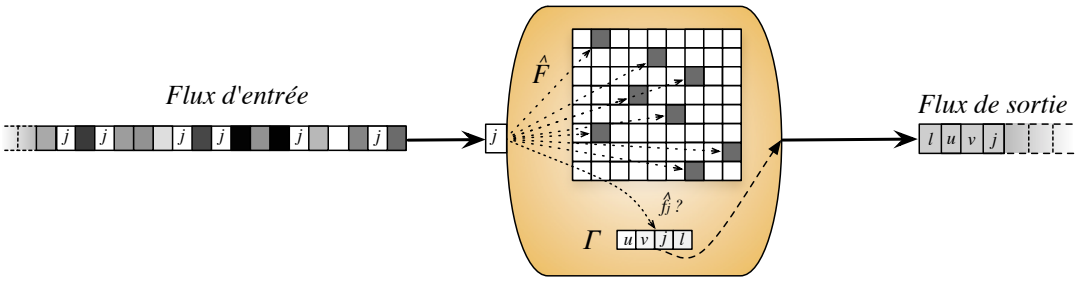


FIGURE 3.12 – Structure de données d'un nœud $i \in \mathcal{S}$.

Dans un second temps, nous avons proposé une adaptation « non omnisciente » qui retourne un flux quasi-uniforme en sortie. La déviation de ce dernier par rapport à l'uniforme est bornée par n'importe quelle probabilité définie par l'utilisateur. Comme pour toutes les solutions proposées, celle-ci ne repose pas sur une connaissance *a priori* des caractéristiques du flux. L'algorithme 3.11 présente le pseudo-code de notre solution. Nous avons notamment montré qu'avec une empreinte mémoire de $O(\log(n)\log(1/\delta)/\varepsilon)$ bits, le flux de sortie est une (ε, δ)-approximation additive d'un flux uniforme et frais à partir de n'importe quel flux d'entrée arbitrairement biaisé. La figure 3.12 illustre la structure de données et le fonctionnement interne d'un nœud exécutant cet algorithme.

Nous avons également évalué l'effort minimum requis par un adversaire adaptatif pour biaiser le flux de sortie, selon deux attaques représentatives. La première, dénommée *attaque ciblée*, consiste pour l'adversaire à biaiser l'estimation de la fréquence d'un élément donné, alors que la seconde, *attaque par inondation*, a pour objectif de biaiser toutes les fréquences estimées. Guidées par une évaluation modélisant un problème d'urne, les conclusions montrent que l'effort nécessaire d'un adversaire pour subvertir le système peut être rendu arbitrairement grand en augmentant la taille mémoire dédiée à l'échantillonneur.

Nous avons par la suite mené des expérimentations avec des traces réelles ainsi que des flux de données générés synthétiquement pour éprouver la robustesse de notre algorithme. La figure 3.13 présente une courbe de niveau pour laquelle l'axe horizontal représente le temps logique (*i.e.*, le nombre d'éléments reçus sur le flux d'entrée) et l'axe vertical représente les n éléments de l'univers Ω. La couleur de chaque point représente la fréquence d'occurrences de chaque élément reçu jusqu'alors. Plus cette couleur est chaude, plus l'élément est fréquent.

Du haut en bas, cette figure représente donc l'évolution du vecteur de fréquences pour, respectivement, le flux d'entrée, le flux de sortie de l'algorithme 3.11 et le flux de sortie de la version omnisciente. Au démarrage de ces flux, un faible nombre d'éléments ont été générés, expliquant les couleurs sombres des parties gauches. Avec le temps, le nombre d'éléments augmente (jusqu'à $m = 40\,000$) et le biais du flux d'entrée apparaît nettement, celui-ci étant généré par une loi de Poisson tronquée : quelques éléments reviennent à haute fréquence (jusqu'à 400 pour certains), alors que la fréquence des autres éléments reste significativement basse. Comme attendu,

Algorithme 3.11 : Algorithme d'échantillonnage sur un nœud correct $i \in \mathcal{S}$

Entrées : Un flux de données $\sigma^{(i)}$; Paramètres de précision ε, de sensibilité δ et de taille maximale du tampon c;

Sortie : Un flux de données (ε, δ)-uniforme $\sigma_o^{(i)}$;

$s_1 \leftarrow \lceil \log(2/\delta) \rceil$;
$s_2 \leftarrow \lceil e/\varepsilon \rceil$;
$\hat{F}[1..s_1][1..s_2] \leftarrow 0_{s_1,s_2}$;
$\Gamma_i \leftarrow \emptyset$;
choisir s_1 fonctions de hachage $h_1, \ldots, h_{s_1} : \Omega \to [\![s_2]\!]$ parmi une famille 2-universelle;
pour tous les $u \in \sigma^{(i)}$ **faire**
    Tâche T_1 *(Estimation des fréquences par le Count-Min Sketch)* **:**
        Algorithme 3.6 proposé par Cormode et Muthukrishnan [2005] appliqué à u;
    Tâche T_2 *(Echantillonnage en mémoire tampon* Γ_i*)* **:**
        $\hat{q} \leftarrow \min_{1 \le j_1 \le s_1} \min_{1 \le j_2 \le s_2} \hat{F}[j_1][j_2]$;
        si $|\Gamma_i| < c$ **alors**
            $\Gamma_i \leftarrow \Gamma_i \cup \{u\}$;
        sinon
            avec une probabilité $a_j = \hat{q}/\hat{f}_u$ **faire**
                choisir un élément ℓ parmi Γ_i avec une probabilité $1/c$;
                $\Gamma_i \leftarrow (\Gamma_i \setminus \{\ell\}) \cup \{u\}$;
        choisir un élément ℓ' parmi Γ_i avec une probabilité $1/c$;
        écrire ℓ' sur le flux de sortie $\sigma_o^{(i)}$;

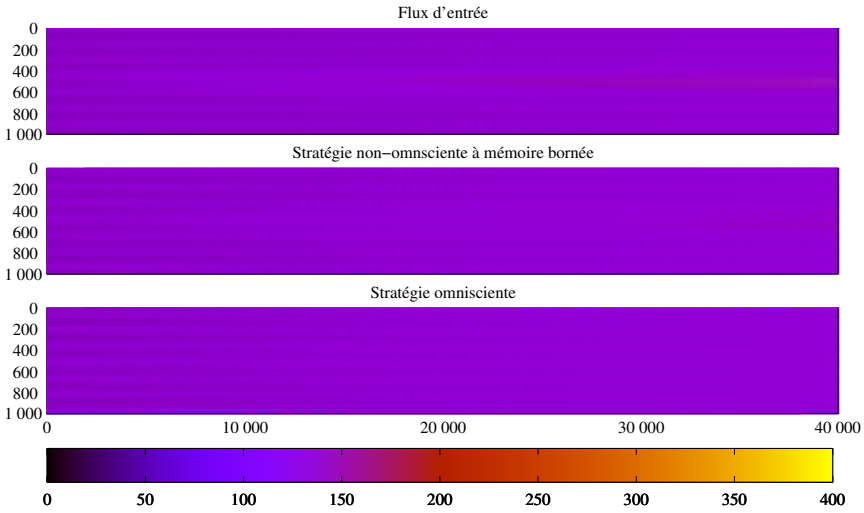


Figure 3.13 – Distribution de fréquence en fonction du temps logique.
Paramètres : $n = 1\,000$, $c = 15$, $s_1 = 10$ et $s_2 = 15$.

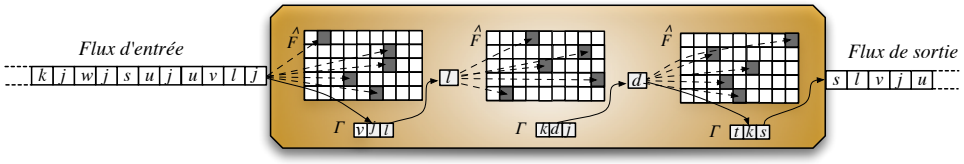
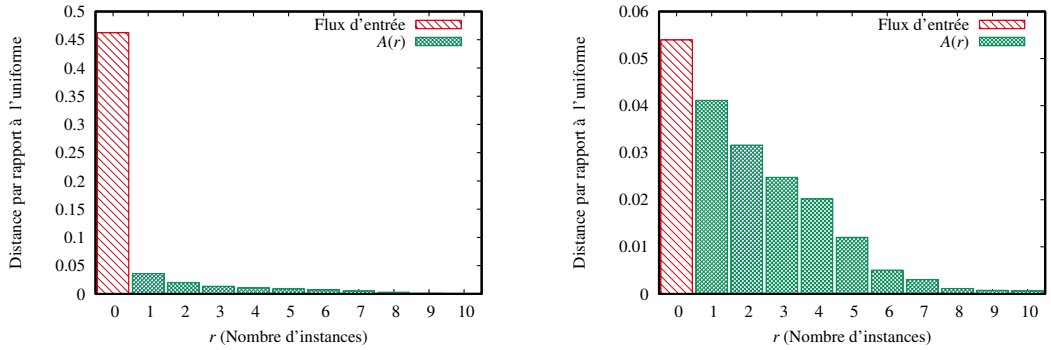


Figure 3.14 – Structure de données de l'algorithme $\mathcal{A}(3)$ sur un nœud $i \in \mathcal{S}$.



(a) Loi de Zipf avec $\alpha = 4$.

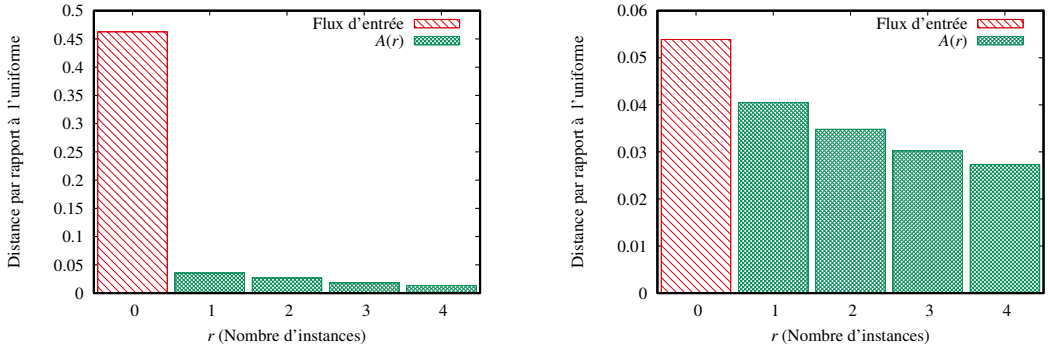
(b) Loi de Poisson tronquée avec $\lambda = n/2$.

Figure 3.15 – Distance entre le flux de sortie généré par $\mathcal{A}(r)$ avec un flux uniforme, en fonction du nombre d'instances en série r de l'algorithme 3.11, **à mémoire proportionnelle**.
Paramètres : $m = 100\,000$, $n = 1\,000$, empreinte mémoire totale de $\mathcal{A}(r) = r(10 \times 15 + 15) \log n$.

la version omnisciente réussit à produire un flux de sortie uniforme, illustré par une coloration verticale homogène. L'algorithme 3.11 n'atteint pas les mêmes performances que la version omnisciente, cependant, la fréquence des éléments centraux est drastiquement réduite, avec une empreinte mémoire très faible (*cf.* légende de la figure 3.13).

Enfin, nous avons proposé une construction originale, dénotée $\mathcal{A}(r)$, qui traite le flux en utilisant plusieurs (r) échantillonneurs en cascade. Nous montrons théoriquement que ce procédé permet d'améliorer la fiabilité d'un unique échantillonneur tout en réduisant le temps de convergence (*i.e.,* établissement du régime stationnaire), et ce, sans nécessiter plus d'espace mémoire. En effet, la mémoire disponible pour la version utilisant un unique échantillonneur est divisée équitablement entre les r échantillonneurs de $\mathcal{A}$. La figure 3.14 illustre la structure de données d'un nœud $i \in \mathcal{S}$ utilisant ce procédé.

Les figures 3.15 et 3.16 illustrent la distance euclidienne entre le flux de sortie de $\mathcal{A}(r)$ et un flux uniforme, en fonction du nombre d'instances r en cascade, à mémoire respectivement proportionnelle et constante (chaque instance de $\mathcal{A}(r)$ utilise exactement $(s_1 s_2 + c) \log(n)$ bits de mémoire dans le premier cas, et $(s_1 s_2 + c) \log(n)/r$ bits dans l'autre). La valeur $r = 0$ représente ainsi un échantillonneur sans effet, soit le flux d'entrée original. Formellement, la distance



(a) Loi de Zipf avec $\alpha = 4$.

(b) Loi de Poisson tronquée avec $\lambda = n/2$.

FIGURE 3.16 – Distance entre le flux de sortie généré par $\mathcal{A}(r)$ avec un flux uniforme, en fonction du nombre d'instances en série r de l'algorithme 3.11, **à mémoire contante**. *Paramètres* : $m = 100\,000$, $n = 1\,000$, empreinte mémoire totale de $\mathcal{A}(r) = (10 \times 15 + 15) \log n$.

euclidienne entre deux vecteurs $\mathbf{f}$ et $\mathbf{f}'$ est définie par

$$\mathcal{D}_{EU}(\mathbf{f}, \mathbf{f}') = \sqrt{\sum_{u \in \Omega} (f_u - f'_u)^2}. \tag{3.18}$$

Lorsque le flux d'entrée est fortement biaisé (*cf.* figure 3.15a), l'impact d'une unique instance est prédominante sur les suivantes. Cependant, pour des distributions moins extrêmes, l'amélioration est à peu près proportionnelle à r (*cf.* figure 3.15b). La lecture de la figure 3.16 nous informe que la distance à l'uniforme du flux de sortie généré par $\mathcal{A}(r)$ décroît également progressivement en fonction de r. En d'autres termes, utiliser une série de petits agrégats est plus efficace que l'utilisation d'un agrégat unique plus grand pour produire un flux uniforme. De plus, la comparaison entre les figures 3.15 et 3.16 montre une faible différence. Par exemple, l'utilisation de 4 agrégats, chacun utilisant $(s_1 s_2 + c) \log(n)/r$ bits (*cf.* figure 3.16a) produit un meilleur résultat que l'utilisation de 3 agrégats, chacun munis de $(s_1 s_2 + c) \log n$ bits (*cf.* figure 3.15a). La même tendance est observable pour les flux d'entrée moins fortement biaisés (*cf.* figures 3.15b et 3.16b). Ceci s'explique par le fait que nous avons réduit les dimensions de c et s_1 (mais pas s_2). Ainsi, chaque élément $u \in \Omega$ est plongé uniformément au hasard dans l'intervalle $[\![s_2]\!]$ que ce soit dans les « petits » ou les « grands » agrégats. D'un autre côté, s_1 représentant le nombre de partitions de Ω vers $[\![s_2]\!]$ est réduit. Cela souligne clairement l'impact prédominant de s_2 sur s_1 dans le CM Sketch [Cormode et Muthukrishnan, 2005].

L'ensemble de ces résultats est proposé pour le modèle à flux unique, avec la condition identique à celle proposée par Bortnikov *et al.* [2009], à savoir : nous supposons qu'il existe un temps logique t_0 à partir duquel le flux reste suffisamment longtemps dans un régime stable en terme d'ensemble d'éléments uniques [Godfrey *et al.*, 2006] pour atteindre le régime stationnaire. Cette hypothèse est nécessaire pour rendre la définition d'échantillon significative. Cependant, nous

pouvons relâcher cette hypothèse dans le modèle à fenêtre glissante pour laquelle notre algorithme produira un flux uniforme et frais sur les éléments appartenant à la fenêtre courante seulement. L'utilisation de l'algorithme PROPORTIONNEL ou DIVISEUR (*cf.* section 3.3.2) en lieu et place du CM Skecth dans l'algorithme 3.11 permet d'adapter directement et simplement notre approche au modèle des fenêtres glissantes. Une seconde approche, orthogonale et actuellement en cours d'étude, repose sur la méthode dite de pivot [Chauvet, 2012]. Celle-ci permettra de prendre en compte des adversaires plus puissants, et ce, dans tous les modèles introduits en section 2.1.

CHAPITRE 4

Applications pratiques

« Pfuel était un de ces théoriciens si férus de leur théorie qu'ils en oublient le but,
à savoir l'application pratique : par amour de la théorie, il méprisait toute pratique. »

La guerre et la paix (tome 2), par Léon Tolstoï (1869).

« Quelles applications pratiques pouvez-vous tirer de vos travaux ? Et finalement, à quoi ça sert ? » Quiconque n'a jamais eu à répondre à de telles injectives lors du traditionnel moment des questions/réponses à l'issue d'un exposé me jette la première pierre.

Dans la grande majorité des travaux présentés dans le chapitre précédent, j'ai toujours eu à cœur de motiver mes recherches avec un objectif applicatif bien défini. Ce chapitre propose un tour d'horizon pleinement subjectif de ces derniers.

Nous aborderons dans un premier temps l'utilisation des résumés de distribution dans la métrologie des réseaux, puis l'usage des algorithmes à flux répartis pour la sûreté de fonctionnement. Enfin, nous présenterons comment optimiser les architectures de traitement de flux à la volée (type Apache Storm [2014] ou Spark Streaming [2013]) par l'utilisation d'agrégats.

4.1 Métrologie des réseaux

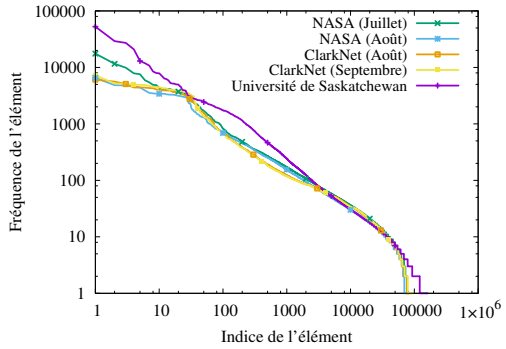
L'utilisation des algorithmes introduits dans le chapitre précédent est pertinente dans le cadre de la métrologie des grands systèmes et réseaux. Un des objectifs repose sur l'estimation de la similarité entre plusieurs flux, ou entre un flux observé et un flux attendu. L'extraction de statistiques par l'utilisation de ces méthodes permet par exemple de calculer des métriques de surveillance sur des échantillons ou des agrégats, ou de détecter en temps réel la présence d'intrusion dans le trafic réseau [Chan *et al.*, 2012, Salem *et al.*, 2010, Yuan et Mills, 2005]. L'approche flux répond parfaitement aux contraintes du trafic IP, où les routeurs ne peuvent dédier à la métrologie une part importante de leur capacité mémoire et calculatoire. C'est d'autant plus vrai pour les applications réparties telles que les réseaux de capteurs.

Nous avons ainsi soumis des jeux de données issus d'applications en production à nos algorithmes. Ces données réelles nous donnent une représentation réaliste des besoins et des comportements de nos algorithmes en régime usuel. Les jeux de données utilisés sont disponibles sur le dépôt en ligne de l' Internet Traffic Archive [2008]. Deux d'entre eux représentent deux semaines de consignations de requêtes HTTP sur un serveur web du fournisseur d'accès Internet ClarkNet (ancien service de la métropole Baltimore-Washington DC). Deux autres contiennent les fichiers de journalisation de deux mois de requêtes HTTP sur le serveur web du centre spatial Kennedy de la NASA. Le dernier jeu de données représente sept mois de requête HTTP sur le serveur web de l'université de Saskatchewan, au Canada. La table 4.1 présente quelques statistiques descriptives de ces jeux de données, en terme de longueur de flux (*cf.* « # éléments (m) »), de nombre d'éléments distincts dans chaque flux (*cf.* « # distincts (n) ») et de nombre d'occurrences de l'élément le plus fréquent (*cf.* « fréq. max. »). La figure 4.1a illustre l'allure du vecteur de fréquences de chaque trace réelle, ordonné de manière décroissante. Chacun de ces jeux de données semble suivre une loi de Zipf (aspect linéaire en échelle double logarithmique).

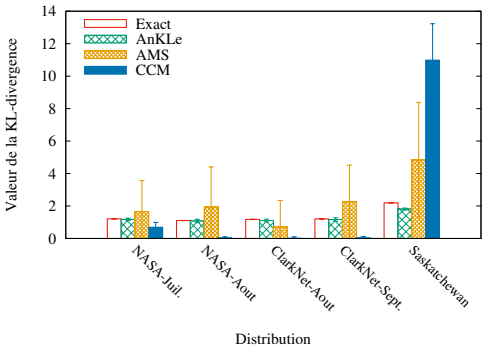
Nous avons ainsi alimenté les algorithmes repris ci-après avec ces traces et tirons les conclusions

Table 4.1 – Statistiques des 5 jeux de données réels de Internet Traffic Archive [2008].

Nom de la trace	Numéro	# éléments (m)	# distincts (n)	fréq. max
NASA (Juillet)	1	1 891 715	81 983	17 572
NASA (Août)	2	1 569 898	75 058	6 530
ClarkNet (Août)	3	1 654 929	90 516	6 075
ClarkNet (Septembre)	4	1 673 794	94 787	7 239
Saskatchewan	5	2 408 625	162 523	52 695



(a) Distribution des fréquences pour chaque jeu de données réelles.



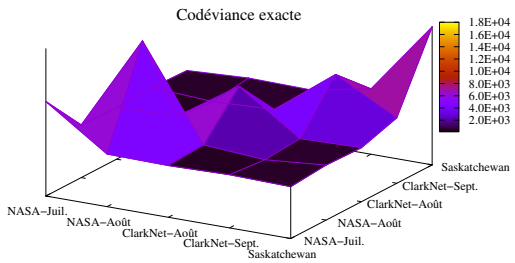
(b) AnKLe : Comparaison entre la valeur exacte de la KL-divergence et différents estimateurs sur des flux réels.

Figure 4.1 – Vecteurs de fréquence des jeux de données réelles et estimation de la KL-divergence par l'utilisation de AnKLe.

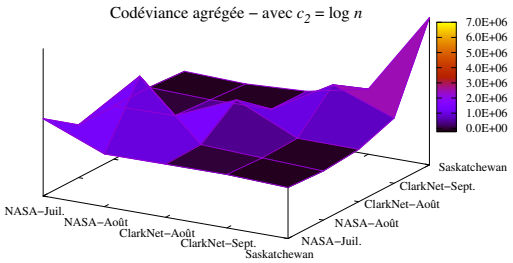
de leur comportement en environnement réel :

AnKLe : comparaison avec un flux nominal La figure 4.1b confirme les résultats observés sur les flux synthétiques, présentés en section 3.1.3. En effet, la pertinence et l'efficacité d'AnKLe sont observables par comparaison aux autres approches de l'état de l'art. Quelle que soit la trace donnée, AnKLe permet d'obtenir des estimations de la KL-divergence avec un flux uniforme nettement plus précises qu'avec les estimateurs CCM [Chakrabarti *et al.*, 2007] ou AMS [Alon *et al.*, 1996]. Cette meilleure précision est par ailleurs valable tant sur la moyenne de ces approximations que sur leur écart type.

Codéviance : étude de la matrice de dispersion Comme dans la section 3.1.1, la figure 4.2 représente les matrices de codéviance exactes et estimées pour les 5 flux de la table 4.1. Ici encore, les axes des abscisses et ordonnées représentent les jeux de données, et l'axe des cotes représente la valeur de la cellule correspondante dans la matrice. À nouveau, en fixant le paramètre c_2 à $\log n$, nous obtenons par notre algorithme d'approximation des courbes de niveaux très similaires à celles issues des calculs exacts sur le flux complet. Il est à noter que plusieurs ordres de grandeurs séparent les valeurs résultantes, lesquels s'expliquent par le facteur de surestimation

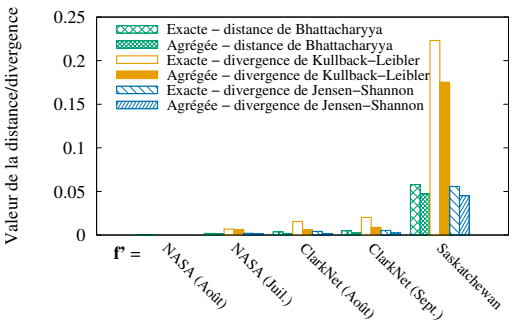


(a) Matrice de codéviance exacte

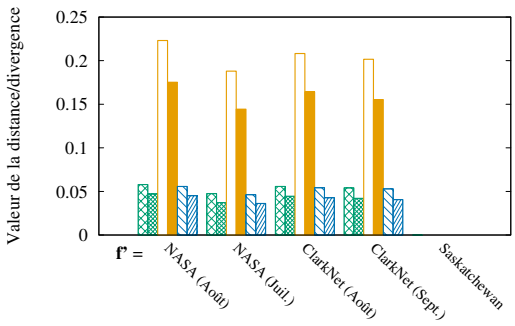


(b) Matrice de codéviance agrégée obtenue à partir de notre algorithme réparti avec $c_2 = \log n$

FIGURE 4.2 – Courbes de niveaux représentant les matrices de codéviance sur des flux réels.



(a) Vecteur de fréquences $\mathbf{f}$ correspondant à la trace de la NASA du mois d'août.



(b) Vecteur de fréquences $\mathbf{f}$ correspondant à la trace de l'université de Saskatchewan.

FIGURE 4.3 – Comparaison entre les métriques Sketch-$\star$ et les valeurs exactes sur des traces réels.

de notre algorithme. Cette surévaluation peut cependant être compensée, voire supprimée, par un mécanisme d'apprentissage court permettant d'évaluer le facteur $\mathcal{E}_k$.

Métrique Sketch-$\star$: détection des changements de comportement L'application de nos métriques Sketch-$\star$ sur les jeux de données réelles permet d'exhiber des résultats intéressants, notamment sur la détection de différences de comportements des flux, pourtant issus d'une distribution de même type. La figure 4.3 présente les valeurs exactes et estimées des trois métriques introduites en section 3.1.2, pour les cinq jeux de données. Visuellement, il se détache deux groupes : les quatre premières traces réelles et la cinquième, isolée. En effet, les statistiques de la table 4.1 illustrent bien que cette dernière est plus massive, elle possède plus d'éléments distincts et quelques éléments bien plus surabondants que les autres. Quoique toutes les distributions possèdent une allure identique comme le confère la figure 4.1a, la fiabilité de l'estimation de $\widehat{\phi}_k$ est

particulièrement bonne, en faisant un très bon outil pour comparer tout couple de flux de données. Cette fiabilité est diminuée, mais dans une faible mesure, lorsque les flux comparés sont différents (*cf.* figure 4.3b). Nous avons également observé dans [Anceaume et Busnel, 2012] le fort impact de la non-symétrie de la KL-divergence sur le calcul de métrique (à la fois sur les flux complets ou sur les agrégats). Cependant, celui-ci n'est pas concrètement visible sur les jeux de données présentés dans le cadre de cette étude.

4.2 Sûreté de fonctionnement

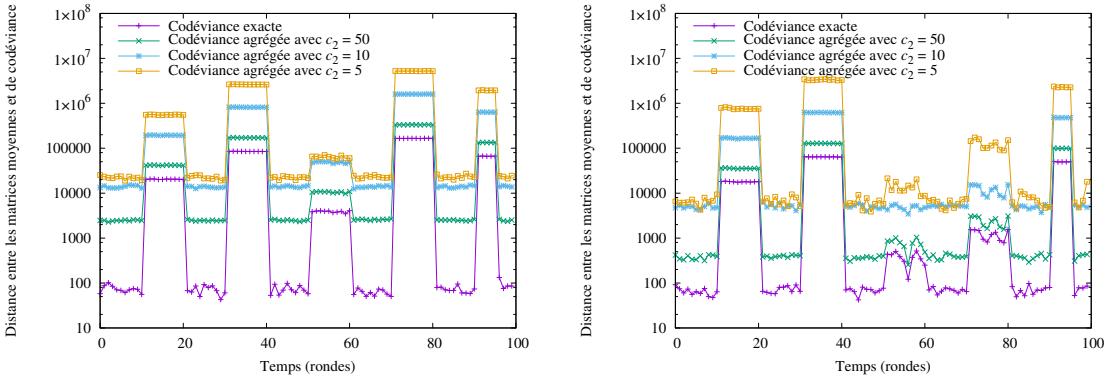
Nous avons également choisi de pousser plus en avant ces résultats et de les appliquer à la sûreté de fonctionnement des réseaux et grands systèmes. Nous présentons ci-après l'application de la codéviance à la détection d'attaques par déni de service réparti, puis le repérage d'icebergs sur des routeurs. Ces travaux ont été développés dans le cadre des projets SocioPlug [2013] et DeSceNt [2014].

4.2.1 Détection de déni de service réparti

Une attaque par déni de service (DoS) a pour objectif de faire tomber une ressource en ligne en inondant cette ressource avec plus de requêtes qu'il n'est capable d'en traiter. Une attaque par déni de service distribué (DDoS) possède le même objectif mais initié par plusieurs sources, préalablement infectées par un logiciel malveillant (*malware*), impliquant l'arrêt immédiat de la cible de l'attaque (tels que des sites de e-commerce par exemple). Appliquer une surveillance continue du trafic passant par les routeurs réseaux est une approche classique pour détecter et limiter ces attaques, via une analyse des signatures IP très fréquentes pouvant représenter des cibles potentielles.

L'utilisation de la codéviance dans ce cadre d'étude parait pertinente. En effet, comme introduit par Yuan et Mills [2005], un tel outil de surveillance du trafic réseau permettrait de fournir aux applications de métrologie des informations significatives sur les changements de comportement du trafic. Ainsi, ces applications seraient en mesure de remonter des alertes sur la présence d'attaques DDoS, et leur localisation.

La figure 4.4 illustre l'efficacité de notre algorithme réparti de codéviance agrégée dans la détection de différents scénarii d'attaques, en temps réel. Spécifiquement, nous calculons pour chaque ronde de l'algorithme réparti, la distance entre la matrice de codéviance $\widehat{\mathrm{COD}}$ construite à partir des flux observés et la moyenne des matrices de codéviance $\mathbb{E}(\widehat{\mathrm{COD}}_N)$ calculée en régime stable et « normal ». Cette distance a été proposée par Jin et Yeung [2004]. Formellement, étant donné



(a) Distance proposée par Jin et Yeung [2004] **(b)** Distance proposée par Anceaume et Busnel [2014a]

FIGURE 4.4 – Evolution de la distance des matrices de dispersion en fonction du temps logique.

les matrices M et M' de taille s, considérons la distance suivante :

$$\|M - M'\| = \sqrt{\sum_{i=1}^{s}\sum_{j=1}^{s}(M_{i,j} - M'_{i,j})^2}.$$

Nous évaluons pour chaque ronde r la variable d_r définie par

$$d_r = \|\widehat{\mathrm{COD}}_r - \mathbb{E}(\widehat{\mathrm{COD}}_N)\|.$$

Jin et Yeung [2004] proposent ainsi de détecter les comportements anormaux, par rapport à ceux dits normaux, comme suit. D'abord, le trafic réseau est analysé en régime stable et, à la fin de l'analyse, un point c et une constante a pour d_r sont estimés pour vérifier que $\mid d_r - c \mid < a, \forall r \in \mathbb{N}^*$. La constante a est sélectionnée comme la borne supérieure des variables indépendantes et identiquement distribuées $\mid d_r - c \mid$. Ensuite, lors de l'investigation de présence potentielle d'attaque DDoS sur un réseau, pour tout r, les motifs qui ressortent avec $\mid d_r - c \mid > a$ seront considérés comme anormaux. Cependant, nous estimons qu'il n'est pas possible de caractériser un trafic réseau normal *a priori*, nous avons adapté cette définition en observant le comportement récent du trafic sous surveillance. En détail, pour chaque ronde $r > 1$, la distance est calculée entre la matrice de codéviance $\widehat{\mathrm{COD}}_r$ courante et la matrice moyenne $\mathbb{E}(\widehat{\mathrm{COD}}_r)$ calculée sur les rondes précédentes $1, \ldots, r-1, r$:

$$\mathbb{E}(\widehat{\mathrm{COD}}_r) = \frac{(r-1)\mathbb{E}(\widehat{\mathrm{COD}}_{r-1}) + \widehat{\mathrm{COD}}_r}{r}.$$

Comme l'illustre la figure 4.4b, cette distance produit de meilleurs résultats, plus cohérents, qu'en utilisant celle proposée originellement par Jin et Yeung [2004], laquelle est présentée dans la figure 4.4a, sans nécessiter d'analyse a priori du trafic observé.

À partir de ces distances, nous avons alimenté notre algorithme réparti avec différents modèles

de trafic réseau. En particulier, la figure 4.4 représente la valeur de ces deux distances, en fonction du temps (en terme de rondes), quand la codéviance est calculée exactement et lorsqu'elle est estimée avec notre algorithme pour différentes valeurs de c_2. Il est ainsi observable que, nonobstant les deux ordres de grandeur entre les valeurs exactes et estimées, l'allure des variations de codéviance reste similaire, principalement pour la figure 4.4b. Différents scénarii d'attaques ont été simulés. Entre les rondes 0 et 10, tous les 10 nœuds du système reçoivent des traces synthétiques qui suivent la même distribution nominale (*e.g.,* une loi de Poisson). Des rondes 10 à 20, une attaque « ciblée » est lancée en inondant un unique nœud (*i.e.,* un flux parmi les dix adopte désormais une loi de Zipf avec un paramètre $\alpha = 4$). Ceci donne lieu à un accroissement drastique et abrupt de la distance. L'observation révèle que la codéviance estimée suit exactement celle de l'exacte, fournissant un résultat particulièrement intéressant. Ensuite, après un retour à un trafic normal, la moitié des flux sont remplacés par des flux de distribution de Zipf (entre les rondes 30 et 40), représentant une attaque par inondation sur un groupe de nœuds. Comme pour la charge précédente, les matrices de codéviance sont fortement impactées par cette attaque. Ente les rondes 50 et 60, les flux suivent tous une distribution de Zipf de paramètre $\alpha = 1$ représentant un réseau chargé de manière instable mais pas une attaque a proprement dit. Durant les deux dernières périodes d'attaques, toutes les traces suivent respectivement une loi de Zipf avec différentes valeurs de paramètre $\alpha \geq 2$, lesquelles représentent une attaque par inondation vers un groupe de nœuds ciblés.

Ces expériences permettent d'extraire la valeur seuil a. Par exemple, fixer a à $1\,000$ pour la codéviance exacte et pour la codéviance agrégée avec $c_2 = 50$ conduit à la détection de toutes les attaques DDoS présentes. Considérant la codéviance agrégée avec $c_2 = 10$ (respectivement 5), a devrait être fixé autour de $10\,000$ (respectivement $50\,000$) pour détecter également toutes les attaques.

La leçon principale à retenir de ces évaluations est la bonne performance de notre algorithme réparti, quel que soit le modèle d'attaque subie.

4.2.2 Détection d'iceberg

Une nouvelle forme de DDoS a récemment vu le jour, consistant à masquer les attaques en répartissant celles-ci stratégiquement sur différentes routes. Ainsi, localement, les sous-flux malveillants n'apparaissent pas comme une menace directe, mais globalement, ils représentent une proportion significative du trafic réseau [Manjhi *et al.*, 2005]. Le terme d'*attaque par iceberg* a été proposé pour représenter ce type d'attaque, chaque routeur ayant seulement la visibilité d'une infime partie de l'attaque [Zhao *et al.*, 2010]. Une solution naturelle serait donc de centraliser les informations collectées localement sur chaque routeur par analyse de leur trafic respectif. Cela implique donc que *(i)* les routeurs soient capables de traiter en ligne un flux de très grande taille pour découvrir la présence potentielle d'une attaque par iceberg suffisamment tôt et que *(ii)* les communications entre les routeurs et le coordinateur central soient réduites au maximum afin

Table 4.2 – Empreinte mémoire avec tampons et compteurs de 32 bits.

Θ	CM Sketch	$\sum_{k=1}^{5} \lvert\Gamma_k\rvert$	$\lvert\Lambda_s\rvert$	Gain d'espace mémoire
$0,1$	$3,48$ kB	$0,77$ kB	$0,32$ kB	$98,57\%$
$0,005$	$69,59$ kB	$7,20$ kB	$6,40$ kB	$74,00\%$

d'éviter que ce dernier soit lui-même la cible d'une attaque DDoS. Évidemment, diminuer la fréquence de ces échanges ne doit pas se faire au détriment de la latence de détection des icebergs et, en aucun cas, introduire de faux-négatifs (*i.e.,* non-détection d'un iceberg réel).

Nous avons proposé une solution au problème de détection et d'identification d'icebergs, tout en garantissant les deux propriétés ci-dessus [Anceaume *et al.*, 2015c]. Nous utilisons pour cela l'algorithme introduit en Section 3.2, du chapitre précédent. En résumé, localement, seules les menaces locales identifiables par un routeur sont surveillées et transmises au coordinateur, qui sera en charge d'interroger les autres routeurs pour vérifier la véracité de cette attaque. Nous obtenons ainsi une (ε, δ)-approximation du problème d'identification des icebergs, pour tout $\varepsilon \in [0,1]$ et $\delta \leq 1/2$.

Nous avons implémenté notre solution sur un prototype déployé sur un banc de test composé de 20 Raspberry Pi Modèle B et deux serveurs interconnectés par un réseau Ethernet Gigabit. Chaque Raspberry héberge un nœud, l'un des deux serveurs joue le role du coordinateur et l'autre génère les S flux à destination des nœuds. Ces expérimentations se veulent une preuve de concept, les routeurs internets (tels que Cisco ou Juniper type M ou T) pourront largement gérer un code éxécuté sur un Raspberry Pi.

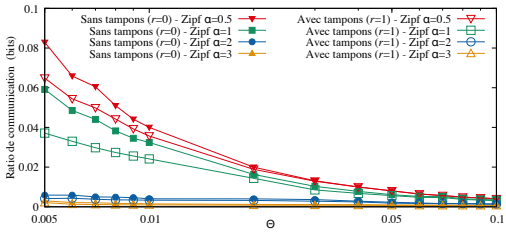
Les flux ont été produits à la fois à partir de traces réelles mais également générés synthétiquement, afin de mettre en exergue des comportements difficiles à obtenir sur des traces existantes, démontrant ainsi la robustesse de notre solution.

La probabilité de défaillance δ et l'erreur tolérée ε de notre algorithme ont été respectivement fixées à $\delta = 0,1$ et $\varepsilon = 0,1$. En utilisant les bornes démontrées dans [Anceaume *et al.*, 2015c], cela revient à une empreinte mémoire au maximum de $7,5\%$ de celle requise par l'algorithme naïf qui consisterait à maintenir un compteur pour chaque élément reçu. En réalité, nous montrons qu'utiliser seulement $1,43\%$ de la mémoire nécessaire à cet algorithme naïf est suffisant (*cf.* table 4.2). Ceci est atteint en réduisant le nombre de colonnes s_2 dans le CM Sketch à $s_2 = \lceil e/\Theta \rceil$, au lieu des $s_2 = \lceil 2(1-\Theta)/(0,1\Theta) \rceil$ initialement prévu (*cf.* section 3.2) [1].

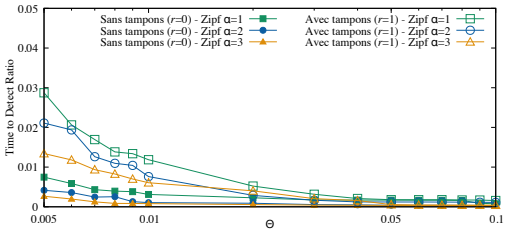
D'autres résultats numériques montrent que le surcoût de communication généré par notre solution reste très faible (moins de $1,2\%$ du nombre global d'éléments reçus par les routeurs). Enfin, ils illustrent que moins de 3% de lecture du flux global est nécessaire pour détecter tous les icebergs, s'il n'existe pas de variation majeure de la distribution de ceux-ci.

Plus précisément, la figure 4.5a illustre le ratio entre nombre de bits échangés par l'algorithme

1. Pour des raisons de simplification, nous présentons ici l'utilisation mémoire en codage 32 bits. Ceci conduit à une surestimation des besoins étant donné que $\log n$ et/ou $\log m$ bits sont suffisants.



(a) Ratio entre le nombre de bits échangé par notre algorithme et le nombre de bits reçus sur les 20 flux répartis, en fonction du seuil Θ.



(b) Ratio entre le temps nécessaire à notre algorithme pour détecter tous les icebergs et le temps de réception des 20 flux d'entrées, en fonction du seuil Θ.

FIGURE 4.5 – Efficacité et précision de notre solution, en fonction de Θ.
Paramètres : $m = 2\,000\,000$ et $n = 10\,000$.

réparti et le nombre de bits reçus sur le flux local, en fonction de Θ. Il est à noter que dans ces courbes, les points ont été reliés pour améliorer la lisibilité de la figure, mais ne présagent rien sur les valeurs effectives entre deux points réellement exécutés sur notre prototype. La première remarque repose sur le coût de communication additionnel négligeable induit par notre algorithme pour détecter des icebergs : moins de $8,5\%$ de la taille des flux sont échangés entre les nœuds de $\mathcal{S}$ et le coordinateur. Ceci s'applique même pour les distributions faiblement biaisées, représentant le pire des cas pour notre algorithme (*cf.* relation 3.12, page 34). De plus, l'impact des tampons Γ_k sur le surcoût de communication est important, surtout pour les distributions faiblement biaisées et des petites valeurs de Θ. Ces résultats sont clairement frappant en comparaison avec les travaux de Zhao *et al.* [2010], lesquels obtiennent un ratio de données brutes égal à 75% pour des distributions fortement biaisées [Zhao *et al.*, 2010, Section 7.B]).

La figure 4.5b illustre la latence de détection des icebergs. Cette latence est calculée comme le temps de détection de tous les icebergs, divisé par le temps nécessaire pour recevoir complètement tous les flux. Comme introduit précédemment, moins de 3% du flux global est nécessaire pour détecter tous les icebergs (dans le pire des cas de nos expérimentations et pour des flux ordonnés aléatoirement). La seconde remarque porte à nouveau sur l'impact des tampons Γ_k, lesquels introduisent logiquement un temps de détection plus long. Ainsi, avec des tampons de taille maximale, la latence de détection est augmentée par un facteur 4 alors que le surcoût de communication est réduit par un facteur 2 (*e.g.*, Zipf $\alpha = 1$ avec $\Theta = 0,005$ dans les figures 4.5a et 4.5b). L'étude de l'existence d'une valeur r optimale, qui permettrait de minimiser le temps de détection et le surcoût de communication, représente une question ouverte pertinente.

Enfin, nous avons alimenté notre prototype avec des jeux de données réels collectés lors d'une attaque DDoS [The CAIDA UCSD "Anonymized Internet Traces 2008" Dataset, 2008, The CAIDA UCSD "DDoS Attack 2007" Dataset, 2010]. Le flux global, réparti sur les 20 nœuds de notre prototype, représente $m = 2 \times 10^8$ éléments (codés sur 32 bits), et contient $n = 825\,695$ adresses de destination uniques. Ceci représente une trace de trafic de 15 minutes, surveillée sur des liaisons du *backbone* Internet OC192. Ce jeu de donnée est assurément plus large que les flux générés

Table 4.3 – Résultats sur des traces réelles avec $\Theta = 0,1$

Taille du flux global (m)	2×10^8
Nombre d'éléments distincts (n)	825 695
Icebergs existants	1
Icebergs détectés	1
Faux-positifs	0
Faux-négatifs	0
Fréquence de l'iceberg	$3,2 \times 10^6$
Fréquence estimée de l'iceberg	$3,4 \times 10^6$
Ratio de communication (bits) sans tampons	$3,6 \times 10^{-4}$
Ratio de communication (bits) avec tampons	$3,2 \times 10^{-4}$

synthétiquement, augmentant de fait le nombre de collisions dans le CM Sketch. Les principaux résultats issus de l'analyse de cette expérience sont résumés dans la table 4.3.

La cible de l'attaque DDoS représente l'unique iceberg, avec une probabilité d'occurrence égale à $0,15$ ($\Theta = 0,1$), alors que tous les autres éléments arrivent avec une probabilité inférieure à 5×10^{-3}. Comme indiqué dans la table 4.3, la cible de l'attaque est correctement détectée (sans faux positif, ni négatif). Le ratio de communication a été au plus égal à $3,6 \times 10^{-4}$. Il est à noter que le ratio a été calculé en ne prenant compte que les adresses contenues dans les messages échangés, et non pas sur le message lui-même, calcul qui aurait donné des résultats bien plus favorables pour notre solution. Ces bon résultats sont le fruit du grand écart entre les probabilités d'occurrence de l'unique iceberg et des autres éléments, ce qui est typiquement le cas dans les attaques DDoS classiques.

4.3 Traitement de flux temps réel

Les infrastructures de traitement de flux sont actuellement en plein essor, en tant qu'outils d'analyse sur des flux de données continus. Leurs temps de réponse inférieurs à la seconde couplés à leur adaptabilité pour le large échelle en font l'instrument privilégié de nombreux industriels du « Big data ».

Une application de traitement de flux, telle que dans Apache Storm [2014] ou Spark Streaming [2013], est modélisée par un graphe orienté acyclique, où les opérateurs sur les données, représentés par les sommets, sont interconnectés par des flux de d'éléments contenant les données à analyser, qui sont les arcs du graphe. Le passage à l'échelle est mis en œuvre lors de la phase de déploiement, où chaque opérateur de données peut être parallélisé en utilisant plusieurs instances, chacune étant en charge d'un sous-ensemble des éléments reçus sur le flux d'entrée de l'opérateur. Hirzel *et al.* [2014] ont proposé une vaste vue d'ensemble des optimisations possibles pour les systèmes de traitement de flux en temps réel.

4.3.1 Équilibrage de charge

Équilibrer la charge parmi les différentes instances d'un opérateur parallèle est primordial [Cardellini *et al.*, 2002]. Cela permet d'aboutir à une utilisation optimale des ressources et ainsi, d'atteindre un débit de sortie plus important, réduisant de fait la latence de traitement d'un élément en entrée.

La stratégie de répartition des éléments d'un flux sur les instances parallèles dépend fortement de la logique applicative implémentée par l'opérateur lui-même. Deux approches principales sont communément adoptées :

Groupement par aléa Les éléments sont assignés aléatoirement à une instance donnée.

Groupement par clé Les éléments sont assignés à une instance selon la valeur d'une donnée contenue dans l'élément lui-même.

Le groupement par aléa est le choix naturel des opérateurs sans-états (*stateless*), étant plus simple à mettre en œuvre (*e.g.*, en utilisant un ordonnancement dit de *tourniquet*, ou *round-robin*). Cependant, l'affaire devient plus complexe lorsque l'opérateur est à états (*stateful*). Les états doivent être synchronisés continuellement entre les instances, avec des dégradations de performances sévères à l'exécution.

4.3.1.1 Groupement par clé

Pour simplifier le déploiement parallèle d'opérateurs à états, une méthode populaire consiste à subdiviser le flux d'éléments en plusieurs sous-flux disjoints, au regard de la clé. Chaque instance de l'opérateur recevant l'un de ces sous-flux est donc assurée de recevoir tous les éléments contenant une clé spécifique. Une solution classique est de grouper les clés en utilisant des fonctions de hachage, induisant potentiellement un déséquilibre de charge des instances d'opérateurs en cas de distribution biaisée des clefs du flux d'entrée. Nous avons proposé une solution permettant d'obtenir une répartition de la charge proche de l'optimal, quelle que soit la distribution [Rivetti *et al.*, 2015b]. Cette solution repose sur le fait que l'équilibre est majoritairement influencé par les clés les plus fréquentes. En identifiant ces valeurs dominantes et en les groupant judicieusement avec des clés plus rares, les sous-flux ainsi obtenus permettent d'atteindre un équilibrage de charge quasi-optimal.

La figure 4.6 présente le fonctionnement et l'architecture de notre algorithme DKG (*Distribution-aware Key Grouping*), pour réaliser le groupement par clé sur k instances parallèles. Ce dernier fonctionne en trois phases successives :

1. *Apprentissage* : Lors de la première phase, l'algorithme prend connaissance de la distribution du flux. Pour cela, DKG effectue une projection aléatoire, via une fonction de hachage 2-universelle $h : \Omega \rightarrow [\![k\mu]\!]$ (figure 4.6.A), où μ est un paramètre de précision. En effet, plus le codomaine est grand, moins les « blocs » de la projection contiennent d'éléments,

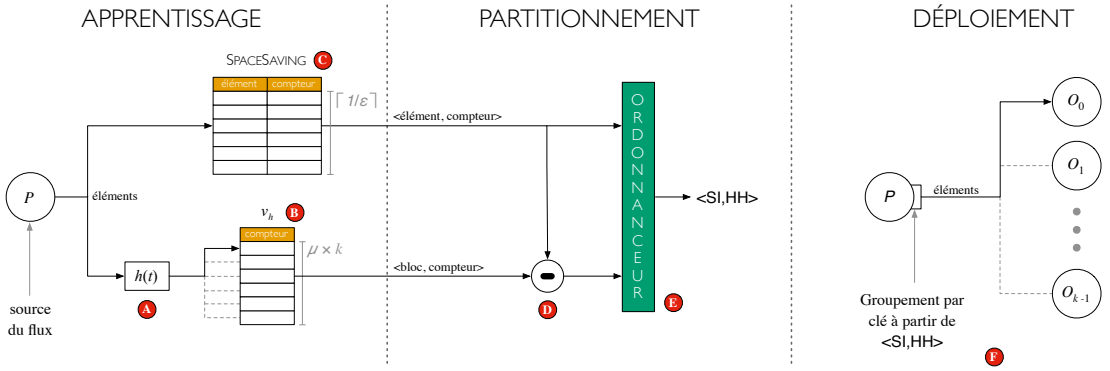


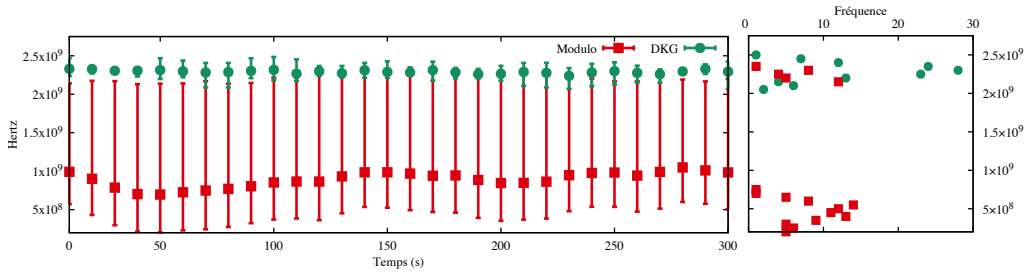
FIGURE 4.6 – Architecture et phases de travail de l'algorithme DKG.

permettant une meilleure granularité de la répartition. Le vecteur v_h contient ainsi la fréquence cumulée des éléments, pour chacun des blocs (figure 4.6.B). L'utilisation de cette projection ne garantit cependant pas que les éléments surabondants seront correctement répartis. En parallèle, DKG utilise donc l'algorithme SPACESAVING de Metwally *et al.* [2005] pour détecter ces éléments (figure 4.6.C).

2. *Partitionnement* : La phase suivante consiste à construire une surjection entre l'univers Ω et l'ensemble des instances de l'opérateur (O_0 à O_{k-1}), en tenant compte de l'information collectée dans la phase d'apprentissage. La structure de donnée du SPACESAVING contient ainsi une estimation des fréquences de tous les éléments surabondants. DKG retranche alors leur contribution dans le vecteur de fréquences agrégé v_h (figure 4.6.D). Puis, les deux structures de données sont concaténées et transmises à un ordonnanceur. Nous avons adapté le problème de l'ordonnancement pour multiprocesseurs [Garey et Johnson, 1979]. Formellement, nous considérons un ensemble de blocs $b_i \in \mathcal{B}$, chacun associé avec sa fréquence f_{b_i}, et un ensemble d'instances $O_j \in \mathcal{R}$ ($|\mathcal{R}| = k$), chacune associée à sa charge L_j (somme des fréquences des blocs assignés à l'instance O_j). Nous cherchons une association qui minimise la charge maximale des instances : $\max_{j \in [\![k]\!]} (L_j)$. Ce problème étant NP-dur, nous avons adopté l'heuristique classique *Least Processing Time First (LPTF)*, qui consiste à assigner le bloc ayant la fréquence la plus grande à l'instance qui a la plus petite charge courante. Ce bloc est retiré de $\mathcal{B}$ et le processus est réitéré jusqu'à ce que $\mathcal{B}$ soit vide (figure 4.6.E). Cet algorithme est une $(\frac{4}{3} - \frac{1}{3k})$-approximation de l'ordonnancement optimal.

3. *Déploiement* : La dernière phase utilise cette surjection pour réaliser le groupement par clé avec un équilibrage des instances quasi-optimal (figure 4.6.F). Lors de la réception d'un élément u, si u est un élément surabondant ($u \in \mathsf{HH}$), son instance associée est retournée. Sinon, le haché de u est calculé et l'instance associée au bloc $h(u)$ dans SI est retournée.

Nous avons montré que DKG fournit une association $(1+\beta)$-optimale, où β représente le seuil minimum de fréquence des éléments surabondants, en utilisant $O\left((k\mu + 1/\beta)\log m + \log n\right)$



(a) Chronologie **(b)** Distribution

FIGURE 4.7 – Consommation CPU (Hertz) pour 300 secondes d'exécution
Paramètres : $\beta = 0,1$, $\varepsilon = 0,05$, $\mu = 2$ and $k = 4$

bits dans la phase d'apprentissage et d'ordonnancement, et $O\left((k\mu + 1/\beta)\log n\right)$ bits une fois déployé.

Gedik [2014] a proposé une solution proche de DKG, en utilisant un algorithme de *comptage par perte* pour garder une trace des éléments surabondants, lesquels seront explicitement associés à un sous-flux. Cependant, dans leur solution, les autres éléments sont associés à l'aveugle. Une autre approche intéressante, proposé par Nasir *et al.* [2015], applique le *pouvoir du choix binaire* pour équilibrer la charge. Leur solution associe chaque clé à deux sous-flux distincts et transmet un élément correspondant sur le sous-flux le moins chargé. Cela revient à traiter un flux modifié où chaque valeur possède la moitié de sa charge du flux initial. Cependant, cette solution ne peut être appliquée dans le cas général, car elle nécessite une phase de *réduction* (à l'image du paradigme *MapReduce* [Dean et Ghemawat, 2008]) pour fusionner les états fractionnés.

Nous avons également confronté DKG à un environnement de simulations avec des flux générés, puis sur un prototype alimenté par des données réelles. L'ensemble de ces résultats est présenté dans [Rivetti *et al.*, 2015b]. L'implémentation du prototype comme fonction de groupement personnalisée dans Apache Storm [2014] a été testée sur les données du [ACM DEBS 2013 Grand Challenge, 2013, Query 3] [2]. Nous l'avons déployée sur le service Microsoft Azure [2014], utilisant des machines virtuelles standard de Tier A4 (4 cœurs et 7 Go de RAM).

La figure 4.7a illustre l'utilisation processeur (en Hertz) en fonction du temps (5 minutes d'exécution) pour DKG et Modulo (l'implémentation par défaut du groupement par clé d'Apache Storm [2014]) sur $k = 4$ instances. L'utilisation CPU a été mesurée comme la consommation moyenne de Hertz pour chaque instance sur des fenêtres de 10 secondes. Les valeurs moyennes, minimales et maximales sur les 4 instances sont ainsi représentées. La consommation CPU moyenne de DKG est proche du maximum pour Modulo, lequel a une utilisation moyenne bien plus faible. Comme attendu, l'algorithme Modulo abouti à une forte sous-utilisation des ressources.

La figure 4.7b présente la distribution des consommations CPU pour le même intervalle de temps. En d'autres termes, l'axe des abscisses représente le nombre de fois qu'une instance est parvenue

2. Le code source de ce prototype est accessible à `http://github.com/rivetti/dkg_storm`

à une valeur donnée (en Hertz). Pour éviter un écrasement de la distribution, nous avons discrétisé l'axe des ordonnées en arrondissant chaque valeur au multiple de 5×10^7 le plus proche. D'une part, les utilisations CPU pour les répliques de DKG sont concentrées entre 2×10^9 et $2,5 \times 10^9$ Hz, *i.e.,* toutes les instances ont une charge équitable avec DKG. D'autre part, Modulo atteint également cet intervalle, mais la plupart des points sont proches de 5×10^8 Hz. Alors que le groupement de clé induit par DKG équilibre la charge sur toutes les instances, Modulo sous-utilise plusieurs instances (en l'occurrence 3 lors de notre expérience) et concentre la charge sur une instance donnée. Cet optimisation des ressources permet d'obtenir un meilleur débit et une durée d'exécution significativement réduite. En particulier, dans nos expérimentations, DKG a au moins doublé le débit par rapport à Modulo, pour un nombre d'instances $k \in \{4, 5, 6, 8, 10\}$. Enfin, la phase d'apprentissage a un impact fort sur les performances et la mise en œuvre de notre solution. Cependant, nos tests ont révélé que les performances de DKG convergent après moins de 1 000 éléments reçus.

4.3.1.2 Groupement par aléa

Comme présenté précédemment, dans le cas des opérateurs sans état, l'utilisation du groupement par aléa est privilégiée pour répartir les éléments sur les instances parallèles des opérateurs. Les implémentations standard comme Apache Storm [2014] mettent en œuvre une simple stratégie de tourniquet qui garantit que tous les opérateurs reçoivent le même nombre d'éléments. Cette approche est pertinente tant que le temps de traitement des éléments sur un opérateur est homogène (chaque élément nécessite un temps d'exécution constant). La charge de chaque instance sera ainsi identique [Amini *et al.*, 2006, Arpaci-Dusseau *et al.*, 1999, Sharaf *et al.*, 2008].

Cependant, dans de nombreux cas d'usage pratiques, le temps de traitement de chaque élément dépend de l'élément lui-même. Par exemple, un opérateur peut implémenter une logique avec des branches conditionnelles, où seul un sous-ensemble des éléments traverse une branche donnée. Les éléments doivent donc être ordonnancés pour réduire le temps d'exécution global d'un opérateur parallélisé. Toutefois, le temps de traitement des éléments reçus n'est pas connu lors de l'ordonnancement.

Nous avons ainsi proposé une première solution de groupement par aléa qui s'attaque au problème de l'équilibrage de charge entres instances parallèles (non nécessairement uniformes) d'un opérateur générant des temps de traitement d'éléments hétérogènes [Rivetti *et al.*, 2016a]. Notre algorithme OSG (*Online Shuffle Grouping*) répartit les éléments sur les instances, avec une utilisation de ressource minimale et en ligne. OSG est en mesure d'adapter dynamiquement l'ordonnancement en fonction des changements de charge en entrée. Notre approche permet de réduire le temps de traitement de chaque élément, en utilisant une heuristique de *file d'attente la plus courte d'abord* [Gupta *et al.*, 2007], sur des files dont la taille est définie comme la somme des temps de traitement des éléments en attente. OSG utilise avantageusement des agrégats pour suivre l'évolution des temps de traitement de chaque élément transmis aux instances et repose

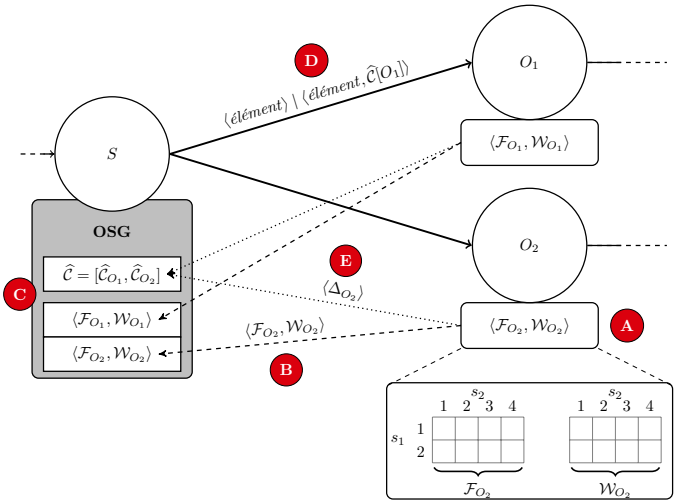


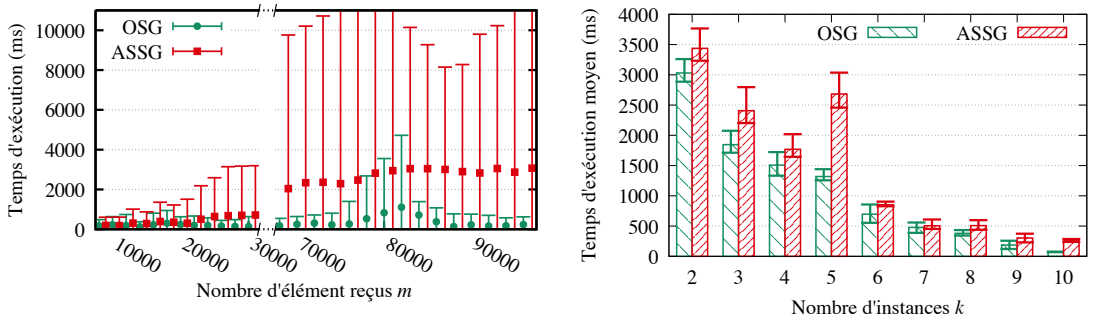
FIGURE 4.8 – Architecture et phases de travail de l'algorithme OSG.

sur un algorithme d'ordonnancement glouton pour répartir les éléments suivants à la volée.

La figure 4.8 présente le fonctionnement et l'architecture de notre algorithme OSG sur un exemple avec deux instances parallèles d'un opérateur O. Chaque instance utilise deux CM Sketch de même taille et partageant les même fonctions de hachage (figure 4.8.A). La première, dénotée $\mathcal{F}_{O_i}$, permet d'estimer la fréquence des éléments reçus sur cette instance. La seconde, dénotée $\mathcal{W}_{O_i}$, estime quant à elle le temps de traitement cumulé de chaque élément reçu sur cette instance (la somme des temps de traitement, en local, pour un élément donné, *i.e.,* le temps de traitement de l'élément multiplié par la fréquence locale de cet élément). Ces deux matrices sont mises à jour après chaque traitement d'un élément reçu sur le sous-flux d'entrée de l'instance. Périodiquement, ces deux matrices sont transmises à l'ordonnanceur (figure 4.8.B), puis réinitialisées. En outre, l'ordonnanceur conserve un estimateur de temps d'exécution cumulé pour chaque instance parallèle, dans un vecteur $\widehat{\mathcal{C}}$ de taille k (figure 4.8.C). L'ordonnanceur réaligne ce vecteur régulièrement, en transmettant des requêtes de synchronisation aux k instances, exploitant l'envoi d'un élément par exemple (figure 4.8.D). Chaque instance retourne ainsi la différence Δ_{O_i} entre le temps d'exécution cumulé local et l'estimation transmise par l'ordonnanceur (figure 4.8.E).

Nous proposons une analyse théorique de notre solution qui *(i)* montre que notre ordonnanceur en ligne est $(2-1/k)$-optimal par rapport à la version omnisciente et hors-ligne, et *(ii)* fournit des bornes d'erreur ainsi qu'une analyse probabiliste de la fiabilité de l'algorithme d'approximation des temps de traitement des éléments. Par conséquent, OSG permet d'obtenir un gain significatif en terme de temps d'exécution global (et pour tout élément du flux), par rapport à une stratégie de tourniquet classique, et ce, même dans un contexte de temps de traitement hétérogènes.

Enfin, nous avons évalué les performances d'OSG par simulation, puis en l'intégrant dans un prototype intégré dans Apache Storm [2014] sur la plateforme Microsoft Azure [2014], alimenté par des traces réelles issues de Twitter [2006] ou générées à partir du Social Network Benchmark [2015]. La figure 4.9 présente les résultats obtenus sur ce prototype.



(a) Evolution chronologique du temps d'exécution.

(b) Temps d'exécution global moyen en fonction du nombre d'instances k. — *Paramètre* : $m = 2\,115\,000$

FIGURE 4.9 – Temps d'exécution global de l'algorithme OSG sur le prototype intégré dans Apache Storm [2014] sur la plateforme Microsoft Azure [2014].

Le temps d'exécution en fonction du temps est présenté en figure 4.9a, à la fois pour notre solution OSG et pour la solution native ASSG (*Apache Storm Shuffle Grouping*). L'axe des abscisses représente le nombre d'éléments reçus jusqu'alors, et chaque point de la courbe représente le temps d'exécution global maximum, moyen et minimum, sur les 2 000 derniers éléments. Sur la partie gauche (jusqu'à $m = 10\,000$), OSG et ASSG fournissent les mêmes résultats. Puis, OSG commence à converger, permettant de réduire le temps d'exécution ainsi que sa variance (correspondant à l'étape B de la figure 4.8, et donc l'utilisation des données collectées). Après 75 000 éléments, nous avons injecté un changement brutal dans les temps de traitement réels des éléments, sur toutes les différentes instances. Les performances d'OSG se dégradent donc immédiatement (les matrices $\mathcal{F}$ et $\mathcal{W}$ contenant des données obsolètes). A partir de $m = 80\,000$, l'ordonnanceur reçoit les premières mises à jour et retourne progressivement vers un régime stable et quasi-optimal, démontrant bien la capacité d'OSG à s'adapter dynamiquement aux évolutions de charges.

La figure 4.9b illustre le temps d'exécution global minimum, moyen et maximum, pour les deux mêmes stratégies, en fonction du nombre d'instances parallèles. Excepté la crête inattendue avec ASSG pour $k = 5$, le temps d'exécution diminue logiquement lorsque k augmente. Pour toutes les configurations testées, OSG permet d'obtenir un temps moyen inférieur à la solution native. De surcroît, le temps maximum atteint avec la stratégie OSG est la plupart du temps inférieur au temps minimum induit par ASSG. De plus, pour atteindre ces bonnes performances, le surcoût de communication engendré par OSG ne représente que quelques milliers de messages additionnels, par rapport à la taille du flux d'entrée de l'ordre de 2 millions d'éléments.

4.3.2 Délestage de charge

Malgré les efforts précédents pour équilibrer la charge sur chacun des opérateurs de calcul, approvisionner correctement en ressources ce type d'infrastructure est loin d'être trivial. De nom-

breux facteurs doivent être pris en compte dans la conception de ces systèmes : la complexité calculatoire des opérateurs, les surcoûts induits par l'infrastructure, les caractéristiques des flux entrants, *etc.* Ces derniers varient imprévisiblement au cours du temps, à la fois en débit et en contenu. Les montées brutales en charge peuvent générer des goulets d'étranglement inattendus et contraindre les systèmes à violer les qualités de services envisagées (habituellement exprimées en temps de latence de traitement des éléments). Le délestage de charge est ainsi considéré comme une approche concrète pour réagir à des trafics fortement volatiles.

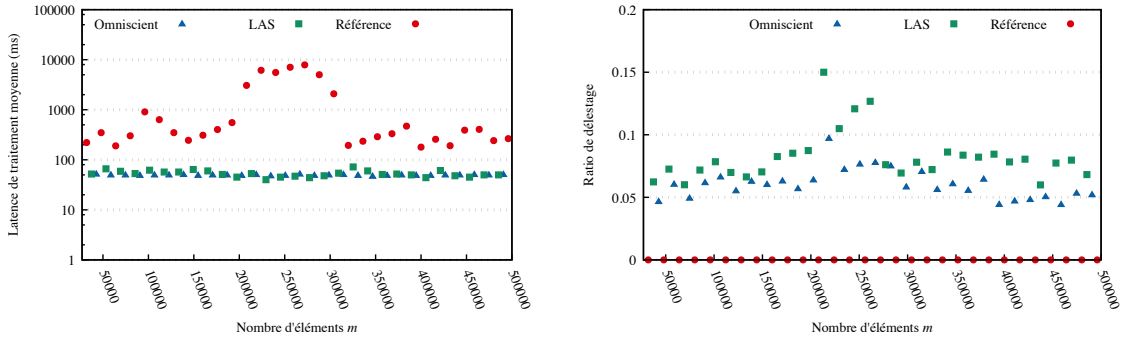
Les solutions de délestage existantes éliminent aléatoirement des éléments lorsqu'un pic de charge est détecté, ou bien, elles appliquent un modèle de coût prédéfini permettant de prendre la meilleure décision de délestage, de manière déterministe. Cependant, toutes ces solutions supposent que tous les éléments nécessitent un même temps de traitement [Abadi *et al.*, 2003, Babcock *et al.*, 2004, He *et al.*, 2014, Tatbul *et al.*, 2007, 2003].

Inspirés par les travaux présentés ci-dessus, nous avons proposé LAS (*Load-Aware Shedding*), une solution de délestage qui prend en compte le temps de traitement des éléments et permet de respecter les objectifs de temps de latence négociés [Rivetti *et al.*, 2016b]. Le fonctionnement interne de LAS est identique à celui de OSG, présenté en figure 4.8, permettant d'estimer le temps d'exécution de n'importe quel élément. Ainsi, il est possible de prévoir en amont le temps que les éléments passeront à attendre dans le tampon des opérateurs suivants. Si l'ajout d'un nouvel élément entraîne le dépassement du seuil de latence maximale tolérée, celui-ci sera simplement éliminé du flux pour garantir la qualité de service annoncée. À nouveau, notre solution ne nécessite aucune connaissance préalable sur le flux ou la topologie du système. Elle a été conçue pour s'adapter dynamiquement aux changements dans les flux d'entrée et les caractéristiques des opérateurs considérés.

Nous avons tout d'abord démontré la correction et l'optimalité de notre algorithme de délestage, lequel ne nécessite ni modèle de coût prédéfini, ni hypothèse sur la distribution des éléments. Puis, nous avons prouvé que LAS est une (ε, δ)-approximation de l'algorithme optimal.

Enfin, nous avons à nouveau intégré notre solution dans Apache Storm [2014] déployé sur la plateforme Microsoft Azure [2014], et alimenté par un jeu de données réelles issues de Twitter [2006]. Ce dernier contient $500\,000$ *tweets* avec environ $35\,000$ éléments distincts, l'élément le plus fréquent ayant une probabilité d'occurrences de l'ordre de $0,065$. Chaque tweet pouvant contenir différentes informations, leurs temps de traitement varient de 1 milli-seconde (pour 36% du flux) à 152 milli-secondes (le temps moyen est de $9,7$ milli-secondes). Le seuil de latence toléré a été fixé à 32 milli-secondes.

La figure 4.10 présente les performances de notre solution sur ce prototype, en fonction du temps, en comparaison avec *(i)* une stratégie *omnisciente* (*i.e.,* stratégie identique à LAS, mais avec un oracle qui connaît le temps de traitement exact de chaque élément du flux) et *(ii)* une stratégie de *référence* (sans agrégat) qui utilise le même algorithme de délestage de LAS, mais où le même temps de traitement estimé est identique pour tous les éléments, égal à la moyenne des temps ob-



(a) Temps de latence d'exécution des éléments.

(b) Nombre d'éléments éliminés par rapport à la taille du flux.

FIGURE 4.10 – Performance de l'algorithme LAS sur le prototype intégré dans Apache Storm [2014] sur la plateforme Microsoft Azure [2014].

servés jusqu'alors. La stratégie LAS permet d'atteindre des performances extrêmement proches de la stratégie omnisciente, éliminant une quantité d'éléments similaires. A l'inverse, la stratégie de référence possède des latences d'exécution avec au moins un ordre de grandeur supplémentaire. En effet, la moyenne des temps d'exécution étant faible par rapport au seuil, cette stratégie ne déleste aucun élément en cours de traitement (*cf.* figure 4.10b), alors qu'au moins 5 à 10% seraient nécessaires pour garantir la qualité de service attendue. Ces résultats, ainsi que ceux présentés dans [Rivetti *et al.*, 2016b], confirme l'efficacité de LAS à garder la maîtrise des temps d'attente, et ainsi à fournir des performances fiables et prévisibles, au prix d'un délestage d'une partie du flux.

CHAPITRE 5

Conclusions et perspectives

« Choisissez un travail que vous aimez
et vous n'aurez pas à travailler un seul jour de votre vie. »

Proverbe de Confucius (Ve siècle avant J.-C.)

Ce manuscrit résume une part significative de mes activités de recherche des dernières années. Je me suis efforcé de présenter plusieurs résultats originaux, de résumer l'état de l'art relatif à différents domaines de recherche et présenter différents points de vue autour de nos contributions.

Le chapitre 1 présente ainsi le contexte général de nos recherches et la diversité des problématiques liées aux flux de données à grande échelle. Par ces exemples, nous avons cherché à montrer plusieurs faits, notamment la richesse des modèles existants et l'intérêt de ces derniers dans de nombreux cas d'usage.

Dans le chapitre 2, nous avons présenté un survol des différents modèles de flux ayant été proposés, principalement lors de la dernière décennie. L'adjonction d'un adversaire adaptatif à ces modèles a permis d'ouvrir un pan de recherche vaste et passionnant.

Le chapitre 3 donne lieu à un panorama synthétique de certains de nos résultats, majoritairement algorithmiques. Nous y avons notamment traité des problématiques de résumé de distribution, d'approximation de métriques, de détection d'éléments surabondants ou d'échantillonnage. Ces solutions sont toujours étayées d'une analyse théorique et de simulations poussées pour mettre en exergue les performances de nos algorithmes.

Enfin, le chapitre 4 présente un catalogue de l'ensemble des mises en œuvre pratiques de nos algorithmes. D'une part, cela permet d'observer et d'analyser le comportement de nos contributions en environnement réaliste, mais également de démontrer les gains tangibles qu'apportent des algorithmes du modèle flux dans un environnement réel. Ces expérimentations ont été menées dans le cadre de projets collaboratifs, financés par l'Agence Nationale pour la Recherche, tels que SocioPlug [2013] ou DeSceNt [2014], mais également par des partenaires de la sphère industrielle, tels que les deux projets Microsoft Azure for Research Award [2015, 2016] que j'ai eu l'honneur de voir financés.

Dans la suite de cet ultime chapitre, concluant ce document, je revisite les diverses facettes du modèle de flux, à la lumière des travaux présentés précédemment, pour présenter les perspectives de recherche que j'envisage mener à l'avenir.

5.1 Vers la décentralisation des modèles répartis

Compte tenu du nombre potentiellement croissant de sources de données et d'unités de traitement dans les centres de données, il apparaît indispensable de développer des solutions entièrement décentralisées, répartissant la charge entre les participants et permettant de passer à l'échelle en terme de nombre d'entités déployées. Par conséquent, dans le contexte des flux répartis à grande échelle, l'approche collaborative dans la conception de systèmes dédiés apparaît nécessaire. En effet, ces derniers sont la plupart du temps développés dans un cadre applicatif précis. La généralisation de ces déploiements est pertinente comme le propose le projet SocioPlug [2013], dont l'objectif est fournir des modèles pour programmer, interroger et sécuriser des fédé-

rations d'entités à faibles ressources en préservant la symétrie d'utilisation, l'équité et le respect de la vie privée. Ces fédérations doivent ainsi être capables de délivrer un service sans nécessiter d'intervention extérieure (humaine ou matérielle), la collaboration devant être conçue de manière *auto-organisante*. Cette dernière notion a pour optique l'émergence d'un comportement général et global du système, à partir d'acteurs indépendants et possédant uniquement des informations restreintes et locales du système. Ainsi, la charge est répartie sur l'ensemble du système, et l'augmentation du nombre de participants n'entraîne pas la formation de goulots d'étranglement, ni de points de défaillance critique (par l'unicité d'un serveur), couramment observés sur les systèmes centralisés.

L'attrait de la décentralisation et de l'auto-organisation a été fortement en vogue depuis le début des années 2000 [Dolev *et al.*, 2007, Gordon, 2010, Ratnasamy *et al.*, 2001]. L'ouverture récente du monde des réseaux à des solutions auto-organisantes illustre pleinement la convergence des communautés systèmes, réseaux, télécommunications, bases de données, *etc.* Soutenus par 3GPP (*3rd Generation Partnership Project*) et le NGMN (*Next Generation Mobile Networks*), deux exemples récents intègrent des solutions de réseaux auto-organisants : LTE (*Long Term Evolution*), ainsi qu'une amélioration de l'UMTS (*Universal Mobile Telecommunications System*) [Brunner et Flore., 2009, Feng et Seidel, 2008]).

Dans le contexte présenté dans ce manuscrit, il devient à mon avis nécessaire de s'abstraire de l'organe coordinateur des modèles à flux répartis (*cf.* sections 2.1.3 et 2.1.4), en proposant des algorithmes totalement décentralisés. L'objectif est d'améliorer la sûreté de fonctionnement de nos algorithmes en s'affranchissant du point central de défaillance de ces algorithmes. De nombreux algorithmes répartis permettent de mettre en œuvre de manière décentralisée des mémoires partagées et du consensus, avec de très bonnes garanties, tels que Paxos [Lamport, 1998] ou les travaux de Castro et Liskov [2002], Chandra et Toueg [1996]. Ces protocoles ne sont cependant que modérément extensibles dans un contexte à large échelle. La proposition de nouvelles extensions décentralisées dans le cadre des modèles à flux répartis est une piste d'investigation stimulante. De surcroît, cette dernière doit porter une attention spécifique à la recherche d'optimalité en terme de surcoût de communication. En effet, par exemple, la solution proposée dans le cadre de CASE [Anceaume *et al.*, 2015b] (*cf.* section 3.3.1) est loin d'être optimale, les flux étants redirigés continuellement d'un nœud à un autre.

5.2 Intégrer la sémantique des données

Une seconde perspective, orthogonale et complémentaire à la section précédente, repose sur l'intégration d'informations sémantiques dans les algorithmes de traitement de flux de données. En effet, la grande majorité des solutions proposées dans ce contexte ne présuppose aucune connaissance du contenu des flux *a priori*. Tous les modèles reposent sur une abstraction des éléments qui les composent, à l'image de l'ensemble des propositions présentées dans ce document.

Néanmoins, l'ajout d'informations sémantiques au moment même de la conception des algorithmes permettrait assurément d'optimiser les futures solutions. L'objectif sera ainsi de concevoir des méthodologies et algorithmes innovants, s'appuyant sur des approches développées dans la « science des données », laquelle partage les richesses de la statistique et de l'informatique.

La première piste concrète envisagée repose sur les données massives en santé (DMS), lesquelles ne se limitent pas à un grand volume de données ou de sources. Les données biomédicales se réfèrent à des notions de complexités, d'enjeux, et d'opportunités portées par l'analyse combinée de ces données [National Institutes of Health, 2015]. En recherche biomédicale, ces sources de données sont diverses, hétérogènes, désorganisées, massives et multimodales, étant produites par diverses entités (chercheurs, praticiens, établissements hospitaliers, appareils nomades, *etc.*). Les DMS ont radicalement changé la manière dont l'information est traitée par les chercheurs en médecine [Toubiana et Cuggia, 2014], les données collectées et produites étant à présent potentiellement partagées et réutilisables. Cette quantité d'information promet l'émergence de nouvelles fonctionnalités médicales, incluant l'aide à la décision clinique, la pharmacovigilance, l'épidémiologie, la gestion sanitaire [Raghupathi et Raghupathi, 2014].

Deux projets de recherches complémentaires, dont j'ai rejoint le consortium, viennent d'être acceptés pour financement par l'ANR, dans le cadre d'études prospectives alliant informatique et statistique pour les DMS : INSHARE [2016] et de BigClin [2016]. Suite à l'intégration de nombreuses sources de données dans un centre de données spécialisé, l'un des objectifs communs est de développer un moteur analytique qui prendra en compte de nombreux critères, tels que le biais des données, le partitionnement, l'équilibrage de charge, la disponibilité des réplicats, la conception d'agrégats et d'indicateurs statistiques, *etc.* De nouvelles métriques adéquates seront proposées pour traiter en parallèle de grands volumes de données, en temps réel ou non. Par exemple, nous envisageons de concevoir des agrégats reposant sur des espaces de Hilbert [Berlinet et Thomas-Agnan, 2004].

Une seconde voie de recherche considérée incluant un enrichissement sémantique concerne les données transitant sur les réseaux mobiles. Le développement d'applications dans ce contexte a majoritairement suivi un patron de conception sous-optimal, se limitant le plus souvent à répliquer des services préexistants sur les réseaux filaires tels que les jeux en ligne ou les interactions ubiquitaires avec les réseaux sociaux. Quelques exceptions notables ont vu le jour comme des assistants de navigation, lesquels utilisent des informations générées par les capteurs embarqués dans les appareils nomades (en particulier, les récepteurs GPS), afin de faciliter la mobilité des utilisateurs Raptis *et al.* [2005]. Bien que cette approche améliore la productivité et la satisfaction des utilisateurs, les capacités grandissantes de ces appareils sont souvent sous-utilisées (en terme de puissance de calcul, de communication ou de capteurs).

Nous avons dernièrement conjecturé que le déploiement de services répartis pour appareils mobiles nécessitent un changement de paradigme [Busnel *et al.*, 2013]. Nous y avons notamment proposé un programme de recherche, ayant un impact potentiel à la fois scientifique et sociétal.

L'objectif repose sur trois concepts : *(i)* la possibilité de partager les ressources calculatoires avec les autres appareils à proximité, *(ii)* une communication « permissive » dont les motifs protocolaires seront appris en ligne, et finalement, *(iii)* la mise en œuvre d'un stockage coopératif de la connaissance commune acquise précédemment. Ces pistes de recherches se situent au croisement des réseaux ad-hoc, du calcul ubiquitaire et du traitement de données à la volée. Au niveau social, la mise à disposition de bibliothèques logicielles et de nouveaux standards de communication ouvrirait la voir à un déploiement viral d'une nouvelle classe d'applications, qui pourrait contribuer à améliorer la qualité d'expérience des utilisateurs.

De surcroît, savoir prédire le débit d'une connexion mobile à venir aurait également un impact significatif sur la qualité d'expérience en utilisant les réseaux cellulaires, car cela permettrait d'offrir d'immenses possibilités, notamment aux fournisseurs de services Internet. Ces derniers pourraient ainsi adapter leurs contenus à la qualité de service accessible par l'utilisateur, dans le but de maximiser sa qualité d'expérience. Nous avons proposé récemment une étude qui illustre les capacités de prédiction atteignables grâce à des données collectées à différents niveaux du réseau [Samba *et al.*, 2016]. Nous avons ainsi proposé une approche simple fondée sur des méthodes d'apprentissage pour prédire le débit d'une connexion à partir d'information minimale sur le contexte d'utilisation.

Dans des travaux futurs, un comportement adaptatif à la qualité d'expérience pourrait être développé par les fournisseurs de services en fonction du débit prédit. Un des problèmes ouverts de cette approche repose sur la prise en compte des brusques changements de couverture durant une connexion, lors d'un changement de relais par exemple. La connaissance du contexte environnemental de l'utilisateur considéré représente une information sémantique primordiale à prendre en considération. Des méthodes additionnelles tolérantes aux évolutions des conditions de couverture et permettant la prédiction de la mobilité (*e.g.,* par des études de corrélations ou d'apprentissage statistique) représentent ainsi de bons candidats pour répondre à ce défi.

5.3 Ouverture vers d'autres modèles de flux

En dernier lieu, une perspective de recherche particulièrement attractive se trouve au croisement du traitement des flux de données et de l'algorithmique des graphes. En effet, les systèmes à large échelle et les grands réseaux de communications sont souvent représentés par des graphes de communication ou d'interaction sociales par exemple. La dimension de ces objets explose littéralement dans les systèmes à grande échelle, tels que les réseaux de téléphonie mobile, les réseaux sociaux ou le routage sur internet, rendant les solutions classiques de la théorie des graphes inadaptées.

Par exemple, le modèle des flots de liens permet de représenter l'activité d'un système complexe. Au cours du temps, un lien apparaît lorsque deux entités du système entrent en interaction. Ces flots de liens peuvent notamment représenter la dynamique des interactions humaines,

lesquelles impliquent de nombreuses applications, de la diffusion épidémique aux modèles de mobilités [Starnini *et al.*, 2013]. L'étude de ces flots de liens, par leur dynamique, la détection de motifs récurrents, ou le groupement de sommets (*clustering*), représente un défi passionnant et pertinent. Viard *et al.* [2015] proposent par exemple d'étudier ces grands graphes dynamiques (*i.e.,* dont la topologie évolue au cours du temps) à travers le prisme des sciences des réseaux et des propriétés structurelles de ces objets.

Seulement, l'analyse de ces grands graphes en temps réel nécessite de nombreuses ressources de calcul. L'une des approches en vogue récemment repose sur le groupement de sommets (respectivement de liens) pour diviser le traitement sur des opérateurs parallèles [Chen *et al.*, 2015, Vaquero *et al.*, 2013, Xie *et al.*, 2014]. Cette technique de pré-traitement des graphes permet d'améliorer l'équilibrage de charge entre opérateurs. Cependant, le coût du partitionnement du graphe complet est absolument prohibitif. De nouvelles contributions pour découper le graphe à la volée ont ainsi vu le jour récemment, avec un paradigme issu du modèle flux [Petroni *et al.*, 2015, Xu *et al.*, 2014]. Ces algorithmes permettent ainsi de répartir les sommets (respectivement les arcs) du graphe rapidement, de manière adaptative et naturellement parallélisable. Néanmoins, des stratégies trop naïves conduisent rapidement à répliquer une grande proportion des éléments en entrée sur différentes instances, impliquant de fait un besoin de synchronisation important pour garantir la cohérence des données, entamant sérieusement les performances [Petroni, 2015].

L'élaboration et l'analyse de solutions innovantes dans ces nouveaux modèles représentent un défi motivant, enthousiasmant et séduisant.

Pour conclure, la quantité de questions ouvertes dans le champ de l'analyse et du traitement des flux de données à la volée reste pléthorique. Les cas d'utilisation semblent illimités dans notre société où le numérique prend une place de plus en plus importante tous les jours [Gartner Inc., 2015].

Bibliographie

Publications personnelles

N. Alhadad, Y. Busnel, P. Serrano-Alvarado et P. Lamarre : Trust evaluation of a system for an activity with subjective logic. *In Proceedings of the 11th International Conference on Trust, Privacy, and Security in Digital Business (TrustBus)*, pages 48–59, 2014.

N. Alhadad, P. Lamarre, Y. Busnel, P. Serrano-Alvarado, M. Biazzini et C. Sibertin-Blanc : Sociopath : Bridging the gap between digital and social worlds. *In Proceedings of the 23rd International Conference on Database and Expert Systems Applications (DEXA)*, pages 497–505, 2012.

N. Alhadad, P. Serrano-Alvarado, Y. Busnel et P. Lamarre : Trust evaluation of a system for an activity. *In Proceedings of the 10th International Conference on Trust, Privacy, and Security in Digital Business (TrustBus)*, pages 24–36, 2013.

N. Alhadad, P. Serrano-Alvarado, Y. Busnel et P. Lamarre : System modeling and trust evaluation of distributed systems. *Transaction on Large-Scale Data- and Knowledge-Centered Systems*, 22:33–74, 2015.

E. Anceaume et Y. Busnel : An information divergence estimation over data streams. *In Proceedings of the 11th IEEE International Symposium on Network Computing and Applications (NCA)*, 2012.

E. Anceaume et Y. Busnel : Sketch *-metric : Comparing data streams via sketching. *In Proceedings of the 12th IEEE International Symposium on Network Computing and Applications (NCA)*, pages 25–32, 2013.

E. Anceaume et Y. Busnel : Deviation estimation between distributed data streams. *In Proceedings of the 10th European Dependable Computing Conference, Newcastle (EDCC)*, pages 35–45, 2014a.

E. Anceaume et Y. Busnel : A distributed information divergence estimation over data streams. *IEEE Transaction on Parallel and Distributed System*, 25(2):478–487, 2014b.

E. Anceaume, Y. Busnel et S. Gambs : Uniform and Ergodic Sampling in Unstructured Peer-to-Peer Systems with Malicious Nodes. *In Proceedings of the 14th International Conference on Principles of Distributed Systems (OPODIS)*, volume 6490, pages 64–78, 2010.

E. Anceaume, Y. Busnel et S. Gambs : Characterizing the adversarial power in uniform and ergodic node sampling. *In Proceedings of the 1st International Workshop on Algorithms and Models for Distributed Event Processing (AlMoDEP)*. ACM, 2011.

E. Anceaume, Y. Busnel et S. Gambs : Ankle : Detecting attacks in large scale systems via information divergence. *In Proceedings of the 9th European Dependable Computing Conference (EDCC)*, 2012.

E. Anceaume, Y. Busnel et S. Gambs : On the power of the adversary to solve the node sampling problem. *Transaction on Large-Scale Data- and Knowledge-Centered Systems*, 11:102–126, 2013a.

E. Anceaume, Y. Busnel, P. Lajoie-Mazenc et G. Texier : Reputation for inter-domain qos routing. *In Proceedings of the 14th IEEE International Symposium on Network Computing and Applications (NCA)*, pages 142–146, 2015a.

E. Anceaume, Y. Busnel, E. Le Merrer, R. Ludinard, J.-L. Marchand et B. Sericola : Anomaly characterization in large scale networks. *In Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 68–79, 2014.

E. Anceaume, Y. Busnel et N. Rivetti : Estimating the frequency of data items in massive distributed streams. *In Proceedings of the 4th IEEE Symposium on Network Cloud Computing and Applications (NCCA)*, pages 59–66, 2015b.

E. Anceaume, Y. Busnel, N. Rivetti et B. Sericola : Identifying global icebergs in distributed streams. *In Proceedings of the 34th IEEE Symposium on Reliable Distributed Systems (SRDS)*, pages 266–275, 2015c.

E. Anceaume, Y. Busnel, E. Schulte-Geers et B. Sericola : Optimization results for a generalized coupon collector problem. *Journal of Applied Probability*, 53(2):1–7, 2016.

E. Anceaume, Y. Busnel et B. Sericola : Uniform node sampling service robust against collusions of malicious nodes. *In Proceedings of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2013b.

E. Anceaume, Y. Busnel et B. Sericola : New results on a generalized coupon collector problem using markov chains. *Journal of Applied Probability*, 52(2):405–418, 2015d.

M. Bertier, Y. Busnel et A.-M. Kermarrec : On Gossip and Populations. *In Proceedings of the 16th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, 2009.

M. Bertier, Y. Busnel et A.-M. Kermarrec : Dynamic computation of population protocols. *In Proceedings of the 17th International Conference on Telecommunications (ICT)*, pages 707–714, 2010.

S. Bonomi, Y. Busnel, R. Baldoni et R. Prakash : FAROES : fairness and reliability using overlay expenseless set-out for duty-cycle optimization in WSN. *In Proceedings of the 22nd International Conference on Parallel and Distributed Computing and Communication Systems (PDCCS)*, pages 198–204, 2009.

Y. Busnel : Nothing can compare with a population, besides agents. *In Proceedings of the 21st International Conference on Telecommunications (ICT)*, pages 421–425, 2014.

Y. Busnel, R. Beraldi et R. Baldoni : A formal characterization of uniform peer sampling based on view shuffling. *In Proceedings of the 2nd International Workshop on Reliability, Availability, and Security (WRAS)*, 2009.

Y. Busnel, R. Beraldi et R. Baldoni : On the uniformity of peer sampling based on view shuffling. *Elsevier Journal of Parallel and Distributed Computing*, 71(8):1165–1176, 2011a.

Y. Busnel, N. Cruz, D. Gillet, A. Holzer et H. Miranda : Reinventing mobile community computing and communication. *In Proceedings of the 12th IEEE International Conference on Ubiquitous Computing and Communications (IUCC)*, pages 1450–1457, 2013.

Y. Busnel, L. Querzoni, R. Baldoni, M. Bertier et A.-M. Kermarrec : On the deterministic tracking of moving objects with a binary sensor network. *In Proceedings of the 4th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 46–59, 2008b.

Y. Busnel, L. Querzoni, R. Baldoni, M. Bertier et A.-M. Kermarrec : Analysis of deterministic tracking of multiple objects using a binary sensor network. *ACM Transactions on Sensor Networks*, 8(1):8, 2011b.

Y. Busnel, P. Serrano-Alvarado et P. Lamarre : Trust your social network according to satisfaction, reputation and privacy. *In Proceedings of the 3rd International Workshop on Reliability, Availability, and Security (WRAS)*, 2010.

Y. Mocquard, E. Anceaume, J. Aspnes, Y. Busnel et B. Sericola : Counting with population protocols. *In Proceedings of the 14th IEEE International Symposium on Network Computing and Applications (NCA)*, pages 35–42, 2015.

N. Rivetti, E. Anceaume, Y. Busnel, L. Querzoni et B. Sericola : Proactive Online Scheduling for Shuffle Grouping in Distributed Stream Processing Systems. *In Proceedings of the 13th ACM/IFIP/USENIX International Conference on Middleware (Middleware)*, 2016a.

N. Rivetti, Y. Busnel et A. Mostéfaoui : Efficiently summarizing data streams over sliding windows. *In Proceedings of the 14th IEEE International Symposium on Network Computing and Applications (NCA)*, pages 151–158, 2015a.

N. Rivetti, Y. Busnel et L. Querzoni : Load-aware shedding in stream processing systems. *In Proceedings of the 10th ACM International Conference on Distributed Event-Based Systems (DEBS)*, 2016b.

N. Rivetti, L. Querzoni, E. Anceaume, Y. Busnel et B. Sericola : Efficient key grouping for near-optimal load balancing in stream processing systems. *In Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems (DEBS)*, pages 80–91, 2015b.

A. Samba, Y. Busnel, A. Blanc, P. Dooze et G. Simon : Throughput prediction in cellular networks : Experiments and preliminary results. *In Actes de la 1ères Rencontres Francophones sur la Conception de Protocoles, l'Évaluation de Performance et l'Expérimentation des Réseaux de Communication (CoRes)*, 2016.

Autres publications non citées dans le manuscrit

D. R. Avresky et Y. Busnel, éditeurs. *Proceedings of the 14th IEEE International Symposium on Network Computing and Applications (NCA)*, 2015. IEEE. ISBN 978-1-5090-1849-9.

Y. Busnel : *Auto-organisation et collaboration pour réseaux de capteurs.* Editions Universitaires Européennes, 2010. ISBN 978-6-1315-1456-2.

Y. Busnel, M. Bertier, E. Fleury et A.-M. Kermarrec : GCP : gossip-based code propagation for large-scale mobile wireless sensor networks. *In Proceedings of the 1st International Conference on Autonomic Computing and Communication Systems (Autonomics)*, page 11, 2007.

Y. Busnel, M. Bertier et A.-M. Kermarrec : SOLIST or How To Look For a Needle in a Haystack ? *In Proceedings of the 4th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2008a.

Y. Busnel et I. Gashi, éditeurs. *EDCC 2015 - Fast Abstracts & Student Forum Proceedings*, volume abs/1602.06683 de *CoRR*, 2015. ArXiv.

Y. Busnel et A.-M. Kermarrec : PROXSEM : interest-based proximity measure to improve search efficiency in P2P systems. *In Proceedings of the 4th European Conference on Universal Multiservice Networks (ECUMN)*, pages 62–74, 2007.

H. Weatherspoon, H. Miranda, K. Iwanicki, A. Ghodsi et Y. Busnel : Gossiping over storage systems is practical. *ACM SIGOPS Operating Systems Review*, 41(5):75–81, 2007.

Travaux connexes

D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul et S. Zdonik : Aurora : a new model and architecture for data stream management. *The International Journal on Very Large Data Bases (VLDB)*, 12(2), 2003.

S. M. Ali et S. D. Silvey : General Class of Coefficients of Divergence of One Distribution from Another. *Journal of the Royal Statistical Society. Series B (Methodological)*, 28(1):131–142, 1966.

N. Alon, P. B. Gibbons, Y. Matias et M. Szegedy : Tracking join and self-join sizes in limited storage. *In Proceedings of the 18th ACM Symposium on Principles of Database Systems (PODS)*, pages 10–20, 1999.

N. Alon, Y. Matias et M. Szegedy : The space complexity of approximating the frequency moments. *In Proceedings of the 28th ACM Symposium on Theory of computing (STOC)*, pages 20–29, 1996.

L. Amini, N. Jain, A. Sehgal, J. Silber et O. Verscheure : Adaptive control of extreme-scale stream processing systems. *In Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2006.

E. Anceaume, R. Ludinard, B. Sericola et F. Tronel : Modeling and evaluating targeted attacks in large sacle dynamic systems. *In Proceedings of the 41rst IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2011b.

C. Arackaparambil, J. Brody et A. Chakrabarti : Functional monitoring without monotonicity. *In Proceedings of the 36th ACM International Colloquium on Automata, Languages and Programming : Part 1*, 2009.

A. Arasu et G. S. Manku : Approximate counts and quantiles over sliding windows. *In Proceedings of the 23rd ACM Symposium on Principles of Database Systems (PODS)*, 2004.

R. H. Arpaci-Dusseau, E. Anderson, N. Treuhaft, D. E. Culler, J. M. Hellerstein, D. Patterson et K. Yelick : Cluster i/o with river : Making the fast case common. *In Proceedings of the 6th Workshop on Input/Output in Parallel and Distributed Systems (IOPADS)*, 1999.

B. Awerbuch et C. Scheideler : Group Spreading : A Protocol for Provably Secure Distributed Name Service. *In Proceedings of the 31rst International Colloquium on Automata, Languages and Programming (ICALP)*, pages 183–195, 2004.

B. Awerbuch et C. Scheideler : Towards a Scalable and Robust Overlay Network. *In Proceedings of the 6th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2007.

B. Babcock, M. Datar et R. Motwani : Load shedding for aggregation queries over data streams. *In Proceedings of the 20th IEEE International Conference on Data Engineering (ICDE)*, 2004.

Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar et L. Trevisan : Counting distinct elements in a data stream. *In Proceedings of the 6th International Workshop on Randomization and Approximation Techniques (RANDOM)*, pages 1–10. Springer-Verlag, 2002.

M. Basseville et J.-F. Cardoso : On entropies, divergences, and mean values. *In Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, 1995.

A. Berlinet et C. Thomas-Agnan : *Reproducing Kernel Hilbert Spaces in Probability and Statistics*. Springer-Verlag, 2004.

A. Bhattacharyya : On a measure of divergence between two statistical populations defined by their probability distributions. *Bulletin of the Calcutta Mathematical Society*, 35:99–109, 1943.

E. Bortnikov, M. Gurevich, I. Keidar, G. Kliot et A. Shraer : Brahms : Byzantine Resilient Random Membership Sampling. *Computer Networks*, 53:2340–2359, 2009.

L. Bregman : The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7(3):200–217, 1967.

C. Brunner et D. Flore. : Generation of pathloss and interference maps as son enabler in deployed umts networks. *In Proceedings of the 69th IEEE Vehicular Technology Conference (VTC Spring)*, 2009.

V. Cardellini, E. Casalicchio, M. Colajanni et P. S. Yu : The state of the art in locally distributed web-server systems. *ACM Computing Surveys*, 34(2), 2002.

J. L. Carter et M. N. Wegman : Universal classes of hash functions. *Journal of Computer and System Sciences*, 18:143–154, 1979.

M. Castro et B. Liskov : Practical byzantine fault tolerance and proactive recovery. *ACM Transaction on Computer System*, 20(4):398–461, 2002.

A. Chakrabarti, K. Do Ba et S. Muthukrishnan : Estimating entropy and entropy norm on data streams. *In Proceedings of the 23rd International Symposium on Theoretical Aspects of Computer Science (STACS)*. Springer, 2006.

A. Chakrabarti, G. Cormode et A. McGregor : A near-optimal algorithm for computing the entropy of a stream. *In Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 328–335, 2007.

H.-L. Chan, T.-W. Lam, L.-K. Lee et H.-F. Ting : Continuous monitoring of distributed data streams over a time-based sliding window. *Algorithmica*, 62(3-4):1088–1111, 2012.

V. Chandola, A. Banerjee et V. Kumar : Anomaly detection : A survey. *ACM Computing Surveys*, 41(3):1–58, 2009.

T. D. Chandra et S. Toueg : Unreliable failure detectors for reliable distributed systems. *Journal of ACM*, 43(2):225–267, 1996.

M. Charikar, K. Chen et M. Farach-Colton : Finding frequent items in data streams. *Elsevier Theoretical Computer Science*, 312(1):3–15, 2004.

Guillaume Chauvet : On a characterization of ordered pivotal sampling. *Bernoulli*, 18(4):1320–1340, 2012.

R. Chen, J. Shi, Y. Chen et H. Chen : Powerlyra : Differentiated graph computation and partitioning on skewed graphs. *In Proceedings of the 10th European Conference on Computer Systems (EuroSys)*, pages 1 :1–1 :15, 2015.

G. Cormode : Continuous distributed monitoring : A short survey. *In Proceedings of the 1st International Workshop on Algorithms and Models for Distributed Event Processing (AlMoDEP)*, 2011.

G. Cormode et S. Muthukrishnan : An improved data stream summary : the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

G. Cormode, S. Muthukrishnan et K. Yi : Algorithms for distributed functional monitoring. *In Proceedings of the 19th ACM-SIAM Symposium On Discrete Algorithms (SODA)*, 2008.

G. Cormode, S. Muthukrishnan et K. Yi : Algorithms for distributed functional monitoring. *ACM Transaction on Algorithms*, 7(2):21 :1–21 :20, 2011.

G. Cormode et K. Yi : Tracking distributed aggregates over time-based sliding windows. *In Proceedings of the 24th International Conference on Scientific and Statistical Database Management (SSDBM)*, 2012.

Graham Cormode et Minos Garofalakis : Sketching probabilistic data streams. *In Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data (MOD)*, pages 281–292, 2007.

T.M. Cover et J.A. Thomas : *Elements of Information Theory*. Wiley New York, 1991.

Imre Csiszár : Eine informationstheoretische ungleichung und ihre anwendung auf den beweis der ergodizitat von markoffschen ketten. *Magyar. Tud. Akad. Mat. Kutató Int. Közl*, 8:85–108, 1963.

M. Datar, A. Gionis, P. Indyk et R. Motwani : Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.

J. Dean et S. Ghemawat : Mapreduce : Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

E.D. Demaine, A. Lopez-Ortiz et J.I. Munro : Frequent estimation of Internet packet streams with limited space. *In Proceedings of the 11th European Symposium on Algorithms (ESA)*, 2003.

X. Dimitropoulos, M. Stoecklin, P. Hurley et A. Kind : The eternal sunshine of the sketch data structure. *Computer Networks*, 52(17), 2008.

D. Dolev, E. N. Hoch et R. Van Renesse : Self-stabilizing and Byzantine-tolerant Overlay Network. *In Proceedings of the 11th International conference on Principles of Distributed Systems (OPODIS)*, pages 343–357. Springer-Verlag, 2007.

J. Douceur : The Sybil Attack. *In Proceedings of the 1rst International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 251–260, 2002.

P. Druschel et A. Rowstron : Past : A large-scale, persistent peer-to-peer storage utility. *In Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS)*, 2001.

M. Durand et P. Flajolet : Log-log counting of large cardinalities. *In Proceedings of the 11th European Symposium on Algorithms (ESA)*, 2003.

C. Estand et G. Varghese : New directions in traffic measurement and accounting : Focusing on the elephants, ignoring the mice. *ACM Transactions on Computer Systems*, 21(3):270–313, 2003.

S. Feng et E. Seidel : Self-organizing networks (son) in 3gpp long term evolution. Rapport technique, Nomor Research GmbH, 2008.

P. Flajolet et G. Nigel Martin : Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985.

D. Frye et P. Southerington : Dormant Malware Attacks - What's Next ? *In RSA Conference*, 2012.

M. R. Garey et D. S. Johnson : *Computers and Intractability : A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., 1979. ISBN 0716710447.

B. Gedik : Partitioning functions for stateful data parallelism in stream processing. *The International Journal on Very Large Data Bases (VLDB)*, 23(4), 2014.

P. B. Gibbons et S. Tirthapura : Estimating simple functions on the union of data streams. *In Proceedings of the 13th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 281–291, 2001.

P. B. Gibbons et S. Tirthapura : Distributed streams algorithms for sliding windows. *Theory of Computing Systems*, 37(3):457–478, 2004.

P. B. Godfrey, S. Shenker et I. Stoica : Minimizing churn in distributed systems. *In Proceedings of the 29th ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2006.

L. Golab, D. DeHaan, E. D. Demaine, A. Lopez-Ortiz et J. I. Munro : Identifying frequent items in sliding windows over on-line packet streams. *In Proceedings of the 3rd ACM SIGCOMM Internet Measurement Conference (IMC)*, 2003.

D. M. Gordon : *Ant Encounters : Interaction Networks and Colony Behavior*. Princeton University Press, 2010. ISBN 9781400835447.

S. Guha, P. Indyk et A. Mcgregor : Sketching information divergences. *Machine Learning*, 72 (1-2):5–19, 2008. ISSN 0885-6125.

S. Guha, A. McGregor et S. Venkatasubramanian : Streaming and sublinear approximation of entropy and information distances. *In Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 733–742, 2006.

V. Gupta, M. Harchol-balter, K. Sigman et W. Whitt : Analysis of join-theshortest-queue routing for web server farms. *In Proceedings of the 25th IFIP WG 7.3 International Symposium on Computer Modeling, Measurement and Evaluation (PERFORMANCE)*, 2007.

Z. Haung, K. Yi et Q. Zhang : Randomized algorithms for tracking distributed count, frequencies and ranks. *In Proceedings of the 31st ACM Symposium on Principles of Database Systems (PODS)*, 2012.

Y. He, S. Barman et J. F. Naughton : On load shedding in complex event processing. *In Proceedings of the 17th International Conference on Database Theory (ICDT)*, 2014.

M. Hirzel, R. Soulé, S. Schneider, G. Gedik et R. Grimm : A catalog of stream processing optimizations. *ACM Computing Surveys*, 46(4), 2014.

Jean Jacquelin : Zéro puissance zéro. *Quadrature*, 66:34–36, 2007.

M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec et M. van Steen : Gossip-based Peer Sampling. *ACM Transaction on Computer System*, 25(3):Article 8, 2007.

G. P. Jesi, A. Montresor et M. van Steen : Secure Peer Sampling. *Elsevier Computer Networks*, 54(12):2086–2098, 2010.

S. Jin et D.S. Yeung : A covariance analysis model for ddos attack detection. *In Proceedings of the 4th IEEE International Conference on Communications (ICC)*, pages 1882–1886, 2004.

D. M. Kane, J. Nelson et D. P. Woodruff : An optimal algorithm for the distinct element problem. *In Proceedings of the 29th ACM Symposium on Principles of Database Systems (PODS)*, 2010.

V. Karamcheti, D. Geiger, Z. Kedem et S. Muthuskrishnan : Detecting malicious network traffic using inverse distribution of packet contents. *In Proceedings of the 1st Workshop on Mining Network Data (MineNet)*, 2005.

D. R. Karger et M. Ruhl : Simple Efficient Load Balancing Algorithms for Peer-to-Peer. *In Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2004.

R.M. Karp, S. Shenker et C.H. Papadimitriou : A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems (TODS)*, 28(1):51–55, 2003.

B. Krishnamurthy, S. Sen, Y. Zhang et Y. Chen : Sketch-based change detection : Methods, evaluation, and applications. *In Proceedings of the 3rd ACM SIGCOMM Internet Measurement Conference (IMC)*, pages 234–247, 2003.

S. Kullback et R. A. Leibler : On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

A. Lakhina, M. Crovella et C.Diot : Mining anomalies using traffic feature distributions. *In Proceedings of the 28th ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2005.

A. Lall, V. Sekar, M. Ogihara, J. Xu et H. Zhang : Data streaming algorithms for estimating entropy of network traffic. *In Proceedings of the 27th joint International conference on Measurement and modeling of computer systems (SIGMETRICS)*. ACM, 2006.

L. Lamport : The part-time parliament. *ACM Transaction on Computer System*, 16(2):133–169, 1998.

D. Liu, P. Ning et W. Du : Detecting Malicious Beacon Nodes for Secure Location Discovery in Wireless Sensor Networks. *In Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2005.

Z. Liu, B. Radunovic et M. Vojnovic : Continuous distributed counting for non-monotonic streams. *In Proceedings of the 31st ACM Symposium on Principles of Database Systems (PODS)*, 2012.

Q. Lv, P. Cao, E. Cohen, K. Li et S. Shenker : Search and Replication in Unstructured Peer-to-Peer Networks. *In Proceedings of the 15th International Conference on Supercomputing (ICS)*, pages 84–95, 2002.

C. Lynch : Big data : How do your data grow ? *Nature*, 455(7209):28–29, 2008.

N. Lynch : *Distributed Algorithms.* Morgan Kaufmann Publishers, 1996.

A. Manjhi, V. Shkapenyuk, K. Dhamdhere et C. Olston : Finding (recently) frequent items in distributed data streams. *In Proceedings of the 21st IEEE International Conference on Data Engineering (ICDE)*, 2005.

V. Mayer-Schönberger et K. Cukier : *Big data : A revolution that will transform how we live, work, and think.* Houghton Mifflin Harcourt, 2013.

A. Metwally, D. Agrawal et A. El Abbadi : Efficient computation of frequent and top-k elements in data streams. *In Proceedings of the 10th International Conference on Database Theory (ICDT)*, 2005.

J. Misra et D. Gries : Finding repeated elements. *Science of Computer Programming*, 2(2):143–152, 1982.

Tetsuzo Morimoto : Markov processes and the h-theorem. *Journal of the Physical Society of Japan*, 18(3):328–331, 1963.

S. Muthukrishnan : *Data Streams : Algorithms and Applications.* Now Publishers Inc., 2005.

M. A. U. Nasir, G. De Francisci Morales, D. Garcia Soriano, N. Kourtellis et M. Serafini : The Power of Both Choices : Practical Load Balancing for Distributed Stream Processing Engines. *In Proceedings of the 31st IEEE International Conference on Data Engineering (ICDE)*, 2015.

O. Papapetrou, M. N. Garofalakis et A. Deligiannakis : Sketch-based querying of distributed sliding-window data streams. *VLDB Endowment – Proceedings of the 38st International Conference on Very Large Data Bases (VLDB)*, 5(10):992–1003, 2012.

F. Petroni, L. Querzoni, K. Daudjee, S. Kamali et G. Iacoboni : HDRF : Stream-Based Partitioning for Power-Law Graphs. *In Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 243–252, 2015.

Fabio Petroni : *Mining at scale with latent factor models for matrix completion.* Thèse de doctorat, Sapienza University of Rome, 2015.

V. Poosala et Y. E. Ioannidis : Estimation of query-result distribution and its application in parallel-join load balancing. *In Proceedings of the 22th International Conference on Very Large Data Bases (VLDB)*, pages 448–459, 1996.

W. Raghupathi et V. Raghupathi : Big data analytics in healthcare : promise and potential. *Health Information Science and Systems*, 2(1):1–10, 2014.

D. Raptis, N. Tselios et N. Avouris : Context-based design of mobile applications for museums : a survey of existing practices. *In Proceedings of the 7th ACM International Conference on Human Computer Interaction with Mobile Devices & Services (MobileHCI)*, pages 153–160, 2005.

S. Ratnasamy, P. Francis, M. Handley, R. Karp et S. Shenker : A scalable content-addressable network. *In Proceedings of the 24th ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2001.

A. I. T. Rowstron et P. Druschel : Pastry : Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *In Proceedings of the 3rd IFIP/ACM International Conference on Distributed Systems Platforms*, pages 329–350, 2001.

O. Salem, S. Vaton et A. Gravey : A scalable, efficient and informative approach for anomaly-based intrusion detection systems : theory and practice. *International Journal of Network Management*, 20(5):271–293, 2010.

M. A. Sharaf, P. K. Chrysanthis, A. Labrinidis et K. Pruhs : Algorithms and metrics for processing multiple heterogeneous continuous queries. *ACM Transactions on Database Systems*, 33(1):5 :1–5 :44, 2008.

A. Singh, T.-W. Ngan, P. Druschel et D. S. Wallach : Eclipse Attacks on Overlay Networks : Threats and Defenses. *In Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM)*, 2006.

M. Starnini, A. Baronchelli et R. Pastor-Satorras : Modeling human dynamics of face-to-face interaction networks. *Physical Review Letters*, 110:168701–168705, 2013.

I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek et H. Balakrishnan : Chord : a Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *IEEE/ACM Transaction on Networks*, 11(1):17–32, 2003.

A. Swami et K. B. Schiefer : On the estimation of join result sizes. *In Proceedings of the 4th International Conference on Extending Database Technology : Advances in Database Technology (EDBT)*, pages 287–300, 1994.

N. Tatbul, U. Çetintemel et S. Zdonik : Staying fit : Efficient load shedding techniques for distributed stream processing. *In Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB)*, 2007.

N. Tatbul, U. Çetintemel, S. Zdonik, M. Cherniack et M. Stonebraker : Load shedding in a data stream manager. *In Proceedings of the 29th International Conference on Very Large Data Bases (VLDB)*, 2003.

L. Toubiana et M. Cuggia : Big data and smart health strategies : findings from the health information systems perspective. *Yearbook of Medical Informatics*, 9(1):125–127, 2014.

L. Vaquero, F. Cuadrado, D. Logothetis et C. Martella : Adaptive partitioning for large-scale dynamic graphs. *In Proceedings of the 4th Annual Symposium on Cloud Computing (SOCC)*, pages 35 :1–35 :2, 2013.

D. Vengerov, A. C. Menck, M. Zait et S. P. Chakkappen : Join size estimation subject to filter conditions. *VLDB Endowment – Proceedings of the 41st International Conference on Very Large Data Bases (VLDB)*, 8(12):1530–1541, 2015.

J. Viard, M. Latapy et C. Magnien : Revealing contact patterns among high-school students using maximal cliques in link streams. *In Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 1517–1522, 2015.

C. Xie, L. Yan, W.-J. Li et Z. Zhang : Distributed power-law graph computing : Theoretical and empirical analysis. *In Proceedings of the 27th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 1673–1681, 2014.

N. Xu, L. Chen et B. Cui : LogGP : A Log-based Dynamic Graph Partitioning Method. *VLDB Endowment – Proceedings of the 40th International Conference on Very Large Data Bases (VLDB)*, 7(14):1917–1928, 2014.

K. Yi et Q. Zhang : Optimal tracking of distributed heavy hitters and quantiles. *Algorithmica*, 65:206–223, 2013.

J. Yuan et K. Mills : Monitoring the macroscopic effect of DDoS flooding attacks. *IEEE Transactions on Dependable and Secure Computing*, 2(4), 2005.

L. Zhang et Y. Guan : Variance estimation over sliding windows. *In Proceedings of the 26h ACM Symposium on Principles of Database Systems (PODS)*, 2007.

B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph et J. D. Kubiatowicz : Tapestry : a resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, 2004.

Q. Zhao, A. Lall, M. Ogihara et J. Xu : Global iceberg detection over distributed streams. *In Proceedings of the 26th IEEE International Conference on Data Engineering (ICDE)*, 2010.

Q. Zhao, M. Ogihara, H. Wang et J. Xu : Finding global icebergs over distributed data sets. *In Proceedings of the 25th ACM Symposium on Principles of Database Systems (PODS)*, 2006.

Sitographie

ACM DEBS 2013 Grand Challenge : http://www.orgs.ttu.edu/debs2013/index.php?goto=cfchallengedetails. The 7th ACM International Conference on Distributed Event-Based Systems (DEBS), 2013. En ligne ; accédé en Juin 2016.

Apache Storm : http://storm.apache.org. The Apache Software Foundation, 2014. En ligne ; accédé en Juin 2016.

BigClin : BIG data analytics for unstructured CLINical data. http://www.bigclin.cominlabs.ueb.eu/ – ANR LabEx CominLabs – ANR-10-LABX-07-01, 2016. En ligne ; accédé en Juin 2016.

DeSceNt : Plug-based decentralized social network. http://www.descent.cominlabs.ueb.eu/ – ANR LabEx CominLabs – ANR-10-LABX-07-01, 2014. En ligne ; accédé en Juin 2016.

Gartner Inc. : Hype Cycles. http://www.gartner.com/technology/research/hype-cycles/, 2015. En ligne ; accédé en Juin 2016.

INSHARE : Contribution pour l'intégration et le partage des données de santé pour une recherche collaborative. http://healthdatascience.org/projets/inshare/ – Agence Nationale pour la Recherche – ANR-15-CE19-0024, 2016. En ligne ; accédé en Juin 2016.

Internet Traffic Archive : http://ita.ee.lbl.gov/html/traces.html. Lawrence Berkeley National Laboratory, 2008. En ligne ; accédé en Juin 2016.

Microsoft Azure : https://azure.microsoft.com. Microsoft Corporation, 2014. En ligne ; accédé en Juin 2016.

Microsoft Azure for Research Award : Robust and efficient analytics of massive input streams. https://www.microsoft.com/en-us/research/academic-program/microsoft-azure-for-research/, 2015. En ligne ; accédé en Juin 2016.

Microsoft Azure for Research Award : Robust and efficient analytics of large scale distributed streams. https://www.microsoft.com/en-us/research/academic-program/microsoft-azure-for-research/, 2016. En ligne ; accédé en Juin 2016.

National Institutes of Health : Data Science at NIH : What is Big Data ? https://datascience.nih.gov/bd2k/about/what, 2015. En ligne ; accédé en Juin 2016.

Social Network Benchmark : http://ldbcouncil.org/benchmarks/snb. Linked Data Benchmark Council, 2015. En ligne ; accédé en Juin 2016.

SocioPlug : Cloud social sur des réseaux de plugs, pour un accès à l'information symétrique et respectueux de la vie privée. http://socioplug.univ-nantes.fr/ – Agence Nationale pour la Recherche – ANR-13-INFR-003, 2013. En ligne ; accédé en Juin 2016.

Spark Streaming : https://spark.apache.org/streaming/. The Apache Software Foundation, 2013. En ligne ; accédé en Juin 2016.

THE CAIDA UCSD "ANONYMIZED INTERNET TRACES 2008" DATASET : http://www.caida.org/data/passive/passive/2008/dataset.xml. Cooperative Association for Internet Data Analysis, 2008. En ligne ; accédé en Juin 2016.

THE CAIDA UCSD "DDOS ATTACK 2007" DATASET : http://www.caida.org/data/passive/ddos-20070804/dataset.xml. Cooperative Association for Internet Data Analysis, 2010. En ligne ; accédé en Juin 2016.

TWITTER : https://twitter.com. Twitter Inc., 2006. En ligne ; accédé en Juin 2016.

Table des figures

Liste des tableaux

Liste des Algorithmes