Table des matières

Liste des figures

Liste des tableaux

Liste des codes et pseudo-codes

Remerciements

Je voudrais remercier mon directeur de recherche, le Dr Thierry Badard, de m'avoir donné la chance de travailler au sein de son groupe de recherche et pour avoir eu confiance en moi pendant mes études de deuxième cycle.

Je voudrais aussi remercier Caroline Gagné, ma conjointe, qui m'a conseillé et écouté et avec qui j'ai eu de nombreuses discussions m'ayant permis de mettre des mots sur mes idées.

Je remercie aussi Patrick Lauzière, ancien collègue et surtout ami, pour son aide et pour nos discussions intéressantes.

Je tiens à remercier le département d'informatique du Cégep de l'Outaouais, particulièrement François Pagé et Faouzi Bouguerra, pour m'avoir laissé utiliser les équipements du laboratoire d'informatique pour effectuer mes tests. Je remercie particulièrement Faouzi pour son aide durant la configuration du réseau de communication.

Finalement, je tiens à remercier, par-dessus tout, ma muse et mon inspiration, ma meilleure amie depuis les 19 dernières années. Elle qui est à mes côtés chaque jour et qui sait par sa présence me conseiller, me détendre et me rattacher à l'essentiel. À elle, je veux dédier ce travail. Merci, petite chatte Mika.

Chapitre 1 Introduction

1.1 Contexte

Depuis quelques années, nous assistons à une augmentation du volume d'information créé. Cette quantité d'information provient de l'utilisation des réseaux sociaux, de l'accroissement de l'usage des téléphones intelligents et des tablettes connectés à Internet et de leur capteur intégré. Elle provient aussi des transactions commerciales que nous faisons, des réseaux de capteurs industriels, de vidéos, de caméra de surveillance, de journaux (logs) qui suivent nos comportements sur Internet et sur nos appareils électroniques et, bientôt, d'une panoplie d'objets connectés qui formera l'Internet des objets. Il faut aussi ajouter à ce flot les données provenant des balises géoréférencées WiFi et Bluetooth. Ces balises permettent de détecter les appareils électroniques et de faire leur suivi en milieu urbain. La majorité de ces informations sont produites en continu sous la forme de flux de données. En 2014, une recherche de l'International Data Corporation (IDC) rapportait que les objets connectés mobiles généraient 18 % des données de l'univers numérique et qu'en 2020, cette proportion serait de 27 %(International Data Corporation, 2014a, 2014b).

Traditionnellement, les jeux de données sont peu changeants, c'est-à-dire que l'information n'évolue pas avec le temps ou évolue lentement. On dit que ces données sont au repos, persistantes ou statiques. Une classe d'entité représentant les bornes-fontaines d'une municipalité est un exemple de données statiques.

De l'autre côté du spectre, nous avons des objets dont les changements d'état rapides requièrent des mises à jour fréquentes. Ces données sont dites dynamiques. Une classe d'entité représentant des objets mobiles et ayant des mises à jour de position et d'état à la seconde est considérée comme étant dynamique.

Ce mémoire fait référence aux données statiques comme étant des données traditionnelles. Bien qu'il y ait une gradation entre les données purement statiques et celles dynamiques, le terme dynamique sera réservé à celles dont la mise à jour est rapide et fait l'objet d'une surveillance ou d'un traitement en continu.

Pour une application traitant des données de façon dynamique, les données récentes auront plus de valeur que les données anciennes. Par exemple, une application de surveillance d'objets mobiles voudra avertir le plus tôt possible son utilisateur de la détection d'un patron de déplacement suspect. Plus le temps pour obtenir et traiter la donnée est éloigné du temps où celle-ci a été produite, plus la valeur de l'information obtenue sera de moindre intérêt. Dans un processus décisionnel, il devient donc avantageux de les traiter le plus tôt possible (Laney, 2001; Golab et Özsu, 2003; Stonebraker *et al.*, 2005; Hurwitz *et al.*, 2013).

La valeur intrinsèque de ces flux de données, qui sont produits sans arrêt, peut s'avérer somme toute limitée. Toutefois, lorsqu'ils sont combinés entre eux ou à des données traditionnelles, ils peuvent servir à comprendre

des phénomènes et être utilisés pour offrir de nouvelles perspectives de traitements et d'analyses spatiales pour de nouvelles applications utilisant des flux géospatiaux à grande échelle (Kazemitabar *et al.*, 2011).

De plus, cette information est souvent géoréférencée. Par exemple, les messages d'un utilisateur de Twitter peuvent être géoréférencés à l'aide du GPS de son cellulaire, un capteur connecté à Internet peut transmettre aussi bien le résultat de son analyse que sa position et le journal des transactions d'une carte de crédit peut contenir des éléments spatiaux comme l'endroit de la transaction. Finalement, les véhicules intelligents et autonomes seront à même de fournir une masse d'information géoréférencée sous la forme de flux (Kazemitabar *et al.*, 2011; Wang et Yuan, 2014).

Ces flux géoréférencés deviennent donc des flux spatiaux qui nous parviennent avec de très grands débits et volumes. Il devient donc impossible de traiter ces flux dans un temps raisonnable à l'aide de méthodes dites conventionnelles (Kazemitabar *et al.*, 2011; Shekhar *et al.*, 2014). L'utilisation de techniques de traitement de données massives (Big Data) devient donc une solution pour tirer avantage du dynamisme de cette information ainsi que de son aspect changeant et volatil. Dans le domaine de la géomatique, cela signifie de pouvoir effectuer des opérations spatiales en temps réel utilisant ces flux géoréférencés.

1.2 Notions liées aux traitements spatiaux de flux massifs géoréférencés

Cette section aborde les notions importantes liées aux traitements spatiaux de flux massifs géoréférencés. Dans un premier temps, la notion de flux de données ainsi que les techniques de traitement leur étant propres sont présentées. Nous poursuivons avec la présentation du concept de données massives et les concepts derrière les techniques de traitement de ces données. Finalement, un inventaire des différents traitements spatiaux de base est présenté.

1.2.1 Les flux

Les données peuvent être classifiées en deux grandes catégories, soit 1) les données persistantes et 2) les données sous forme de flux. Les données dites persistantes sont celles qui nous sont les plus familières. Elles sont considérées comme « constantes » dans le temps ou ayant un faible taux de changement. Par exemple, elles peuvent représenter un réseau routier, une couche écoforestière ou une répartition démographique. Ces jeux de données persistantes ne sont en fait qu'une vue instantanée sur une période de temps donné. Le réseau routier est en évolution, le jeu de données démographiques sera mis à jour lors du prochain recensement et il en va de même pour les couches écoforestières. Une des composantes importantes dans le choix des jeux de données est la date d'acquisition. Bien que ces données soient en constant changement, leur fréquence de

mise à jour est faible comparativement à leur fréquence d'utilisation. Ces données possèdent donc une valeur durable dans le temps, sans quoi elles deviendraient obsolètes et inutilisées.

Selon les besoins, ces données seront stockées et structurées de manière à faciliter les traitements et les requêtes sur celles-ci. Par exemple, les données seront réparties dans une base de données relationnelle afin de minimiser la répétition d'information. Celles-ci seront aussi indexées, spatialement ou non, afin d'augmenter la rapidité des requêtes sur le jeu de données. Une autre particularité des données persistantes réside dans le fait que l'entièreté du jeu de données est disponible au moment du traitement. De plus, ces données seront sauvegardées dans le système que si l'état du système reste cohérent.

Dans l'autre catégorie, on retrouve les flux de données. Ces données sont rendues disponibles à travers une séquence de données transmises dans le temps. Comparativement aux données persistantes, la fréquence de mise à jour est très élevée. On considérera que ce flux d'information est potentiellement illimité en taille. Bien que la fréquence de mise à jour soit élevée, celle-ci n'est pas nécessairement constante. De plus, ces données ne nous parviennent pas nécessairement dans l'ordre. Par ailleurs, une des caractéristiques des flux de données est leur nature éphémère, pour laquelle il n'est pas toujours possible de les refaire jouer. Ainsi, les données ne sont transmises qu'une seule fois et si elles ne sont pas traitées ou stockées, elles sont perdues (Babcock *et al.*, 2002; Arasu *et al.*, 2004; Hershberger *et al.*, 2009; Gama, 2010; Bifet *et al.*, 2011).

Les applications utilisant des flux de données diffèrent de celles utilisant des données persistantes. Dans le cas des données persistantes, l'application aura accès au jeu de données entier et ses requêtes seront limitées à l'étendue temporelle de validité du jeu de données. Les résultats seront statiques dans le temps, c'est-à-dire qu'une requête faite à deux moments différents retournera le même résultat.

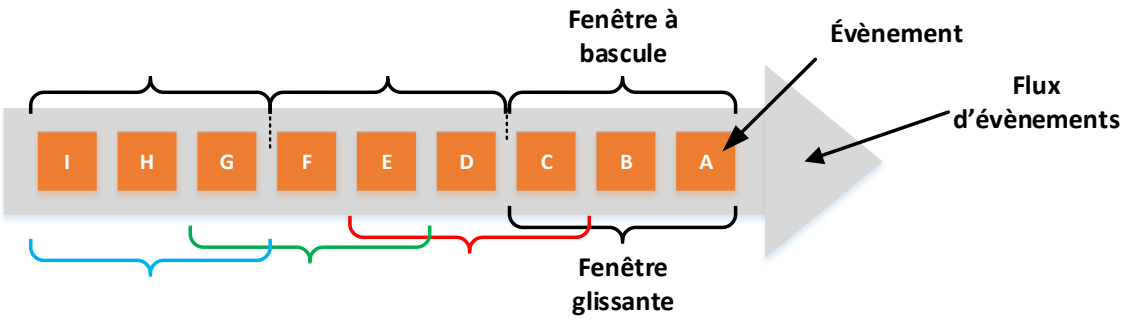
La composante temporelle est quant à elle inhérente aux données transmises en flux. Comme elles sont mises à jour fréquemment, leurs valeurs et les résultats d'analyse de celles-ci diminuent dans le temps. Dès lors, il s'avérera nécessaire de traiter ces informations en temps réel (Stonebraker *et al.*, 2005; Hurwitz *et al.*, 2013).

L'utilisation de systèmes de gestion de bases de données (SGBD) n'étant pas adaptée aux flux de données, des systèmes de gestion de flux de données (SGFD) ont été proposés pour gérer l'arrivée et le traitement rapide de ces données. Ces SGFD permettent aussi de répondre aux requêtes en continu, qui sont intrinsèques à ce type de données (Babcock *et al.*, 2002). Ces applications produisent elles-mêmes des résultats en continu sous la forme d'un flux. Ces requêtes en continu doivent pouvoir intégrer tant les données sous forme de flux que celles provenant de jeux de données persistantes (Huang et Zhang, 2008; Galić *et al.*, 2014). Qui plus est, ces applications doivent aussi gérer les opérateurs dits bloquants (Golab et Özsu, 2003; Stonebraker *et al.*, 2005). Par exemple, trouver la moyenne, effectuer un tri ou même faire une opération de jointure nécessitent de

disposer de l'ensemble des données. Or, un flux est par définition illimité. Pour résoudre ce problème, certaines techniques, basées sur des accumulateurs, ont été développées. L'accumulation des données pendant une période de temps ou jusqu'à l'obtention d'une quantité d'information déterminée permet alors de traiter l'information sous la forme de fenêtre. Plusieurs types de fenêtres ont été imaginées, telles que les fenêtres à bascules (*tumbling window*) et les fenêtres glissantes (*sliding window*) (Babcock *et al.*, 2002; Gama, 2010; Bifet *et al.*, 2011).

Dans une fenêtre à bascule, une période de temps ou une quantité de données sont assignées, ce qui correspond à la dimension de la fenêtre. Dans le cas d'une période prédéfinie, toutes les données seront accumulées pendant cette période et seront traitées ensemble. Il n'y a pas de chevauchement de données dans une fenêtre à bascule.

Quant à elles, les fenêtres glissantes comportent, en plus de la dimension de la fenêtre, un paramètre de glissement. Celui-ci permet à une seconde fenêtre de partager une partie des données de la première avec pour conséquence un chevauchement des fenêtres. L'image suivante représente les fenêtres à bascule en haut et glissantes en bas.



Adapté de : https://ci.apache.org/projects/flink/flink-docs-release-1.3/concepts/programming-model.html

Figure 1 Fenêtre à bascule et fenêtre glissante

En plus des caractéristiques déjà mentionnées, tels les fréquences variables de réception des données, leur désordre d'arrivée potentiel, l'intégration de données persistantes dans le traitement de flux et la gestion des opérateurs bloquants, les applications traitant des flux de données doivent permettre d'analyser les données avec un faible temps de latence afin de ne pas causer un débordement qui mènerait à une dégradation des performances et une perte d'information (Babcock *et al.*, 2002; Golab et Özsu, 2003; Arasu *et al.*, 2004; Stonebraker *et al.*, 2005, p. 8; Gama, 2010; Bifet *et al.*, 2011).

Puisque ce mémoire s'intéresse particulièrement aux flux de données spatiales, il convient donc de définir ce qui les distingue des flux en général. Les données contenues dans un flux de données spatiales contiennent un ou des attributs permettant de géolocaliser l'entité (ou l'évènement). L'entité peut avoir un attribut spatial, c'est-à-dire une géométrie, ou un attribut non spatial qui, lorsque couplé à une autre classe d'entités, permettrait de la géolocaliser. Les entités d'un flux de données non spatiales ne seraient pas géolocalisables. Il va sans dire qu'un flux de données spatiales sera considéré comme tel s'il y a un intérêt de la part de l'application d'effectuer des traitements spatiaux sur celles-ci. De plus, ce sont les données qui sont spatiales et non le flux. Par ailleurs, dans l'angle d'analyse de ce mémoire, ce ne sont pas les données qui sont massives, mais plutôt le flux considérant qu'une grande quantité de données spatiales doit être traitée. On parlera donc de flux massif de données spatiales et non de flux spatiaux de données massives. Il existe plusieurs façons d'exprimer la composante spatiale dans un flux. Par exemple, le flux pourrait être sous la forme d'un GeoJSON, d'un GeoRSS incluant une codification GML, sérialisé de façon binaire sous la forme d'un Avro ou tout simplement sous la forme d'un CSV (*comma separated value*) ayant des attributs latitude et longitude et une géométrie sous la forme d'un WKT (*Well Know Text*).

1.2.2 Les données massives (Big data) et leurs traitements

Ce sujet est abordé en deux sous-sections. Premièrement, les données massives (Big Data) sont présentées afin de définir ce qu'elles sont et les défis qu'elles représentent. Par la suite, les techniques et les concepts liés au traitement de ce type de données sont expliqués afin de bien saisir ce nouveau paradigme à l'intérieur duquel les traitements spatiaux doivent s'arrimer.

1.2.2.1 Les données massives (Big Data)

Le désir d'accumulation d'information n'est pas récent. Depuis toujours, les humains ont voulu préserver l'information. Que ce soit par la tradition orale, la représentation des scènes de vie sur les murs des cavernes, par l'invention de l'écriture et de la photographie, les humains ont toujours voulu consigner et préserver leurs informations. Avec les technologies numériques et les moyens de communication, la quantité d'information a augmenté rapidement. Aujourd'hui, les données provenant des appareils mobiles, des réseaux sociaux, des appareils connectés et des capteurs spécialisés fournissent une quantité énorme d'information. On estime que ces données représentaient 4,4 zettaoctets en 2013. On estime que la dimension du monde numérique sera de 44 zettaoctets en 2020 (International Data Corporation, 2014a; OECD, 2015).

D'autre part, le faible coût de stockage et les capacités de calcul grandissantes des ordinateurs ont permis la rétention et le traitement de ce type d'information (OECD, 2015).

L'historien des sciences, George Dyson, définissait les données massives en ces termes : « Le Big Data est ce qui est arrivé quand le coût pour conserver l'information est devenu inférieur à celui de prendre la décision de s'en débarrasser » (traduction libre) (Dyson, 2013).

Dans Shekhar *et al.* (2014), l'auteur donne la définition suivante des données spatiales massives : « La taille, la variété et la fréquence de mise à jour des données mobiles dépassent largement les capacités de l'informatique spatiale traditionnelle et des bases de données spatiales couramment utilisées pour l'apprentissage, la gestion et les traitements. Ces données sont connues sous le nom de données spatiales massives » (traduction libre).

Une autre façon de définir les données massives serait de dire que c'est un jeu de données ou une information qui ne peuvent être stockés et traités par des outils traditionnels qui permettraient d'obtenir une réponse adéquate dans un temps acceptable pour l'usage qui en est fait (Dumoulin, 2014; Siddalingaiah; 2014; Wang et Yuan, 2014). C'est cette définition que nous allons retenir pour ce mémoire.

Les défis des données massives sont représentés par plusieurs facettes souvent appelées les 3 V : volume, variété et vélocité (Laney, 2001; Wang et Yuan, 2014; Lee et Kang, 2015; Li *et al.*, 2015). Le volume de données représente un défi de stockage et de traitement. En effet, certains jeux de donnée sont si volumineux qu'ils ne peuvent pas être stockés dans une seule unité de stockage. De plus, le temps de traitement de ce type de jeux de données serait trop élevé s'il était fait à l'aide d'un seul ordinateur.

La variété des jeux de données est un autre défi, certains jeux pouvant se présenter sous une structure (ex. base de données relationnelle ou fichier csv) tandis que d'autres seront non structurés (ex. courriel, texte, photo ou vidéo). Ces différents jeux de données devront être mis en relation entre eux pour en extraire de l'information ayant de la valeur.

Une autre facette est la vélocité ou la fréquence à laquelle ces données sont mises à jour. Par exemple, la fréquence de mise à jour des données rend « obsolètes » les valeurs plus anciennes pour une application devant prendre des décisions rapidement. Il devient donc nécessaire de pouvoir traiter celles-ci rapidement pour fournir un résultat dans un temps opportun. La vélocité étant liée au flux, c'est donc cet aspect qui sera abordé dans ce mémoire.

Finalement, la valeur ou la véracité des données devient importante lorsqu'on fait face à une multitude de données. Que ce soit selon la provenance de celles-ci (ex. information ayant potentiellement moins de véracité en provenance des réseaux sociaux ou détections de capteurs défaillants) ou selon le but recherché dans l'analyse des données (ex. l'information est-elle valide pour cette analyse?), on voudra alors valider et filtrer cette masse de données (Wang et Yuan, 2014).

1.2.2.2 Traitement des données massives (Big Data)

Comme nous l'avons vu dans la section précédente, le stockage et le traitement des données massives ne peuvent être effectués à l'aide d'un seul ordinateur ou à l'aide des technologies dites traditionnelles. La solution est le stockage et le traitement en parallèle. Plusieurs architectures théoriques permettant le traitement en parallèle ont été étudiées depuis le milieu des années 1980 et durant les années 1990. Parmi celles-ci figure le modèle PRAM (parallel random access machine), où un groupe de n processeurs partagent une mémoire unique. Une autre architecture est le maillage (mesh), où les processeurs sont connectés directement aux processeurs voisins (Aggarwal *et al.*, 1988; Boxer et Miller, 1988; Dehne et Sack, 1989; Dehne, 1994). Cependant, le modèle qui s'est imposé de nos jours est celui d'une grappe d'ordinateur. Ces ordinateurs ne partagent ni mémoire ni processeur, mais sont liés par un lien de communication (Dean et Ghemawat, 2008; Eldawy *et al.*, 2013; Eldawy et Mokbel, 2013; Hurwitz *et al.*, 2013; Siddalingaiah, 2014; Guller, 2015).

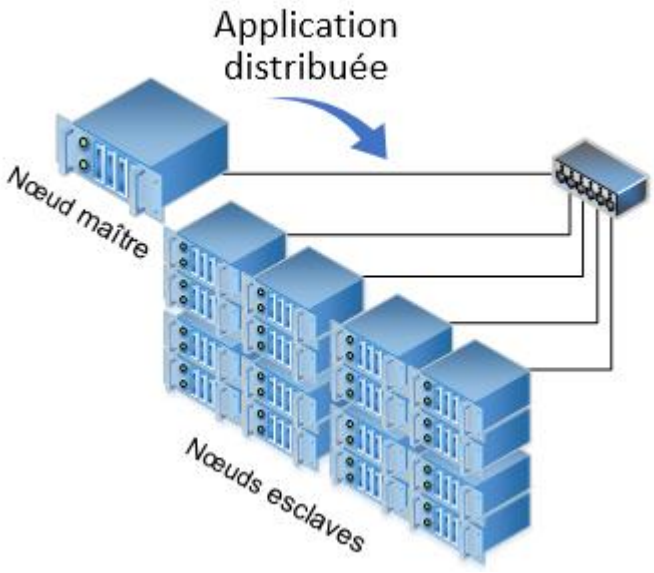


Figure 2 Architecture de traitement en parallèle

Cette architecture, telle que décrite par Google (Ghemawat *et al.*, 2003; Dean et Ghemawat, 2008), est composée de plusieurs ordinateurs contrôlés par un nœud maitre contrôlant la distribution de l'information (Google File System – GFS) et les traitements (Map Reduce). Dans ce type d'architecture, chacun des nœuds contient une partie de la masse d'information. Cette information est dupliquée pour assurer une certaine redondance en cas de défaillance des nœuds. Le concept de base du traitement, appelé Map-Reduce (Dean et Ghemawat, 2008), consiste à distribuer l'application sur les nœuds contenant l'information et non à aller questionner l'information sur les différents nœuds. Comme la quantité d'information est énorme, il ne serait pas possible de rapatrier celle-ci vers un serveur comme dans les applications traditionnelles. Pour résoudre cette problématique, la solution utilisée est la distribution de l'application sur les nœuds contenant l'information (Dean

et Ghemawat, 2008; Hurwitz *et al.*, 2013; Siddalingaiah;, 2014; Guller, 2015). Chacun des nœuds traite donc l'information qu'il contient et transmet le résultat partiel aux nœuds chargés de traiter ces résultats partiels en un résultat final.

Afin de bien comprendre la philosophie derrière ces traitements, il est important de saisir le concept Map Reduce. Dans ce patron d'architecture, chacun des nœuds effectue un traitement de l'information sur une base locale. Comme l'information est elle aussi distribuée, le nœud traitera l'information disponible localement. Une fonction "map" effectue un traitement qui génère un jeu de données de type clé-valeur. Ensuite, la phase appelée « shuffle » combine localement toutes les valeurs associées à une même clé. Finalement, toutes les valeurs intermédiaires associées à une même clé sont acheminées à un nœud pour y être traitées (fonction « reduce »). Il peut y avoir plusieurs phases de MapReduce en séquence pour obtenir le résultat final voulu.

La figure 3 ci-dessous illustre le processus MapReduce et représente un recensement. Chaque homme, femme et enfant de chaque village se voient attribuer une valeur de 1, ce qui représente la fonction « map ». Les recenseurs de chaque village classent les hommes, les femmes et les enfants séparément et comptent leurs sous-totaux respectifs, ce qui représente la fonction « shuffle ». Finalement, trois recenseurs ont chacun pour tâche de compiler les sous-totaux des hommes, des femmes ou des enfants provenant des recenseurs de chacun des villages. À la fin, le total est obtenu pour chaque groupe, ce qui correspond à la phase « reduce ».

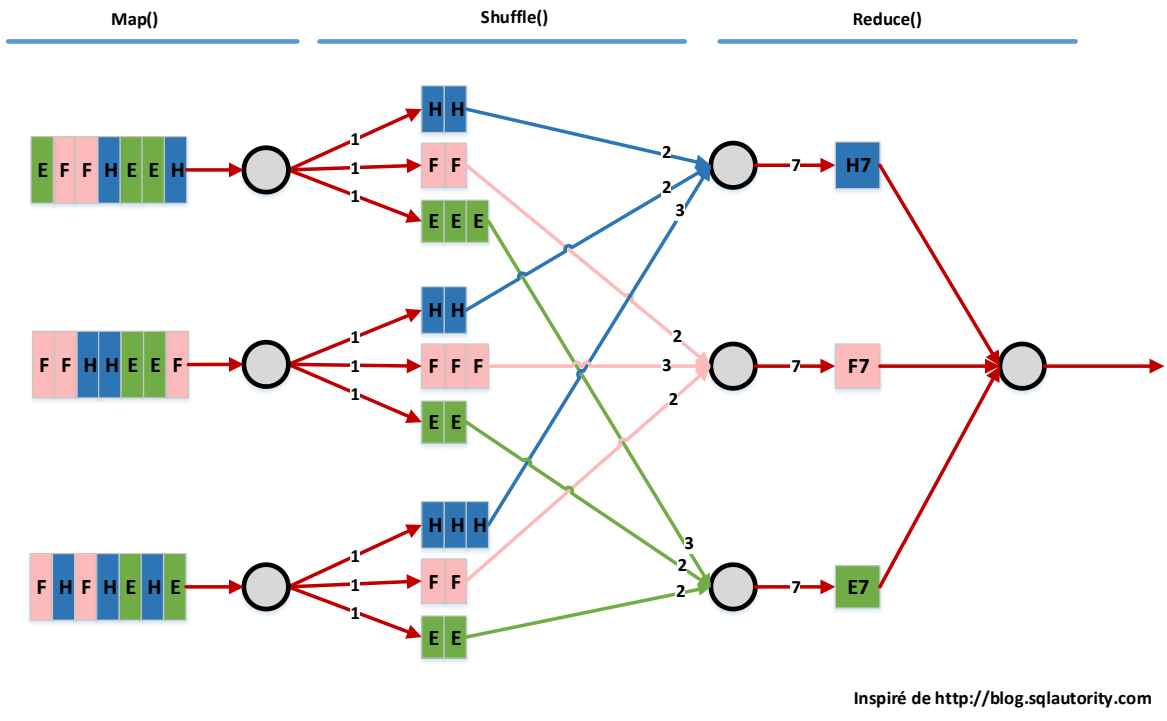


Figure 3 Étapes de traitement MapReduce

Depuis la publication de l'architecture Google File System (GFS) et du concept MapReduce, une myriade d'applications inspirées de ces concepts ont vu le jour dans l'univers des applications ouvertes. Les premiers étant Hadoop HDFS (stockage de données) et Hadoop MapReduce (traitement de données). Par la suite, plusieurs applications sont apparues pour faciliter le traitement de données sous forme de script, de base de données SQL et NoSQL. D'autres se sont spécialisées dans le chargement de données (ex. SQOOP), dans la présentation (Kibana) ou dans la gestion des opérations (Zookeeper). Aujourd'hui, toutes ces technologies forment un écosystème d'applications permettant de répondre aux défis que représentent les données massives. Initialement, les technologies de traitement de données étaient basées sur des traitements en lot de données persistantes. Au fil du temps, des besoins de traitement de flux de données sont apparus et des technologies ont été spécifiquement développées ou adaptées pour ce type de traitement (Storm, Spark, Flink) et pour la gestion des messages en flux (Kafka). Ce sont ces dernières technologies qui retiendront notre attention dans le cadre de ce mémoire.

Bien que la méthode Map-Reduce utilise une combinaison clé-valeur, ce ne sont pas tous les systèmes qui utilisent cette méthode. Cependant, cet exemple permet de définir une base de compréhension qui est souvent commune dans le traitement de l'information massive avec ce type d'architecture. En effet, si la combinaison clé-valeur n'est pas toujours présente de façon explicite, plusieurs systèmes prévoient une méthode permettant de grouper les informations sur la base d'un attribut. Comme nous l'avons vu précédemment, le traitement de flux implique aussi un traitement par fenêtre. Les applications permettant le traitement de flux possèdent donc des opérateurs servant à traiter les informations contenues dans une même fenêtre en les regroupant à l'aide d'un attribut commun.

Le traitement d'un flux de données prendra donc la forme d'une requête en continu et produira un résultat en continu. Ces requêtes pourront avoir pour but de filtrer les données, de les transformer, de détecter des évènements, d'alimenter un modèle d'apprentissage, d'alimenter des systèmes de prise de décision ou encore d'alimenter des jeux de données persistants (Golab et Özsu, 2003).

1.2.3 Les opérateurs spatiaux

Une des facettes de ce travail concerne bien entendu les traitements d'analyse spatiale. Les analyses spatiales sont des techniques permettant d'analyser des objets en tenant compte de leurs formes géométriques, de leurs positions et des relations qui existent entre eux. Il convient donc de répertorier ces traitements et les propriétés qui les caractérisent pour dégager les traits déterminants pour leur adaptation au traitement de flux massif

Une recherche sur la catégorisation des opérateurs spatiaux permet de trouver diverses études et documents caractérisant les propriétés des traitements spatiaux. Dans l'article de Egenhofer (1994), l'auteur classifie les

opérateurs selon leurs arités (opérandes), soit unaires ou binaires. Il définit un opérateur spatial unaire comme une fonction permettant d'obtenir les propriétés d'un attribut spatial. Pour les opérateurs unaires, l'auteur mentionne certains opérateurs permettant d'accéder aux propriétés spatiales pour obtenir leur dimension (0, 1, 2 ou 3), leurs frontières ou leurs intérieurs par exemple. Il mentionne aussi des opérateurs spatiaux unaires basés sur la dimension de l'entité géométrique. Dans cette catégorie, il inclut les opérateurs permettant d'obtenir la longueur, l'aire et le volume d'un objet ainsi que le polygone englobant (convex hull). Les opérateurs spatiaux binaires permettent quant à eux de calculer une valeur entre deux entités, telles que la distance et la direction. Finalement, il décrit les opérateurs binaires topologiques tels que les opérateurs « disjoint », « couvre », « est couvert par » et « est à l'intérieur » servant à déterminer les relations entre les entités.

Une autre approche de classification est mentionnée dans les articles de Clementini et Di Felice (1997, 2000). Les classements y sont définis comme topologiques, métriques et projectifs. Les opérateurs topologiques expriment la connectivité entre les objets. Ils définissent les intersections entre les objets et leur nature (ex. touchent, à l'intérieur, etc.). Les opérateurs projectifs représentent la concavité et la convexité d'une géométrie ainsi que d'autres relations spatiales comme « est à l'intérieur d'une région convexe » d'un objet (insideConvexHull). Finalement, les opérateurs métriques expriment les mesures ou les relations entre géométries comme la distance, la direction, la compacité ou la densité.

Quant à lui, l'Open Geospatial Consortium (OGC) classe les opérateurs géométriques dans plusieurs catégories, soit : les opérateurs de base, les opérateurs relationnels ou topologiques, les opérateurs d'analyse spatiale et les opérateurs métriques (Open Geospatial Consortium, 2011). Le tableau 1 présente un sommaire des différentes classes et de certains opérateurs définis par l'OGC. Les spécifications émises par l'OGC définissent également le type de donnée des opérandes et du résultat.

Tableau 1 Opérateurs tels que définis par l'OGC

Classe	Opérateur	Description
Opérateurs de base	SpatialReference	Retourne le système de référence spatiale de l'objet.
	Dimension	Retourne la dimension de l'objet.
	GeometryType	Retourne le type de l'objet géométrique.
	Envelope	Retourne le rectangle englobant.
	IsEmpty	Retourne vrai si l'objet est vide.
	Boundary	Retourne les limites de l'objet.
Opérateurs relationnels	Equals	Retourne vrai si deux objets sont spatialement égaux.
	Disjoint	Retourne vrai si deux objets sont spatialement disjoints.
	Intersects	Retourne vrai si deux objets sont spatialement intersectés.
	Touches	Retourne vrai si deux objets se touchent spatialement.
	Crosses	Retourne vrai si deux objets se croisent spatialement.
	Within	Retourne vrai si un objet est spatialement à l'intérieur de l'autre.
	Contains	Retourne vrai si un objet est spatialement contenu dans l'autre.
	Overlaps	Retourne vrai si les objets se superposent spatialement.
	Relate	Retourne vrai si les objets sont en relation spatiale selon une matrice d'intersection.
	LocateAlong	Retourne une portion de l'objet géométrique correspondant à la mesure spécifiée.
	LocateBetween	Retourne une portion de l'objet géométrique correspondant à la mesure spécifiée.
Opérateurs d'analyse spatial	Distance	Calcule la distance entre deux objets.
	Buffer	Retourne un objet contenant tous les points contenus dans une certaine distance maximale d'un objet.
	ConvexHull	Retourne un polygone convexe englobant d'un objet.
	Intersection	Retourne le résultat de l'intersection de deux objets.
	Union	Retourne le résultat de l'union de deux objets.
	Difference	Retourne le résultat de la différence entre deux objets.
	SymDifference	Retourne le résultat de la différence symétrique entre deux objets.
Opérateurs métriques	X, Y et Z	Coordonnées d'un point.
	Lenght	Longueur d'une ligne.
	Area	L'aire d'un polygone.
	Centroid	Le centroïde d'un polygone.

Les opérations mentionnées dans le tableau 1 sont considérées comme les opérateurs de base. Ceux-ci interviennent dans des méthodes d'analyse plus complexe tels l'interpolation, le calcul de cellule de Voronoï, le groupement (clustering) et les "k" plus proches voisins (knn), pour n'en citer que quelques-unes (Berg *et al.*, 2008; de Smith *et al.*, 2015). De plus, il faut aussi considérer le contexte temporel des opérateurs spatiaux. Les articles de Galić(2016a, 2016b) décrivent que l'application d'une composante temporelle à une entité peut servir à créer un nouveau type d'objet spatio-temporel et, en ce sens, de nouveaux opérateurs. Dans les études de Giannotti *et al.* (2011) et Galić (2016a), les auteurs mentionnent la recherche de nouveaux patrons spatio-temporels comme l'attroupement (flock), les rencontres (meet), les patrons répétitifs (periodic pattern) et les emplacements fréquents (frequent location). Bien que ces opérateurs complexes soient possiblement dérivés des opérateurs de base, leurs calculs présentent des contextes algorithmiques qui leur sont propres. Ces

algorithmiques renferment assurément des caractéristiques qui devront être considérées lors du traitement de flux massif de données spatiales pour ce type d'opérateur.

1.3 Revue de l'existant

Dans cette section, une revue de travaux existants est présentée. Le traitement spatial de flux massif de données est un sujet qui n'a été abordé que tout récemment. En 2015 Nittel (2015) publiait un article sur le traitement de flux spatiaux dans lequel elle abordait les diverses technologies disponibles. Elle concluait ceci :

- Les travaux faits sur les technologies Big Data sont efficaces pour les traitements en lot de données persistantes, mais moins utiles pour le traitement de flux en temps réel.
- L'utilisation de bases de données spatio-temporelles traditionnelles fonctionne bien jusqu'à 500 insertions par seconde, mais pas au-delà.
- Les systèmes de gestion de flux semblent intéressants, mais sont encore en développement.

Au début de nos travaux, à l'automne 2014, aucune référence de travaux de recherche n'était publiée concernant le sujet. Il a fallu attendre la fin 2016 pour qu'un auteur décrive une application de traitement spatial sur des flux massifs de données.

Comme peu de travaux ont porté sur le sujet, la recherche a donc été élargie. Certains articles présentés ci-dessous décrivent des traitements de flux, mais dans un contexte de données traditionnelles. Nous présentons aussi des travaux sur les traitements spatiaux de données massives persistantes. Ce dernier sujet a été traité depuis plus longtemps. Finalement, le forage de données fait sur des flux de données ou sur des données spatio-temporelles est aussi présenté afin d'avoir un aperçu des applications traitant des flux spatiaux à un plus haut niveau.

1.3.1 Traitement de flux spatiaux

Dans cette section, des travaux traitant de flux spatiaux sont présentés. Certains abordent le sujet de façon pratique à l'aide d'un prototype, tandis que d'autres abordent le sujet d'une manière exploratoire ou théorique en définissant un cadre conceptuel.

Dans Nittel *et al.* (2012), les auteurs décrivent un système d'interpolation spatiale pour un phénomène continu (concentration d'un composé organique volatil). Les mesures sont obtenues sous forme de flux à partir de capteurs mobiles. L'objectif est de produire une image par interpolation IDW comme l'illustre la figure 4.

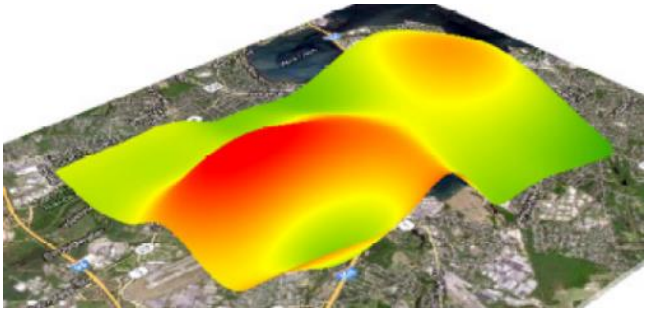


Figure 4 Concentration de COV par interpolation IDW

Un filtre spatial est premièrement appliqué sur le flux pour isoler la zone d'intérêt. Par la suite, les données sont accumulées pendant un certain temps à l'aide de la technique des fenêtres. Chaque information est attribuée à une cellule d'une grille couvrant la zone filtrée. À la fin de la période d'accumulation, les données de chaque grille sont interpolées à l'interne et avec les cellules avoisinantes. Finalement, les informations sont assemblées pour être visualisées. Les auteurs mentionnent que des goulots d'étranglement sont causés par l'empreinte en mémoire et le temps d'exécution. Ils décrivent ensuite la relation entre le nombre de cellules et la qualité de l'interpolation. Puisque la configuration utilisée permet seulement d'obtenir 1,5 point par cellule, les données des cellules adjacentes sont utilisées pour augmenter la qualité. Ceci a pour effet d'augmenter l'empreinte en mémoire et le temps de calcul. Les auteurs présentent trois méthodes pour optimiser les calculs et la mémoire. L'article présente des techniques intéressantes telle l'utilisation de fenêtre pour l'accumulation des données permettant ainsi de résoudre les problématiques des opérateurs bloquants. L'utilisation des filtres spatiaux permet de filtrer le flux pour seulement traiter les informations relatives à la requête en continu. Notons que ce filtre est un opérateur spatial entre un flux de données et une donnée persistante, le polygone représentant la région d'intérêt. L'utilisation d'une grille dans le contexte de cette application est utilisée pour obtenir une valeur pour une position discrète en vue de l'interpolation. La valeur de cette grille est dépendante des valeurs des grilles adjacentes. Une partie du calcul se fait à l'intérieur de la grille et pourrait être facilement parallélisé. Cependant, l'interpolation entre les grilles ne pourrait pas être parallélisée comme elle est décrite. Des trois méthodes d'optimisation des calculs, seule la moins performante pourrait, grâce à son algorithme, être parallélisée. Cet article présente des concepts intéressants bien qu'il n'ait pas été réalisé dans un contexte de données massives.

Dans Zhang et Huang (2010), les auteurs décrivent une méthode permettant de déterminer si un regroupement de points (cluster) obtenus sous la forme d'un flux intersecte une région statique (persistante). L'exemple cité est la formation de régions d'inondations obtenues à partir de capteurs fixes répartis sur un territoire. Dépendant du niveau d'eau détecté par les capteurs, une ou plusieurs régions d'inondation seront déterminées. Par la suite, les comtés (régions statiques) affectés par les inondations sont ciblés. Le résultat est sous la forme d'une valeur

booléenne. La première méthode naïve consiste à créer un regroupement (cluster) à l'aide de l'algorithme DBScan, à calculer un polygone représentant les limites du regroupement puis à vérifier l'intersection. Les auteurs mentionnent que cette méthode est lourde au point de vue des calculs. La deuxième méthode proposée consiste à sélectionner, au hasard, un point à l'intérieur du comté et de vérifier si ce point est un « core point » selon l'algorithme DBScan. Si c'est le cas, alors il y a intersection entre une région d'inondation et le comté, sinon un autre point est testé. Dans cet article, les auteurs utilisent la technique des fenêtres glissantes pour l'accumulation des données. Cet article illustre bien un exemple de relation entre des flux de données et des données statiques et l'utilisation de traitements spatiaux plus avancés tel que le DBScan. Cependant, l'article ne s'inscrit pas dans un contexte de données massives et, en ce sens, n'apporte pas une approche d'exécution en parallèle pour les filtres spatiaux ou une solution pour la problématique des k plus proches voisins intervenant dans le DBSCAN.

L'article de Sharifzadeh et Shahabi (2009) propose une méthode permettant de calculer approximativement une cellule de Voronoi à partir d'un flux de points. Ce flux de points est localisé dans le voisinage d'un point central fixe (germe), donc persistant. Dans le calcul d'une cellule de Voronoi, seul un certain nombre de points contribuent à la solution. Les auteurs expliquent que la nature de l'information étant un flux, les points doivent expirer pour éviter que la cellule ne soit réduite qu'au point central suite aux accumulations successives de données. Ils utilisent donc un concept de fenêtre glissante permettant cette expiration. L'une des conséquences de cette expiration est qu'un point qui ne contribuait pas à la solution initiale puisse y contribuer à la suite de l'expiration d'un point y contribuant. Pour éviter les problèmes de mémoire liés à la conservation de tous les points, un histogramme radial autour du germe est utilisé pour permettre de garder, pour chacune des divisions radiales, seulement les points étant les plus près et pouvant éventuellement y contribuer. Dans cet article, la notion de fenêtre glissante est utilisée pour l'expiration des points. Un partitionnement de l'espace, de forme radiale, est utilisé pour la gestion séparée des points. Il y a encore une illustration des relations entre flux et données persistantes. La discussion ne fait mention que d'un seul germe et ne discute pas d'une solution à plusieurs germes. Finalement, le contexte n'étant pas le traitement de données massives, la technique utilisée n'est pas adaptée au traitement en parallèle.

Dans les articles de Ali *et al.* (2010) et Kazemitabar *et al.* (2010), les auteurs utilisent la plateforme Microsoft StreamInsight pour faire des requêtes en continu sur des flux de données. Dans Kazemitabar *et al.* (2010), les auteurs mentionnent que les librairies spatiales standards ne sont pas adaptées pour fournir un résultat à partir de données provenant de flux et illustrent une méthode permettant d'analyser des flux de données provenant de trafic automobile. L'objectif est de prédire les tendances futures de trafic basé sur les données obtenues par un flux et des données historiques en utilisant une analyse en composante principale. Les auteurs utilisent une jointure spatiale (filtre) entre des données en flux et une région rectangulaire choisie par un utilisateur (entité

persistante) spécifiant une zone d'analyse. L'article ne donne pas beaucoup de détails, mais il est possible de déduire qu'une fenêtre temporelle est utilisée. Certaines techniques utilisées, telle l'utilisation d'un filtre spatial, pourraient être parallélisées. Cependant, le contexte n'est pas le traitement de données massives et l'application présentée n'est pas traitée dans ce sens. Dans Ali *et al.* (2010), un prototype permettant de suivre sur une carte le mouvement de navettes naviguant sur le campus de Microsoft est présenté. Il permet également de déterminer combien de navettes (flux) se trouvent à proximité d'un bâtiment (statique). Dans ces deux articles, les opérateurs spatiaux sont effectués entre des données en flux et des données persistantes (région rectangulaire ou bâtiment). Par contre, aucun n'aborde la problématique des traitements en parallèle ni des données massives.

L'article de Huang et Zhang (2008) propose de nouveaux types de données sur une base spatio-temporelle. Il définit un objet « geo-stream » comme un objet se déplaçant et changeant de géométrie. L'article fournit une approche théorique et mathématique décrivant de nouveaux types de données prenant en compte l'aspect temporel. Par exemple, il décrit un type représentant un flux et un autre type représentant des fenêtres d'accumulation. L'article définit des notations illustrant les informations obtenues à partir de ces types de données, par exemple, comment obtenir la position courante d'un objet ou sa trajectoire des deux dernières heures. Les auteurs font aussi la distinction entre des données temporelles (données persistantes ayant une étiquette temporelle) et un flux de données. Finalement, ils proposent une syntaxe de type SQL permettant de faire des requêtes sur les flux spatio-temporels. Ils présentent plusieurs prototypes de requête. Le tableau 2 illustre les diverses opérations d'algèbre relationnelle des prototypes de requêtes ainsi que les types de données impliqués dans chacune d'elle.

Tableau 2 Caractéristiques des prototypes de requêtes SQL

Type de donnée pour la *Projection* (select)	Type de donnée pour la *Relation* (from)	Type de donnée pour la *Sélection* (where)
Non géométrique	Flux géométrique et géométrie persistante	Durée d'une intersection entre une fenêtre temporelle d'une géométrie et une géométrie persistante.
Géométrique + Non-géométrique	Flux géométrique et géométrie persistante	Aire d'une intersection d'un flux géométrique et d'une géométrie persistante.
Non géométrique	Flux géométrique et géométrie temporelle	Aire des intersections entre le flux et les données temporelles de la dernière année.
Non géométrique	Flux géométrique et flux géométrique	Durée de l'intersection d'une fenêtre de 2 heures avec une fenêtre de 4 heures pour les deux dernières heures.
Non géométrique	Flux géométrique et géométrie persistante	Distance courante entre les deux géométries.
Non géométrique	Flux géométrique et géométrie persistante	Une géométrie persistante est à l'intérieur d'une zone traversée par une fenêtre temporelle d'une géométrie.
Non géométrique	Flux géométrique et géométrie persistante	Durée pendant laquelle une géométrie a été à l'intérieur d'une fenêtre temporelle d'une géométrie en flux.

Cet article présente plusieurs éléments intéressants quant aux divers cas d'étude dans l'analyse de flux spatiaux. Cet article est inspiré de Güting *et al.* (2000) et repris par Galić *et al.* (2014) qui ont produit un prototype. Huang et Zhang (2008) et Galić *et al.* (2014) illustrent plusieurs traitements spatiaux possibles entre des flux et des données persistantes, en plus de l'utilisation de fenêtres de traitement. L'article de Huang et Zhang (2008) est cependant le seul de tous les articles étudiés dans ce mémoire qui illustre un exemple de traitement entre deux flux de données. Cependant, aucun de ces articles n'adresse la problématique sous l'angle des données massives.

Dans l'article de Kazemitabar *et al.* (2011), les auteurs expliquent que les systèmes traditionnels des traitements de données persistantes ne sont pas adéquats pour le traitement massif de données spatiales sous forme de flux. Soutenant la nécessité du traitement de ces données en parallèle d'un côté et de l'aspect économique et de la simplicité opérationnelle de l'autre, ils déterminent que l'infonuagique possède les propriétés nécessaires pour le traitement à grande échelle de flux géospatiaux. Les solutions infonuagiques permettent de rapidement et facilement adapter la puissance de calcul d'un système selon les besoins, en ajoutant ou enlevant des nœuds dans la grappe d'ordinateurs. Le coût de ces ordinateurs, qui sont des ressources externes à l'entreprise, est calculé en fonction de leurs utilisations. Par la suite, les auteurs décrivent une architecture de système en parallèle ressemblant au fonctionnement de Hadoop MapReduce. Finalement, ils décrivent des cas d'études pour différents opérateurs (c'est-à-dire projection, sélection, groupement et agrégation) de façon très générale. Malheureusement, ils ne discutent aucunement des spécificités de certains opérateurs spatiaux, comme les opérateurs topologiques (ex. intersection) et de proximité (ex. k voisins les plus près) et comment adapter ceux-ci dans un traitement en parallèle. L'article mesure divers points de vue où il met en lumière l'importance d'une architecture extensible et dynamique dans le traitement de flux massif de données.

Dans leur travail, Lee et Kang (2015), présentent les défis et opportunités des données géospatiales massives. Les auteurs soulignent l'importance de porter un intérêt plus grand à l'analyse dynamique et interactive des données géospatiales pour la prise de décision en temps opportun. À cette fin, il identifie deux composantes permettant ces analyses : le traitement d'évènement complexe (CEP) et le traitement spatial analytique en ligne (SOLAP). Les CEP permettent de traiter les flux d'information et de détecter en continu des évènements et d'avertir les utilisateurs ou d'autres systèmes. Dans le cas des outils SOLAP, les informations sont traitées et chargées en temps réel dans des entrepôts de données spatiales massives pour être utilisées de façon interactive par un utilisateur.

Dans l'article de Li *et al.* (2015), les auteurs révisent les différentes méthodes et théories actuelles de manipulation et traitement des données géospatiales en vue de déterminer leur capacité face à l'émergence des données géospatiales massives. Un des points sur lequel ils émettent des inquiétudes est le déphasage

entre le monde industriel et académique dans l'utilisation dominante des bases de données spatiales traditionnelles (ex. Oracle et PostGIS) par ce dernier et l'émergence d'architecture parallèle comme MapReduce dans le traitement de données. Ils pointent aussi vers l'importance de développer des méthodes efficaces de partitionnement spatial, possiblement à l'aide de courbe de remplissage (space-filling-curve) de type Hilbert.

Dans les articles de Zheng *et al.* (2009, 2011) et Zheng et Xie (2011), les auteurs s'intéressent au forage de données historiques géospatiales pour en extraire des points d'intérêts (POI). L'objectif de ces recherches est de présenter ces informations à d'éventuels utilisateurs en temps réel. L'analyse est effectuée en calculant des trajectoires, des points de visite ainsi que des séquences de déplacement. L'auteur utilise entre autres un regroupement sur une base de densité spatiale (density base clustering). Dans Giannotti *et al.* (2011), les auteurs proposent aussi des méthodes d'analyse pour effectuer du forage de données pour en extraire des informations de trajectoire. L'auteur présente certaines primitives spatio-temporelles comme l'attroupement représentant la région où coïncide un groupe de points en mouvement (T-Flock), la visite de région dans la même séquence (T-Pattern), le déplacement d'une région vers une autre (T-Flow) et les trajectoires partageant une même propriété (T-Cluster). Bien que ces exemples reposent sur des données historiques, nous trouvons intéressant de les faire ressortir puisque les données massives sont souvent associées au forage de données. Ces travaux peuvent servir d'indicateur permettant de déterminer les types d'opérateurs spatiaux étant susceptibles d'être utiles dans les flux massifs de données spatiales.

L'article de Tang *et al.* (2012) présente un cadre permettant l'analyse de trajectoires pour découvrir les compagnons de voyage dans un flux de données. L'article porte sur l'optimisation des calculs de regroupement (cluster) et d'intersection pendant une certaine période de temps (fenêtre). Dans ce travail, le regroupement est fait sur la base de la densité spatiale. Cet article s'inscrit dans le même contexte que ceux cités précédemment, mais introduit l'aspect de flux de données aux traitements de trajectoire. Cet article permet aussi d'avoir une idée des types de traitements spatiaux qui seraient utilisés dans un contexte de flux massifs de données spatiales, tels les compagnons de voyage.

L'article de Galić (2016b) est un des premiers à aborder la problématique des flux de données spatiales dans un contexte de données massives. L'auteur définit un cadre formel constitué de types de données et d'opérateurs pour des objets mobiles. Il fait le pont entre les bases de données spatio-temporelles, les flux de données et le paradigme des données massives. Finalement, il montre quelques exemples d'applications à l'aide d'un prototype utilisant Apache Flink[1] :

[1] https://flink.apache.org/

- Dans le premier exemple, les objets mobiles se trouvant dans une région d'intérêt statique sont signalés en continu. Pour ce faire, les flux de données sont filtrés spatialement à l'aide d'un polygone statique.

- Le deuxième exemple signale chaque minute tous les objets s'étant déplacés d'au moins 3 km dans les 10 dernières minutes. Les objets sont groupés par leur identifiant et la longueur de leur trajet est calculée à l'aide de la distance entre chacun des points durant la période d'accumulation à l'aide d'une fenêtre glissante. Finalement, un filtre est appliqué sur cette longueur pour ne garder que ceux ayant voyagé plus de 3 km.

- Le troisième exemple permet de trouver, pour chaque objet mobile, sa distance minimale avec un point d'intérêt fixe durant les dernières 30 minutes. Les points sont groupés par leur identifiant pendant 30 minutes à l'aide d'une fenêtre. À la fin de la fenêtre, la distance minimale avec le point fixe est calculée.

- Le quatrième exemple consiste à trouver les objets mobiles qui ont voyagé plus de 10 km dans la dernière heure. Cet exemple est très similaire au deuxième exemple.

- Finalement, le cinquième exemple permet de trouver la trajectoire des objets mobiles ayant été à moins de 500 m d'un point d'intérêt fixe dans les 15 dernières minutes. Les entités sont groupées par leur identifiant pendant 15 minutes à l'aide d'une fenêtre. Par la suite, un filtre basé sur la distance entre les points et un point d'intérêt fixe permet de garder ceux ayant été à moins de 500 m pour calculer leurs trajectoires.

Malgré que ces exemples démontrent bien le cadre décrit dans l'article, toutes les analyses spatiales sont effectuées entre des flux et des géométries persistantes dites traditionnelles. À avis, ces exemples ne représentent que des cas où les traitements en parallèle requièrent peu d'effort afin de séparer les tâches en parallèle (embarrassingly parallel). En effet, il y a peu ou pas de dépendance ou de besoin de communication entre les tâches parallèles. L'exercice aurait été plus ardu si les géométries comparées entre elles avaient été toutes deux issues de flux massif ou si la donnée persistante avait été massive.

Pour conclure cette section, un des concepts importants qui ressort dans plusieurs de ces travaux est l'utilisation de fenêtre de traitement pour résoudre la problématique des opérateurs bloquants. On remarque également que les flux sont souvent comparés à des données persistantes, mais un seul article illustre des traitements sur deux flux.

1.3.2 La donnée spatiale dans les traitements massifs de données

Cette section aborde des travaux analysant des traitements spatiaux dans un contexte de données massives persistantes. Ce sujet a été étudié depuis plus longtemps et il a été le sujet de plusieurs recherches. Même si le contexte n'est pas le même que celui abordé dans ce mémoire, il demeure tout de même pertinent d'étudier les techniques utilisées dans un contexte de données massives et persistantes.

L'article de Eldawy et Mokbel (2015) se penche sur des problématiques telles que l'union de polygones, la création d'enveloppes convexes (convex hull), la paire de points les plus éloignés et les plus proches. Bien que

ces algorithmes soient bien définis, les auteurs constatent que malheureusement, ils sont difficilement extensibles à des bases de données comportant des milliards de points. Ils présentent CG_Hadoop (Spatial Hadoop), une version spatialisée de la plateforme de traitement de données massives Hadoop. Ce système organise les données en un index global (grille) et un index local (R-Tree). L'index global permet de partitionner les données sur les différents nœuds, tandis que l'index local organise les données à l'intérieur des nœuds. Cette organisation permet de faire un élagage, à l'aide d'un filtre, des données ne participant pas à la solution. Les auteurs comparent deux versions d'une union de polygones. Une avec une partition spatiale où les données d'une même région ont été placées sur un même nœud et une autre version où les polygones ont été distribués aléatoirement sur les nœuds. La première méthode s'avère plus efficace, car les polygones adjacents seront combinés en un seul avant le traitement final. Les calculs finaux seront donc moins intenses pour les nœuds où se concentreront ces tâches. De plus, cette méthode engendrera moins de trafic sur le réseau.

L'article de Eldawy et Mokbel (2013) des mêmes auteurs porte sur des opérations de filtre spatial, de voisin le plus proche (kNN) et de jointure spatiale et sont effectuées à l'aide de la plateforme "Spatial Hadoop" décrite précédemment. L'opération de filtre spatial est accomplie en utilisant l'index global (grille) pour ne garder que les partitions contenues à l'intérieur d'un rectangle englobant. Par la suite, pour chaque partition, les points sont filtrés à l'aide de l'index local (R-Tree). L'opération kNN est faite en deux temps. Premièrement, l'index global est utilisé pour trouver la partition dans laquelle le point se situe. Par la suite, l'index local est utilisé pour obtenir les kNN de cette partition. Pour vérifier l'exactitude de la réponse, un cercle ayant un rayon équivalant à la distance entre le point et le $k^{ième}$ élément est tracé. Le résultat est exact si ce cercle est compris uniquement dans la partition. Dans le cas contraire, le calcul est fait en utilisant les partitions qui sont superposées au cercle. Pour la jointure spatiale, les index globaux des deux fichiers (contenant les classes d'entités) sont utilisés pour trouver les partitions se superposant. Par la suite, l'index local est utilisé pour trouver les entités ayant une jointure spatiale.

Encore des mêmes auteurs, les articles de Eldawy (2014) et Eldawy et Mokbel (2015) mentionnent d'autres fonctions spatiales pouvant être calculées à l'aide d'une architecture MapReduce. Par exemple, des fonctions de base permettant d'extraire de l'information d'une entité, telle que l'aire d'un polygone, des prédicats relationnels de base retournant une valeur booléenne (ex. touche, intersection), des fonctions d'analyses spatiales comme le centroïde et l'intersection et finalement des fonctions d'agrégation spatiale comme l'enveloppe convexe (convex hull).

Les auteurs de Aji *et al.* (2013) ont développé une approche similaire à celle citée précédemment. Leur version, Hadoop-GIS, permet l'intégration avec Hive. Hive est une plateforme de données massive basée sur HDFS (Hadoop File System) qui permet l'interrogation des données sous une forme similaire au langage SQL. Ils

présentent plusieurs types de requêtes spatiales, telles que des requêtes de jointures spatiales, le voisin le plus proche et des agrégations d'entités. La méthode consiste à partitionner les données dans des tuiles selon leurs proximités spatiales. Les auteurs soulèvent cependant deux considérations. La première étant un débalancement des données dans les tuiles dans le cas où les données ne seraient pas réparties uniformément dans l'espace. Pour résoudre cette problématique, les tuiles ayant une forte densité sont divisées. Pour ce faire, ils utilisent des méthodes comme les R*-Tree ou Hilbert R-Tree. La deuxième problématique est liée à l'entité dont les frontières croisent plusieurs tuiles. La méthode utilisée consiste à dupliquer cette entité et à l'assigner à chacune des tuiles.

Dans Whitman *et al.* (2014), les auteurs comparent les performances de certaines requêtes spatiales sur des données massives dans une architecture basée sur l'écosystème Hadoop. Il compare une architecture où les données sont indexées par un partitionnement spatial sur les nœuds versus une architecture où les données sont réparties aléatoirement sur les différents nœuds du système. Dans la première méthode, les données sont partitionnées à l'aide d'un Quadtree afin de permettre une distribution balancée de celles-ci sur les nœuds. La figure 5, tirée de l'article, illustre le concept de partitionnement utilisé.

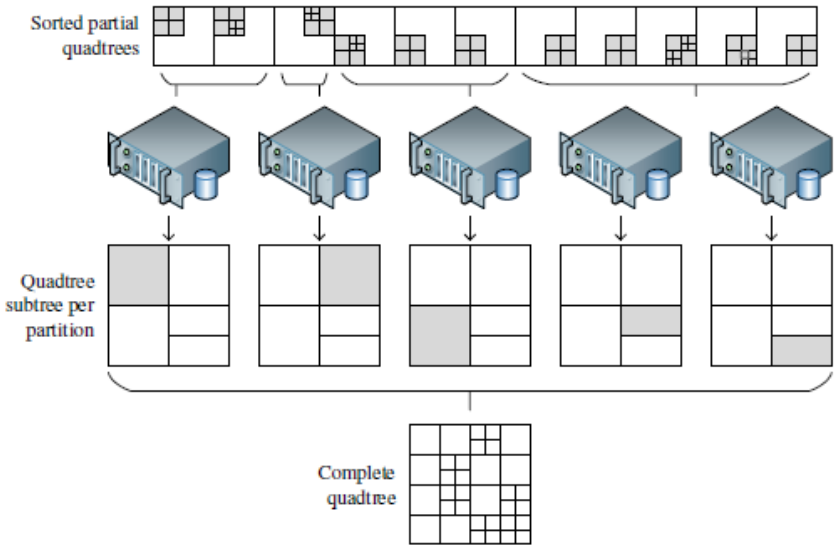


Figure 5 Partitionnement en QuadTree sur Hadoop. Source : Whitman *et al.* (2014)

Finalement, cet article mentionne entre autres que l'indexation spatiale persistante n'est pas toujours justifiée, par exemple lors d'une requête de filtre spatial retournant une portion significative du jeu de données ou lors d'une jointure de deux grands jeux de données. Dans ces cas, le gain en performance ne se démarque pas d'une indexation spatiale à la volée (Neis et Zipf, 2012). Cependant, l'indexation spatiale persistante est justifiée lors d'analyse de proximité spatiale (Aji *et al.*, 2013; Eldawy *et al.*, 2013).

Dans l'article de He *et al.* (2014), les auteurs discutent d'un algorithme pour effectuer un regroupement (clustering) basé sur la densité spatiale (DBSCAN) dans le paradigme MapReduce. L'algorithme DBSCAN consiste à regrouper des points à l'intérieur d'un groupe en se basant sur des critères de proximité entre eux. Deux types de points peuvent faire partie d'un groupe. Les points centraux (core) et les points de bordures (border). Un point est central s'il possède un nombre minimum de points voisins à une distance maximale (ε). Un point est dit de bordure s'il ne possède pas ces caractéristiques, mais qu'il fait partie des points voisins d'un point central. Tous les autres points sont considérés comme du bruit (noise). Les points d'un même groupe se voient assigner un identifiant unique. La solution proposée pour effectuer ce regroupement en MapReduce consiste en trois phases : le partitionnement des données, le regroupement local et finalement une fusion globale. Le partitionnement est fait à l'aide d'un algorithme de partitionnement basé sur le coût de calcul (cost-based partitionning) basé entre autres sur le partitionnement binaire de l'espace (BSP). Le calcul DBSCAN local ne considérera que les points de sa partition. Des effets de bordure peuvent survenir. Ces effets surviennent lorsqu'un point pourrait faire partie des voisins d'un autre voisin, mais qu'il se trouve dans une partition adjacente. Pour résoudre ce problème, les bordures des partitions ont été élargies à deux fois la distance ε. De cette façon, un point se trouvant près de la frontière de la partition ne sera pas oublié. Finalement, les regroupements locaux sont fusionnés de façon globale en effectuant une jointure sur les points appartenant à plusieurs partitions.

En conclusion de cette section, on remarque que le partitionnement des données est au cœur des traitements spatiaux de données massives. Cependant, tous ces travaux sont fondés sur les traitements de données persistantes. Comme le mentionne Nittel (2015), bien que ces systèmes soient efficaces dans le traitement de données en lot, ils sont moins utiles pour les traitements en temps réel.

1.4 Problématique

Le traitement de données massives a vu le jour il y a une dizaine d'années grâce à l'augmentation de la puissance des ordinateurs, le faible coût de stockage et l'explosion des données numériques. Comme nous l'avons vu, le paradigme MapReduce est devenu important dans le traitement des données massives, car il permet l'utilisation d'ordinateurs peu dispendieux et il est évolutif, c'est-à-dire que le système peut augmenter sa puissance en ajoutant simplement des nœuds à la grappe d'ordinateurs. Initialement composé d'un système de gestion de fichiers (HDFS) et d'un système de traitement de données (Hadoop MapReduce), cet écosystème a rapidement évolué avec l'ajout de plusieurs plateformes répondant chacune à un besoin spécifique et améliorant les performances. Cependant, toutes ces technologies étaient centrées initialement sur le traitement en lot de données persistantes. C'est sur ce type de données que les premières adaptations de traitement de données spatiales ont vu le jour. Par la suite, des systèmes de traitement en flux ont été créés (Storm, Flink) ou adaptés à partir de système de traitement en lot (Spark).

Au moment d'écrire ces lignes, très peu d'études ont porté sur les traitements spatiaux de flux massifs. À notre connaissance, seuls les travaux de Galić *et al.* (2014) et Galić (2016a, 2016b) portent sur les traitements spatiaux de flux massifs. Comme nous l'avons vu dans les sections précédentes, ces travaux ont porté sur la définition d'un cadre formel définissant des types de données et des opérateurs pour les objets mobiles. Les opérateurs de traitements spatiaux n'ont pas été abordés dans l'optique où les deux opérandes étaient sous forme de flux. Les exemples de traitements ont tous été faits en plaçant en relation les flux géospatiaux avec une entité spatiale persistante non massive. De ce fait, ces exemples abordent une classe de problème qui représente peu d'effort pour diviser les tâches en parallèle, on dit alors qu'ils sont « embarrassingly parallel ». Ce sont les cas les mieux adaptés à ce type de traitement. Il serait intéressant d'étudier les cas où les analyses sont effectuées sur deux opérandes en flux massif ou les cas où l'une est issue d'un flux massif et l'autre de données persistantes massives. Dans ces articles, les auteurs ont repris une partie des requêtes proposées par Huang et Zhang (2008). Cependant, les opérations entre deux flux n'ont pas été reprises en exemple.

Comme nous avons vu dans les sections précédentes, des travaux ont été faits sur le traitement de flux spatial, sur le traitement en parallèle d'opérations spatiales et sur le traitement de données massives spatiales persistantes. Cependant, il existe une lacune concernant le traitement spatial de flux massifs.

De façon plus générale, quels sont les cas de figure dans le traitement spatial de flux massif? Quelle méthode pourrait être créée pour aborder les problématiques des opérateurs ayant des opérandes issus de flux massifs?

Le traitement spatial de flux massifs n'est pas seulement lié aux opérateurs spatiaux, mais aussi aux différents contextes par lequel la donnée est rendue disponible. Cependant, il n'existe pas de classification des différents cas de figure dans ce domaine.

Les recherches qui ont été faites jusqu'à maintenant amènent certaines pistes de solution dans certains cas. Ces cas font tous intervenir des données persistantes, aucun ne traite du cas de deux flux de données. La problématique est donc de déterminer comment adapter les traitements spatiaux portant sur des relations entre les entités, lorsque celles-ci sont issues de flux massifs de données spatiales. C'est sous cet angle que les traitements spatiaux seront abordés dans ce mémoire.

1.5 Objectifs

L'objectif principal de cette recherche est de développer des méthodes permettant le traitement de données spatiales et massives qui tiennent compte de leur accessibilité (flux ou persistantes), des « volumes » des données (massives vs traditionnelles) et des relations spatiales ou non entre les opérandes. Comme il sera vu plus loin dans ce mémoire, peu de travaux ont été effectués sur ce sujet. Il est donc nécessaire de définir une classification des différents cas de figure pour lesquels un traitement de flux massifs de données doit être utilisé.

Après avoir bien cerné ces cas de figure et leurs caractéristiques propres, il sera possible de développer des méthodes de traitements.

L'atteinte de cet objectif passe par la réalisation des objectifs secondaires suivants :

- Fournir une classification des cas de figure pouvant être rencontrés dans le traitement spatial de flux massifs en tenant compte des caractéristiques de la donnée (accessibilité et volume) ainsi que des caractéristiques des opérateurs.
- Proposer des stratégies générales de traitement pour les différentes classes identifiées.
- Proposer des méthodes de traitement détaillées pour au moins deux opérateurs d'une de ces classes.
- Réaliser un prototype afin de démontrer la faisabilité et tester les performances des méthodes proposées.

1.6 Méthodologie

Cette recherche comporte une phase importante d'exploration considérant le peu d'études disponibles et l'aspect récent du sujet. De plus, comme elle s'appuie sur l'élaboration d'un prototype et sur l'utilisation de nouveaux outils technologiques en cours de développement, la méthode Agile semble appropriée. Cette méthode, qui est particulièrement utilisée dans le développement logiciel, permet de répondre aux éléments imprévisibles d'une façon incrémentale et itérative. La figure 6 présente les différentes phases de la recherche qui sont décrites dans les sections suivantes.

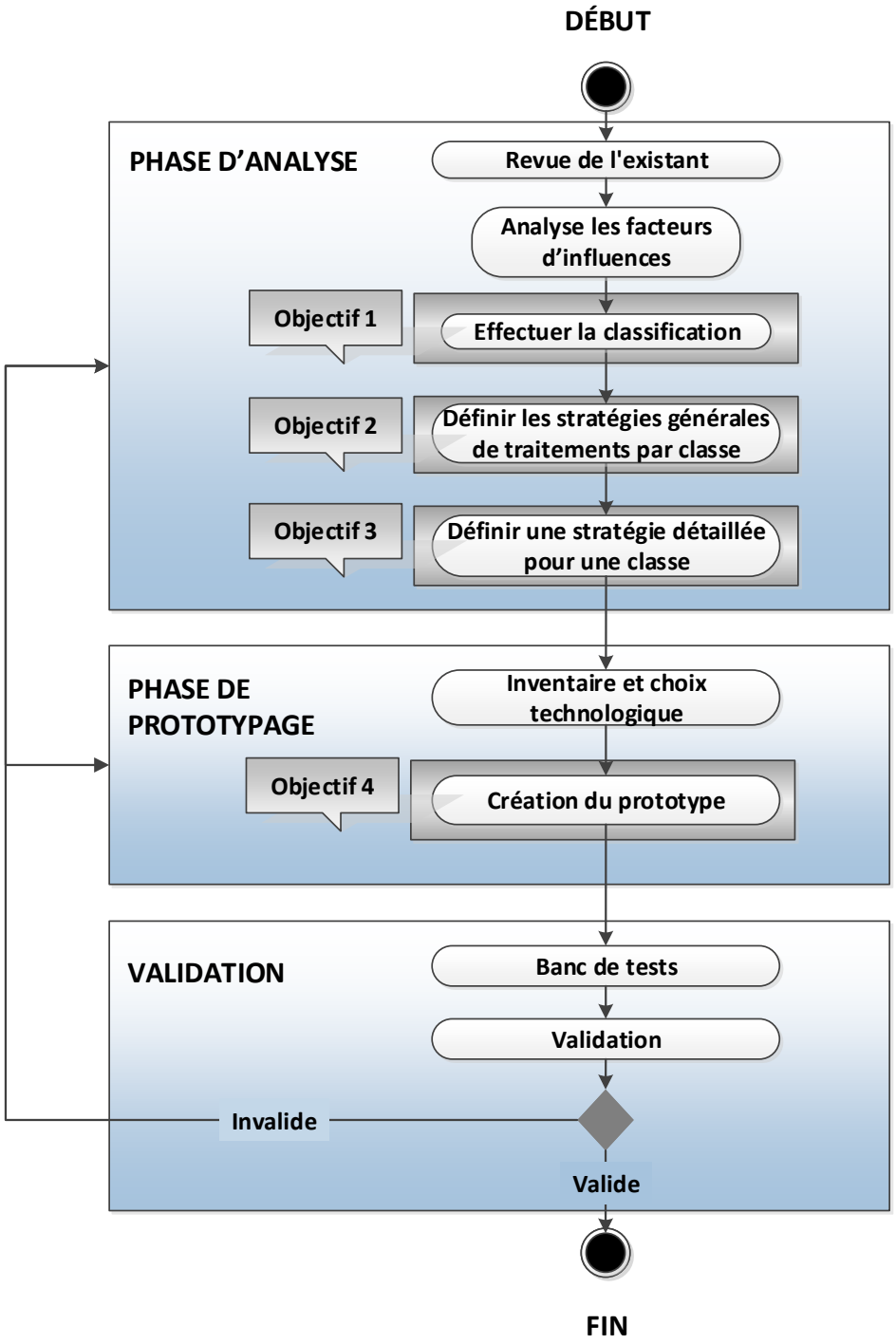


Figure 6 Méthodologie du projet

1.6.1 Phase d'analyse

Cette première phase permet d'analyser la problématique et de proposer des solutions. Elle se découpe selon les étapes suivantes :

- Revue des concepts et analyse des travaux existants;
- Analyse des facteurs influençant les traitements spatiaux dans un contexte de flux massifs;
- Classification des différents cas rencontrés dans ces traitements;
- Élaboration de stratégies générales de traitement pour chacune des classes;
- Proposition de traitements détaillés pour une de ces classes.

Cette phase est une portion importante de nos travaux, car elle regroupe 3 des 4 objectifs secondaires identifiés.

1.6.2 Phase de prototypage

Afin de prouver la faisabilité et tester les méthodes proposées dans la phase précédente, un prototype doit être mis en œuvre. Afin de développer ce dernier, une plateforme technologique a été choisie à l'aide d'une liste de critères permettant de répondre aux besoins établis lors des phases précédentes. Un prototype a ensuite été développé pour 2 opérateurs, ce qui correspond au quatrième objectif secondaire.

1.6.3 Validation

Afin de tester le prototype, un banc de test et une validation des résultats ont eu lieu. La validation consistait à valider l'exactitude des résultats et des méthodes ainsi qu'à effectuer certaines optimisations du code du prototype. Des boucles de rétroaction permettaient de raffiner le prototype ainsi que l'analyse.

1.7 Description des prochains chapitres

Le présent chapitre a situé l'état actuel des recherches à ce jour concernant la problématique qui nous occupe. Il a permis d'identifier le sujet qui sera abordé dans ce travail en définissant la problématique mise en lumière par l'état actuel de la recherche dans le domaine étudié. Des objectifs permettant d'étudier les méthodes de traitements spatiaux dans un contexte de flux massifs ont été énoncés et une méthodologie permettant de les atteindre a été proposée.

Le chapitre 2 traite quant à lui de l'analyse des traitements de flux spatiaux dans un contexte de données massives sous l'angle d'une classification des cas de figure pouvant être rencontrés. Il se poursuit avec la présentation d'une méthode permettant d'effectuer ces traitements. Finalement, il fournit une méthode détaillée

pour l'adaptation des opérateurs relationnels et des analyses de proximité dans un contexte où les opérandes sont issus de flux massifs.

Le chapitre 3 fait un inventaire des technologies existantes permettant le traitement de flux massif de données. Il décrit aussi la mise en œuvre des prototypes pour les traitements spatiaux pour des opérateurs relationnels et des opérateurs d'analyse de proximité. De plus, ce chapitre traite de l'architecture du prototype et du banc de tests.

Les résultats et leur analyse sont présentés dans le chapitre 4.

Finalement, le chapitre 5 conclut ce mémoire en résumant les résultats importants obtenus, en présentant la contribution de ce travail de recherche et en suggérant des perspectives de recherche future.

Chapitre 2 Phase d'analyse

Ce chapitre est consacré à l'analyse des facteurs influençant le choix des méthodes d'adaptation pour le traitement spatial de flux massifs. Dans un premier temps, les facteurs d'influence sont analysés pour déterminer quelles sont les caractéristiques et propriétés qui doivent être prises en compte.

Après cet inventaire, une classification des cas de figure rencontrés lors d'un traitement spatial sur des flux massifs de données est effectuée. Pour chacune de ces classes, des techniques générales d'adaptation sont proposées et une évaluation de la complexité entre ces familles est réalisée.

Suite à cette classification, une classe sera choisie pour permettre d'approfondir l'analyse des techniques d'adaptation proposées pour celle-ci. Par la suite, des traitements spatiaux de base seront choisis afin de tester et d'appliquer les propositions énoncées pour cette classe.

Afin de choisir la plateforme utilisée pour la conception du prototype, qui sera fait dans le chapitre 3, un inventaire des caractéristiques requises pour le système de traitement de flux est effectué.

Finalement, le chapitre est conclu en résumant les quatre caractéristiques importantes des flux massifs de données spatiales qui ont été utilisées pour faire la classification et pour proposer des adaptations de traitements spatiaux.

2.1 Analyse des facteurs influençant les traitements spatiaux de flux massifs

Cette section analyse les différents facteurs influençant les traitements spatiaux de flux massifs. Dans un premier temps, la nature même des opérateurs et de leurs opérandes sera étudiée. Par la suite, la caractéristique de la donnée sera analysée pour comprendre comment elle influence les traitements. Les propriétés temporelles et quantitatives, soit l'intensité en volume ou en débit, sont les caractéristiques abordées.

2.1.1 Les opérateurs spatiaux de base et leurs propriétés

Comme nous l'avons vu, certains opérateurs spatiaux peuvent être considérés comme des opérateurs simples servant de base pour l'élaboration d'autres opérateurs. On les appellera des opérateurs primaires. On retrouve ces opérateurs dans plusieurs travaux et normes dans le domaine spatial, notamment dans les spécifications OGC (Open Geospatial Consortium, 2010, 2011). Ces opérateurs sont ceux qui sont décrits dans le Tableau 1 du chapitre précédent.

Les sous-sections suivantes décrivent deux caractéristiques des opérateurs spatiaux, soit l'arité et l'attribut relationnel entre les opérandes. Ces deux caractéristiques influencent, selon nous, les méthodes de traitements

spatiaux de flux massifs. Le tableau 3 ci-dessous reprend les opérateurs en y ajoutant ces deux caractéristiques. Les explications sont présentées dans les sections suivantes.

Tableau 3 Arité et attribut relationnel des opérandes des opérateurs de l'OGC

Classe	Opérateur	Arité	Attribut relationnel	Description
Opérateurs de base	SpatialReference	1	Aucun	Retourne le système de référence spatial de l'objet.
	Dimension	1	Aucun	Retourne la dimension de l'objet.
	GeometryType	1	Aucun	Retourne le type de l'objet géométrique.
	Envelope	1	Aucun	Retourne le rectangle englobant.
	IsEmpty	1	Aucun	Retourne vrai si l'objet est vide.
	Boundary	1	Aucun	Retourne les limites de l'objet.
Opérateurs relationnels	Equals	2	Spatial	Retourne vrai si deux objets sont spatialement égaux.
	Disjoint	2	Non spatial	Retourne vrai si deux objets sont spatialement disjoints.
	Intersects	2	Spatial	Retourne vrai si deux objets sont spatialement intersectés.
	Touches	2	Spatial	Retourne vrai si deux objets se touchent spatialement.
	Crosses	2	Spatial	Retourne vrai si deux objets se croisent spatialement.
	Within	2	Spatial	Retourne vrai si un objet est spatialement à l'intérieur de l'autre.
	Contains	2	Spatial	Retourne vrai si un objet est spatialement contenu dans l'autre.
	Overlaps	2	Spatial	Retourne vrai si les objets se superposent spatialement.
	Relate	2	Spatial	Retourne vrai si les objets sont en relation spatiale selon une matrice d'intersection.
	LocateAlong	1	Aucun	Retourne une portion de l'objet géométrique correspondant à la mesure spécifiée.
	LocateBetween	1	Aucun	Retourne une portion de l'objet géométrique correspondant à la mesure spécifiée.
Opérateurs d'analyse spatial	Distance	2	Non spatial	Calcule la distance entre deux objets.
	Buffer	1	Aucun	Retourne un objet contenant tous les points contenus dans une certaine distance maximale d'un objet.
	ConvexHull	1	Aucun	Retourne un polygone convexe englobant d'un objet.
	Intersection	2	Spatial	Retourne le résultat de l'intersection de deux objets.
	Union	2	Non spatial	Retourne le résultat de l'union de deux objets
	Difference	2	Non spatial	Retourne le résultat de la différence entre deux objets.
	SymDifference	2	Non spatial	Retourne le résultat de la différence symétrique entre deux objets.
Opérateurs métriques	X, Y et Z	1	Aucun	Coordonnées d'un point.
	Lenght	1	Aucun	Longueur d'une ligne.
	Area	1	Aucun	L'aire d'un polygone.
	Centroid	1	Aucun	Le centroïde d'un polygone.

2.1.1.1 L'arité des opérateurs spatiaux

Les opérateurs du tableau précédant nécessitent, selon le cas, 1 ou 2 paramètres géométriques en entrée. Le nombre de paramètres, ou opérandes, s'appelle l'arité de l'opérateur. Un opérateur unaire aura une arité de 1, tandis qu'un opérateur binaire aura une arité de 2. Par exemple, la création d'une zone tampon (buffer) prend

en paramètre une entité spatiale et une distance. Dans ce cas, on dira que l'opérateur est unaire, car son arité spatiale est de 1. Pour l'opérateur d'Union, deux entités spatiales sont requises comme paramètre et l'opérateur est donc binaire, car son arité est de 2. Les données étant distribuées sur plusieurs nœuds, le nombre d'opérandes devient un élément important à considérer dans la distribution des traitements, car au final, l'opérateur est exécuté localement avec les données dont il dispose.

D'autres facteurs, également importants, sont les relations entre ces opérandes. Ces relations sont abordées dans la section suivante.

2.1.1.2 L'attribut relationnel entre les opérandes des opérateurs spatiaux

Les traitements en parallèle sont le cœur des traitements de données massives. Ceci implique que les données doivent donc être réparties sur les nœuds du système. La méthode de répartition variera selon le traitement effectué. Les opérations effectuées sur un nœud n'ont accès qu'à un sous-ensemble des données qui ont été assignées à ce dernier. Cette distribution n'a pas d'importance pour un opérateur unaire, car celui-ci n'a besoin que de la donnée dont il dispose pour calculer le résultat. On dira que le traitement de cet opérateur est parallèlement embarrassant (embarrassingly parallel), car il y a peu ou pas d'effort à faire pour diviser les tâches en parallèle. Ce sont les cas les mieux adaptés à ce type de traitement. Par exemple, obtenir une zone tampon (buffer), calculer l'aire ou changer de projection sont toutes des opérations ayant une arité de 1, donc n'ayant qu'un seul paramètre géométrique.

Dans le cas d'un opérateur binaire, le calcul requiert que les deux opérandes, les entités spatiales, soient sur le même nœud. La relation liant les deux entités spatiales est une notion importante. Cette relation peut être de nature spatiale ou non spatiale. C'est cette relation qui dictera comment les données devront être réparties sur les nœuds pour un traitement efficace de celles-ci. Par exemple, pour l'opérateur binaire et relationnel « Contient », la proximité spatiale est un facteur important. Pour obtenir un résultat positif, il faut absolument que les deux entités soient spatialement à proximité l'une de l'autre. À l'inverse, les opérateurs « distance » et « union » n'ont pas besoin d'une proximité spatiale pour fournir un résultat non nul. Cet attribut relationnel entre les opérandes est, selon nous, la base d'une répartition efficace des données sur les différents nœuds du système. Cet attribut pourra donc être de nature spatiale ou non spatiale.

Les opérateurs binaires représentent donc un niveau de complexité plus élevé que les opérateurs unaires. Le tableau précédant montre, pour les opérateurs d'arité 2, la nature de l'attribut relationnel entre les opérandes. La nature « Spatiale », « Non spatiale » ou « Aucun » de l'attribut relationnel entre les opérandes d'un opérateur est assigné de la manière suivante :

- **Spatial** : Les opérandes géométriques doivent occuper un même espace, c'est-à-dire être en relation topologique, afin que le résultat soit non nul ou positif.

- **Non spatial** : L'occupation d'un même espace entre les opérandes géométriques n'intervient pas dans l'obtention d'un résultat non nul ou positif.

- **Aucun** : L'opérateur est unaire, il n'a qu'un seul opérande géométrique.

2.1.1.3 Opérateurs spatiaux complexes

En plus des opérateurs de base, d'autres méthodes d'analyse plus complexes existent. Par exemple, la construction de ligne à partir de points, les k plus proches voisins et le groupement par densité spatiale (clustering). D'autres traitements spécifiques aux données temporelles existent aussi, tels que l'attroupement, les rencontres, les emplacements fréquents et la détection de patrons répétitifs (Giannotti *et al.*, 2011; Galić, 2016a).

Certains de ces traitements, comme le knn limitant la recherche à l'intérieur d'un certain rayon, pourraient être vus comme une combinaison simple de traitements spatiaux de base, par exemple les opérateurs distance, zone tampon et intersection ou filtres spatiaux selon la méthode utilisée. D'autres, comme le clustering ou la détection d'attroupements, peuvent être plus complexes à résoudre.

Ces traitements spatiaux complexes possèdent certainement des caractéristiques spécifiques qui seraient à leurs tours des facteurs influençant les méthodes de traitement dans un contexte de flux massif. Les traitements spatiaux complexes utilisent plusieurs types et enchainements d'algorithmes. Ces traitements utilisent certainement des traitements spatiaux de base, mais possèdent aussi des spécificités qui leur sont propres. Puisqu'en ce moment, il n'existe pas encore de méthode de traitement pour les opérateurs de base, il est logique d'aborder ceux-ci en premier. Donc, dans le cadre de ce mémoire, ces traitements plus complexes ne sont pas abordés en détail. Seule l'analyse des "k" voisins les plus proches limités dans un rayon de recherche fait l'objet d'une analyse pour la catégorie des traitements complexes afin d'avoir un aperçu des caractéristiques nécessaires.

2.1.2 Les caractéristiques des données

Cette section analyse les caractéristiques des données et leur influence sur les méthodes de traitement. Les propriétés temporelles des données, statique ou dynamique, sont présentées en premier. Ensuite, l'aspect quantitatif, massif ou traditionnel, est analysé.

2.1.2.1 La propriété temporelle des opérandes

Un autre aspect pouvant influencer les méthodes d'adaptation est la forme sous laquelle les données sont rendues disponibles dans le système. Dans un jeu de données persistantes, l'ensemble de celles-ci est disponible avant le début des traitements. Selon le type de traitement, les données peuvent être structurées ou traitées afin de répondre efficacement aux requêtes. Cependant, lorsque les données sont accessibles à travers un flux, celles-ci doivent être analysées en temps réel afin de produire un résultat ayant une valeur maximale.

De plus, plusieurs auteurs mentionnent que les applications des traitements de données en temps réel doivent pouvoir mettre en relation des données persistantes (ou statiques) et des données sous forme de flux (ou dynamiques) (Costa *et al.*, 2011; Galić, 2016b, 2016c). Il faut donc regarder plusieurs cas de figure.

Dans le cas des opérateurs unaires, nous nous pencherons seulement sur le cas des flux de données. Pour les opérateurs binaires, trois cas de figure sont possibles et sont présentés dans le tableau 4.

Tableau 4 Caractéristique temporelle des données

Propriété temporelle (opérateur binaire)	Description
Flux – Flux	Les deux opérandes géométriques sont issus de flux.
Flux – Persistant	Un des opérandes est issu d'un flux tandis que l'autre est issu d'une source persistante.
Persistant – Persistant	Les deux opérandes sont issus de données persistantes.

D'emblée, le dernier cas n'a pas été considéré, car il s'agit d'un cas de traitement purement en lot et qui n'est pas le sujet de ce mémoire et qui est déjà, nous l'avons vu, abordé dans la littérature.

Dans le cas où l'une des sources de données est persistante, un plus grand contrôle est possible quant à sa structuration et sa répartition sur les nœuds. Leur répartition peut être faite à l'avance. L'opérande sous forme de flux serait réparti sur les nœuds selon l'attribut, spatial ou non spatial, correspondant à l'attribut relationnel entre les opérandes pour l'opérateur utilisé. Le traitement sera donc possible dès la réception de données sous forme de flux.

Finalement, pour le cas où les deux opérandes arrivent sous forme de flux, il est nécessaire, pour une plus grande efficacité, de joindre sur un même nœud les informations devant être traitées ensemble. Aussi, la fréquence des flux étant variable et non synchronisée, il devient nécessaire d'accumuler ces informations avant d'effectuer le traitement. Cette jointure est faite à l'aide de la technique des fenêtres qui a été mentionnée dans le chapitre précédent où les données sont accumulées pendant une certaine période avant d'être traitées.

2.1.2.2 La propriété quantitative des opérandes

Dans cette section, c'est l'aspect de la quantité de données qui sera abordé. Pour les données persistantes, on parlera de volume d'information, tandis que dans le cas de flux de données on parlera de son débit.

Clairement, cette recherche traite des données massives. Cependant, il n'est pas exclu que l'un des opérandes ne possède pas cette caractéristique et se situe plus dans le domaine des données traditionnelles. Le tableau suivant présente les trois cas de figure possibles.

Tableau 5 Caractéristique quantitative des données

Propriété quantitative (opérateur binaire)	Description
Mégadonnées – Données	Une géométrie a un débit ou volume élevé tandis que l'autre a un débit ou volume faible.
Mégadonnées – Mégadonnées	Les volumes ou débits des géométries sont tous élevés.
Données – Données	Cas de traitement par des technologies traditionnelles.

Le cas où les deux opérandes ne sont pas issus de données massives n'est pas considéré, car il n'entre pas dans la catégorie des mégadonnées. Cette situation est traitée à l'aide de systèmes traditionnels.

Dans le cas où un opérande est de faible volume ou débit, il sera peut-être plus facile de rendre disponible celui-ci à chacun des nœuds, facilitant ainsi le traitement. Les données massives pourront alors être réparties aléatoirement sur les nœuds puisqu'une copie des données traditionnelles s'y trouvera.

Le cas où les deux opérandes sont des mégadonnées est plus complexe. Une répartition judicieuse des données doit être faite pour minimiser la duplication de données massives ainsi que les transferts entre les nœuds.

2.2 Classification

Chacune des caractéristiques précédentes représente différentes propriétés des cas de figure rencontrés lors du traitement spatial de flux massifs. Ces propriétés sont des indicateurs pointant vers des techniques permettant d'adapter les opérateurs au contexte spécifique de l'application. La Figure 7 illustre les différents cas de figure possibles selon les propriétés de l'opérateur spatial de base et du contexte l'entourant ainsi que le niveau relatif de complexité les uns par rapport aux autres. Les sections suivantes abordent chacun de ces cas afin de proposer des techniques générales d'adaptation et d'évaluer le niveau de complexité.

La façon d'aborder le problème du traitement spatial est influencée par la combinaison des propriétés, soit l'arité, la quantité, la temporalité et la nature de l'attribut relationnel. Chacune des combinaisons représente un cas d'étude pour lequel nous allons proposer une méthode générale permettant d'effectuer le traitement.

Seuls les cas impliquant des mégadonnées en flux ont été retenus dans ce schéma. Les autres ne correspondent pas au cadre de cette recherche.

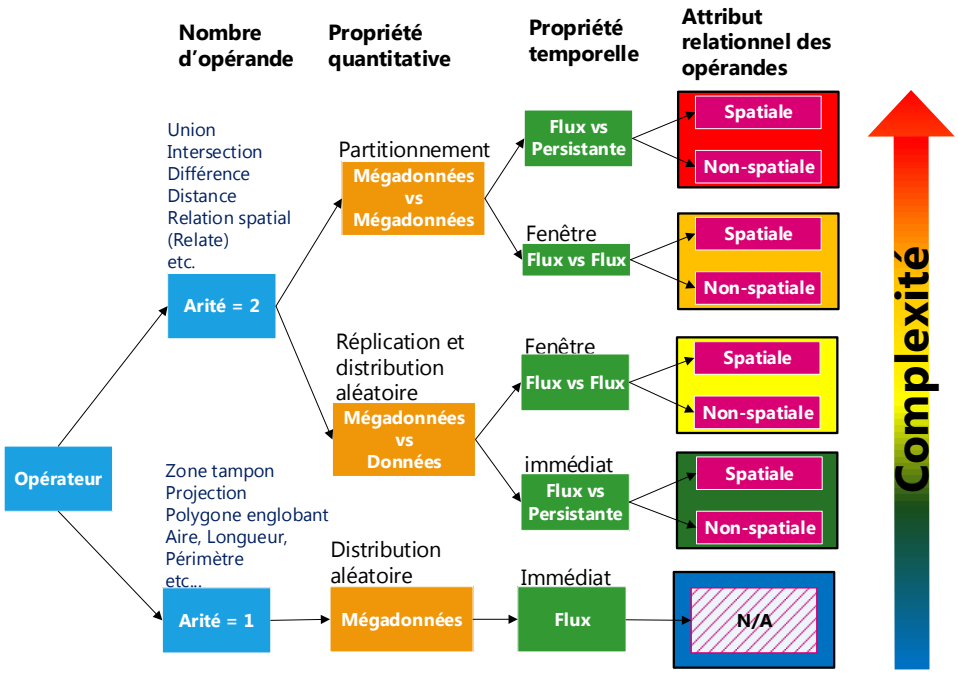


Figure 7 Classification des cas d'études

2.2.1 Cas des opérateurs unaires

Dans le cas des opérateurs unaires, opérateurs à un seul opérande, le traitement peut donc être effectué dès l'arrivée de la donnée. Les données peuvent être réparties également et aléatoirement sur les nœuds. Par exemple, la création d'une zone tampon, le calcul de l'aire d'un polygone ou le changement de système de référence d'un attribut spatial ne requièrent que l'entité spatiale elle-même. L'opérateur peut donc être appliqué sur celle-ci dès son arrivée dans le système. Le flux de mégadonnées sera réparti aléatoirement ou de façon équilibrée sur les différents nœuds. Puisqu'il n'y a qu'un seul opérande spatial, la propriété de contexte relationnel ne s'applique pas. Ce cas de figure pose peu de défi en ce qui concerne le traitement en parallèle.

2.2.2 Cas des opérateurs binaires

Dans le cas des opérateurs binaires, il y a deux opérandes spatiaux. Il faut donc joindre ceux-ci pour appliquer l'opérateur. Une partie de la solution consiste à partitionner les données sur le système de façon à ce que les données devant être traitées ensemble par un opérateur soient sur le même nœud. Le cas des opérateurs binaires représente déjà une complexité plus grande que celui des opérateurs unaires.

Ces techniques de partitionnement des données sont toutefois dépendantes des autres propriétés entourant le contexte du traitement spatial. Les sections suivantes décrivent ces différents cas.

2.2.2.1 Attribut relationnel entre les opérandes

Avant de décrire chacun des cas présentés dans la figure ci-dessus, il est intéressant de comprendre le concept d'attribut relationnel entre les opérandes.

Lorsque deux données doivent être jointes pour leur appliquer un opérateur binaire, il faut définir le critère de jointure. Selon le contexte du problème à résoudre, il sera nécessaire que les données partageant des caractéristiques semblables soient regroupées ensemble. Imaginons qu'il soit nécessaire d'obtenir une ligne représentant le trajet des 5 dernières minutes de véhicules en déplacement. Si le système obtient les informations de centaines de milliers, voire de millions de véhicules chaque seconde, il est légitime de penser qu'il faudra traiter ces informations en parallèle à l'aide d'une grappe d'ordinateurs. Il faudra donc s'assurer que les informations provenant d'une même automobile soient traitées par le même nœud. Toutes ces données pourraient être regroupées par l'identifiant du véhicule. Bien qu'un nœud ne soit pas limité à traiter qu'un seul véhicule, il aurait en sa possession toutes les données liées aux véhicules qu'il doit traiter. La caractéristique de regroupement est basée sur la valeur d'un attribut (ex. l'identifiant du véhicule) ou une valeur calculée à partir d'un attribut.

Dans le cas qui nous intéresse, c'est-à-dire les traitements spatiaux, les relations spatiales peuvent être basées sur des attributs non spatiaux ou spatiaux. L'exemple cité montrait un traitement permettant de produire une nouvelle géométrie. L'exemple suivant utilise un attribut relationnel de nature spatiale. Poursuivant l'exemple débuté plus haut, supposons maintenant qu'après avoir obtenu les trajets pendant la période de 5 minutes, nous désirons identifier les véhicules dont les trajets se sont croisés. Pour l'instant, les lignes sont regroupées sur les nœuds selon l'identifiant du véhicule. Ceci ne garantit pas que toutes celles qui se croisent soient sur un même nœud. À cette étape-ci, il faut donc réunir celles qui ont le plus de chance de se croiser sur un même nœud. Ce sont celles qui partagent une proximité spatiale. La caractéristique de regroupement ou de partitionnement sera donc basée sur un attribut spatial, sa localisation dans l'espace. Cet exemple montre l'importance de tenir compte des attributs relationnels des opérandes dans le traitement d'un opérateur spatial.

2.2.2.2 Propriété quantitative mégadonnées-données traditionnelles

Dans cette famille de cas, l'un des opérandes est issu de mégadonnées. Il est donc nécessaire de partitionner ces mégadonnées sur les différents nœuds du système. L'autre opérande provient de données traditionnelles. Ce type de données peut sans difficulté être pris en charge par un seul ordinateur. Il sera donc possible de répliquer ces données traditionnelles sur l'ensemble des nœuds. De cette façon, l'opérande de type mégadonnées aura accès à l'ensemble des données traditionnelles. Il est donc possible de répartir aléatoirement les mégadonnées sur le système puisque toutes les données traditionnelles seront disponibles sur chacun des nœuds. Imaginons que l'ensemble de la population d'une ville est suivi et que l'on veuille savoir quelles personnes se trouvent à proximité d'un point d'intérêt (fixe/persistant ou mobile/flux). Si la quantité de

points d'intérêts est limitée, il serait possible que chacun des nœuds possède toutes les informations des points d'intérêt. Dès lors, le partitionnement des mégadonnées peut être aléatoire.

Toutefois, il serait possible, pour des questions de performance, d'effectuer un partitionnement des données traditionnelles ainsi que des mégadonnées. Ce cas serait similaire au cas où la propriété quantitative serait de type mégadonnées-mégadonnées du point de vue de la technique de partitionnement.

L'influence des propriétés temporelles dans ce cas de figure est analysée dans les deux sections suivantes.

2.2.2.2.1 Propriété temporelle Flux vs Flux

Lorsque les deux opérandes sont sous la forme de flux, rien ne garantit que leur arrivée dans le système soit synchronisée et que leurs fréquences soient identiques et régulières. Il faudra néanmoins réussir à joindre ces données pour pouvoir effectuer le traitement. La technique des fenêtres décrite précédemment permet d'effectuer cette jointure en accumulant un certain nombre de données ou en les accumulant pendant un certain temps avant de déclencher le traitement.

Le flux issu de données traditionnelles serait transmis à chacun des nœuds, tandis que le flux issu de données massives serait partitionné aléatoirement sur les nœuds. Au terme de l'accumulation (la fin de la fenêtre) l'opérateur sera en mesure d'effectuer le traitement. Dans l'exemple précédant, si les points d'intérêts étaient mobiles, par exemple des cantines mobiles, ce flux serait alors dupliqué sur chacun des nœuds et mis en relation avec un flux massif de données représenté par une population en déplacement. Les individus seraient avisés lorsqu'une cantine mobile est à une certaine distance d'eux.

2.2.2.2.2 Propriété temporelle Flux vs Persistante

Lorsqu'un des opérandes est sous la forme de données traditionnelles persistantes, celles-ci peuvent être initialement chargées sur chacun des nœuds et être disponibles pour le traitement dès l'arrivée du flux de mégadonnées. Il n'est donc pas nécessaire d'utiliser le traitement par fenêtre et le partitionnement des mégadonnées peut être fait aléatoirement sur les nœuds. En utilisant l'exemple précédant, les points d'intérêts sont fixes, par exemple des commerces. Les individus passant à proximité de ces commerces sont identifiés et avisés des promotions en cours. Ce cas est similaire au cas des opérateurs unaires dans le sens où les traitements peuvent être effectués dès l'arrivée des données en flux.

2.2.2.3 Propriété quantitative mégadonnées-mégadonnées

Jusqu'à maintenant, il était possible de distribuer aléatoirement les mégadonnées sur le système, soit parce que l'opérateur était unaire, soit parce que l'un des opérandes était issu de données traditionnelles et pouvait donc être répliqué sur chacun des nœuds du système. Dans cette classe de cas, les deux opérandes sont issus de mégadonnées. Il n'est donc pas possible de répliquer l'une des sources sur tous les nœuds et un partitionnement

non aléatoire devient nécessaire. Ce partitionnement est fait selon l'attribut relationnel entre les deux opérandes. Cet attribut peut être, rappelons-le, de nature spatiale ou non spatiale. L'analyse et l'élaboration de ces méthodes de partitionnement sont abordées plus en détail ultérieurement. Ce partitionnement non aléatoire implique un niveau de complexité supérieur en comparaison avec les cas de figure précédents.

L'influence des propriétés temporelles pour ces cas de figure est analysée dans les deux sections suivantes.

2.2.2.3.1 Propriété temporelle Flux vs Flux

La synchronicité entre les flux et la stabilité de la fréquence d'un flux ne peuvent être garanties. Pour ces raisons, un opérateur binaire devrait attendre un certain délai, en temps ou en quantité de données, avant de pouvoir effectuer le traitement sur les données ainsi accumulées. C'est encore une fois la technique de fenêtre qui sera utilisée. Dans ce cas où les deux opérandes sont issus de flux massifs, il faut donc utiliser la technique des fenêtres en conjonction avec une méthode de partitionnement.

C'est cette classe qui sera détaillée ultérieurement dans ce mémoire, car elle représente une complexité nouvelle par rapport à ce qui a été fait dans d'autres travaux.

2.2.2.3.2 Propriété temporelle Flux vs Persistante

Dans ce dernier cas de figure, l'un des opérandes est disponible à l'avance. Cependant, il est issu de mégadonnées. Dans un système de traitement de données massives persistantes, les données sont généralement distribuées aléatoirement. Cependant, plusieurs travaux ont démontré l'avantage du partitionnement de données spatiales pour certains types de traitement (Aji *et al.*, 2013; Eldawy *et al.*, 2013; Eldawy et Mokbel, 2013; He *et al.*, 2014; Whitman *et al.*, 2014). La technique proposée serait de partitionner les données persistantes sur les nœuds et de créer un index global. C'est la méthode déjà utilisée dans les systèmes persistants. Par la suite, le flux de données serait partitionné selon les mêmes critères. Les flux et données persistantes ayant les mêmes attributs relationnels entre opérandes seraient donc regroupés sur les mêmes nœuds. Cette méthode permettrait au flux d'être traité dès son arrivée dans le système puisque l'opérande statique est déjà disponible.

Une problématique survient lorsque plusieurs types d'opérations, spatiales ou non, doivent être effectuées entre ces deux types de jeux de données, mais que ces deux opérations requièrent des attributs relationnels entre opérandes différents. Par exemple, une jointure par attribut spatial puis une jointure par attribut non spatial. L'une des opérations sera plus efficace grâce à son indexation.

Une autre question intéressante est le cas où l'étendue géographique des données persistantes est beaucoup plus grande que celle du flux de données. Il pourrait en résulter une concentration du flux sur quelques nœuds seulement, ce qui réduirait l'efficacité globale du système comparée à sa capacité.

Une autre problématique de nature technique est la capacité à pouvoir diriger le flux vers un nœud spécifique qui contiendrait les données statiques. Durant la période où nous avons analysé différents systèmes, aucun ne permettait de partitionner des flux massifs et des données statiques massives pour les utiliser ensemble. Cependant, ces systèmes sont en évolution et cette problématique pourrait être résolue dans un avenir rapproché. Ce cas de figure est le plus complexe de tous ceux analysés jusqu'à maintenant.

2.3 Choix de la classe pour l'étude de cas

Suite à l'analyse et à la classification des différents cas de figure, il ressort deux cas présentant une complexité intéressante à étudier plus en profondeur. Ces cas, présentés ci-dessous, sont tous les deux dans la famille des opérateurs binaires ayant des opérandes issus de mégadonnées, soit :

- Flux issus de mégadonnées avec flux issus de mégadonnées;
- Flux issus de mégadonnées avec mégadonnées persistantes.

Ces deux cas demandent l'élaboration de techniques de partitionnement basées sur des attributs spatiaux. Le deuxième cas pose une problématique supplémentaire, puisque les systèmes étudiés ne permettaient pas ce type d'analyse au moment de leur évaluation. Il serait logique de débuter par une problématique commune aux deux cas cités. Une évaluation de l'ampleur du travail nous amène à choisir le premier cas pour la poursuite de l'analyse. À titre comparatif, les travaux faits par Galić en 2016 se situent dans les deux classes où les niveaux de complexité sont les moins élevés.

Les prochaines étapes analysent ce cas d'étude et proposent des techniques permettant d'effectuer des traitements spatiaux. Un certain nombre de traitements spatiaux spécifiques sont ensuite analysés dans le contexte du cas d'étude. Par la suite, un prototype est réalisé pour ces traitements afin de prouver la faisabilité et tester les méthodes proposées.

2.4 Analyse de l'étude de cas

La figure suivante représente le cas d'étude qui est analysé dans cette section. Ce cas représente les traitements spatiaux ayant des opérateurs binaires où les deux opérandes sont issus de flux de mégadonnées.

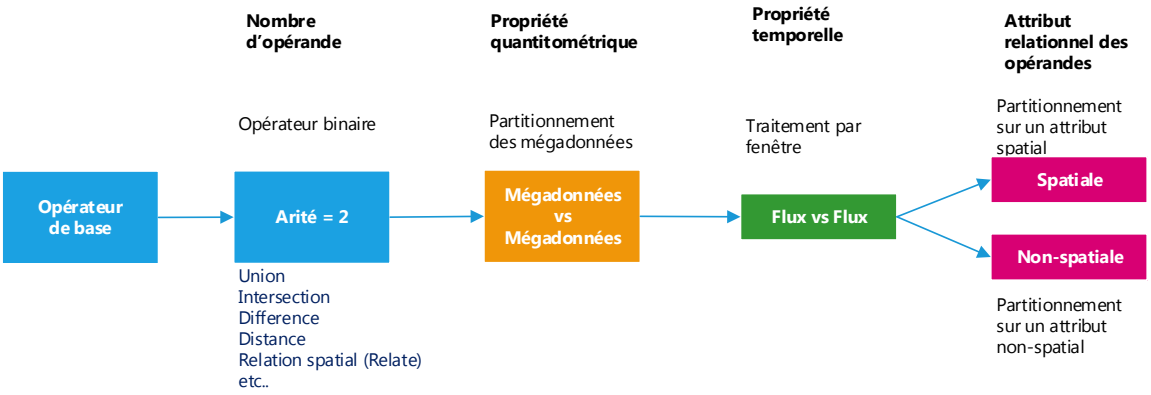


Figure 8 Cas d'étude - opérateur binaire entre flux massifs de données

Pour l'analyse effectuée dans ce travail, les attributs relationnels des opérandes de nature non spatiale ne seront pas étudiés. Ce type de partitionnement est à la base des systèmes de traitement en parallèle et est déjà connu. Seul le partitionnement basé sur un attribut spatial sera analysé.

Dans l'analyse sommaire de ce cas, nous avons déterminé que l'utilisation de la technique des fenêtres ainsi que le partitionnement selon des attributs relationnels des opérandes était la solution à envisager. La figure suivante illustre le processus de façon schématique. Dans l'exemple suivant, les données des flux sont accumulées pendant une certaine période. Lorsque la période se termine, la fenêtre de traitement est déclenchée et les données sont partitionnées spatialement. L'opérateur peut donc effectuer le traitement pour chaque combinaison fenêtre-partition. Le résultat est finalement émis sous la forme d'un flux de données. Les lettres représentent les données, les couleurs représentent des partitions spatiales communes (clés de hachage). Les différents alphabets représentent différentes sources d'informations ou de résultats (classes d'entités).

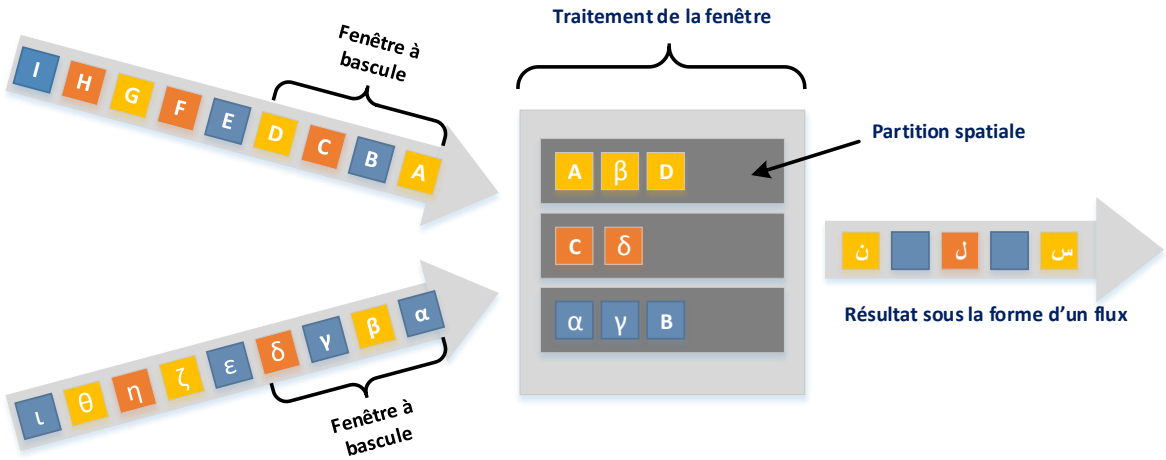


Figure 9 Traitement de flux par fenêtre et partitionnement spatial

2.4.1 Accumulation des flux par fenêtre

La première technique analysée est la technique d'accumulation des données par fenêtre. Cette technique a été décrite précédemment. Les données sont accumulées pendant une période et à la fin de cette période le traitement se déclenche. C'est lors du déclenchement que le partitionnement des données est effectué. Les partitions seront distribuées sur les différents nœuds. Le partitionnement spatial est abordé dans la section suivante.

2.4.2 Le partitionnement spatial

La deuxième technique analysée est le partitionnement spatial. L'objectif d'un tel partitionnement est de permettre le regroupement des entités partageant une proximité spatiale. Dans le cas des données persistantes, la solution apportée est la réorganisation des données sur les nœuds par la création d'index spatial global et local. L'effort requis par cette indexation est rentable par la suite pour optimiser les recherches d'information, telles les jointures spatiales. Dans le cas des données issues de flux, le gain suivant cet effort est moins évident. Comme les données perdent de leur valeur avec l'écoulement du temps et que de nouvelles données sont constamment introduites dans le système, l'index qui serait créé ne serait plus valide lors de la fenêtre de traitement suivante.

2.4.2.1 Options de partitionnement

Dans le cas qui est à l'étude, les données sont traitées à l'aide de fenêtre de traitement. Celles-ci sont donc accumulées pendant une certaine période avant d'effectuer une opération spatiale. Ce sont les données de cette fenêtre qui doivent être partitionnées spatialement. Dans les systèmes de gestion de base de données à référence spatiale, le partitionnement (indexation) des données est effectué sur la base actuelle de la distribution spatiale de celles-ci. Le patron de partitionnement (la structure de l'indexation) est construit à partir de toutes les données disponibles. Lorsqu'une requête est effectuée, le système navigue dans cette structure pour trouver et retourner le résultat. L'insertion de nouvelles données déclenchera une modification dans la structure de l'indexation.

Cependant, dans ce type de système, l'introduction de nouvelles données est très faible en comparaison avec les applications de flux massif de données. Puisque chaque fenêtre de traitement contient de nouvelles données, il faut donc recalculer l'indexation continuellement.

Une distinction est faite entre les termes indexation et partitionnement. Un index est une structure de données navigables permettant de retrouver des données. Les traitements de données massives fonctionnent en regroupant les données à l'aide d'attribut. On ne cherche donc pas des données, mais plutôt on cherche à les

regrouper pour effectuer des traitements. On désire donc partitionner ces données sur la base d'un attribut spatial.

La figure suivante montre deux classes de partitionnement. Le partitionnement prédéterminé et le partitionnement dynamique. Dans un partitionnement dynamique, le patron de partitionnement sera calculé à l'aide de la distribution spatiale des données de la fenêtre de traitement. Or, ces données sont distribuées sur de multiples nœuds. Cela implique une mécanique de calcul local et d'échange d'information entre les nœuds pour finalement assigner à chaque donnée son attribut de partitionnement. À chaque fenêtre de traitement, ce calcul devra être refait.

Dans un patron prédéterminé, le patron de partitionnement est défini à l'avance, ainsi chaque donnée se voit assigner un attribut de partitionnement. Puisque le patron est prédéfini, chacun des nœuds possède cette information avant même l'arrivée des données. Les données peuvent donc être assignées à leur attribut dès leur arrivée dans le système. Puisque cette assignation est indépendante de la distribution des autres données, elle peut donc être faite localement en une seule étape, ce qui facilite sa parallélisation.

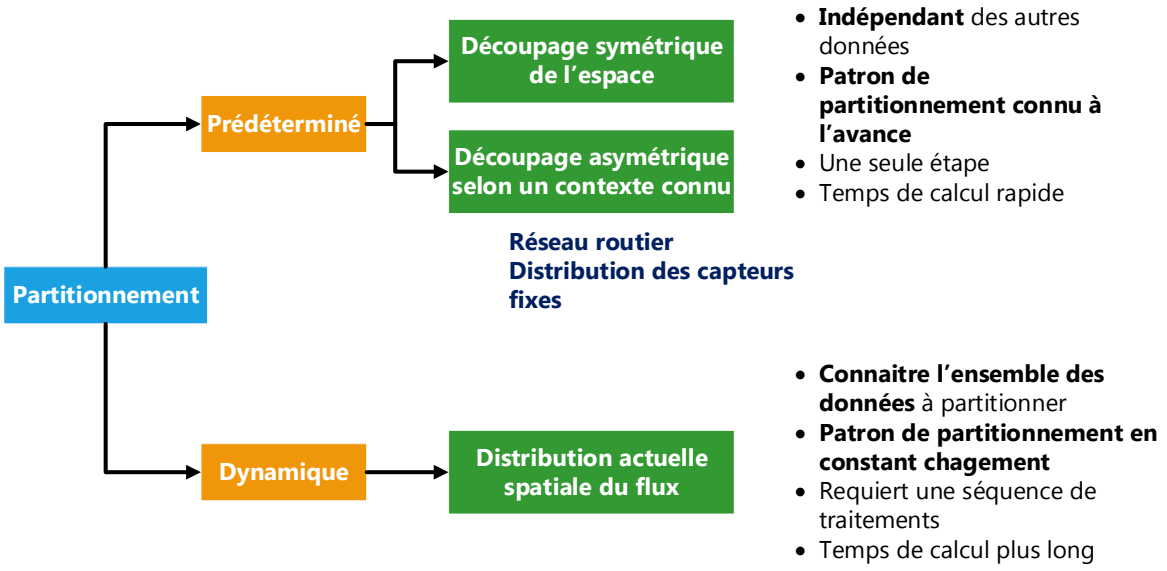


Figure 10 Options de partitionnement

Dans un partitionnement prédéterminé, l'espace peut être découpé de manière symétrique, comme les tuiles d'un jeu d'échecs, sans égard à la nature des données traitées. Le partitionnement pourrait aussi être asymétrique si la nature du jeu permet de relier conceptuellement les données dynamiques à un jeu de données statique. Par exemple, si le flux d'information provient d'un réseau de capteurs fixes, alors la distribution des données est connue à l'avance. Dans un autre exemple, si le flux de données provient de véhicules se déplaçant

sur un réseau routier, alors les différentes zones de densité du réseau routier pourraient servir de base pour le patron de partitionnement.

Comme aucun travail n'a encore été publié sur le partitionnement des flux massifs de données spatiales, toutes ces options restent à explorer. Dans le cadre de ce mémoire de recherche, nous allons explorer le découpage symétrique de l'espace ainsi qu'une version dynamique du partitionnement. Ces deux techniques de partitionnement sont basées sur une méthode de géo-hachage qui est décrite dans la section suivante.

2.4.2.2 Le partitionnement par clé de géo-hachage

Le type de partitionnement que nous allons étudier consiste à découper l'espace sous la forme d'une grille en utilisant la méthode de géo-hachage. Cette technique est basée sur la courbe de Lebesgue (Z-order curve) (Morton, 1966; Wikipedia, 2014). Elle divise l'espace en grille de dimension équivalente (en unité d'arc, dans le cas du géo-hachage).

Cette méthode permet d'obtenir une clé contenant n caractères basée sur les coordonnées d'une entité spatiale. À cette clé correspond une zone géographique. L'étendue de cette zone dépend du nombre de caractères de cette clé. Plus la zone est restreinte, plus la clé contiendra de caractère dû aux subdivisions successives. L'avantage d'une clé de hachage tient au fait que son calcul n'est pas dépendant du nombre de données, contrairement à une indexation de type arborescente (ex. KD-tree, QuadTree).

Les deux figures suivantes montrent différents niveaux de profondeur de géo-hachage. Les géo-hachages présentés sont effectués sur la base de 32 partitions (de 0 à z). Il est cependant possible de diminuer le nombre de partitions par niveau de profondeur en utilisant une base plus petite. La croissance étant exponentielle, sous la forme de $base^{profondeur}$, un moins grand nombre de partitions sera créé pour un niveau de profondeur avec une base plus petite. Il en résultera des partitions ayant des dimensions plus grandes.

La profondeur est représentée par le nombre de caractères de la clé de hachage. Par exemple, la clé « f244tx » représente une tuile ayant une profondeur de 6 niveaux. Dans le découpage présenté aux figures suivantes, il y a une possibilité de 32 tuiles par niveau de hachage, soit 4 lignes et 8 colonnes. Le plan est divisé en deux (droite gauche ou haut bas) de façon successive. À chaque division, une valeur de 0 ou 1 est ajoutée à la clé selon que l'entité se trouve dans la division de gauche/haut (0) ou celle de droite/bas (1). Dans une division à 32 tuiles, il faudra effectuer 5 divisions (trois verticaux et deux horizontales) pour isoler la tuile contenant l'entité spatiale. La clé de hachage, pour ce niveau, sera donc un nombre binaire de 5 chiffres qui équivaut à un nombre décimal entre 0 et 31. Par exemple, le nombre binaire 01011 équivaut au nombre 11 en décimale. Ce nombre est ensuite associé à un caractère afin de faciliter la lecture. Par exemple, la tuile 11 (donc la 12e, car la première est la tuile #0) est associée au caractère « C ». Cette tuile est à nouveau divisée en 32. Ce processus est répété

pour chaque niveau de profondeur. Il en résulte donc une série de caractères comme clé de hachage. Le nombre de découpages pourrait facilement être modifié, par exemple si l'on désire un maximum de 8 tuiles par niveau au lieu de 32, alors la plage de caractères serait modifiée pour avoir un domaine de 8 caractères. Cependant, il faudrait avoir une plus grande profondeur pour avoir des tuiles de grandeur équivalant à celles d'un découpage de 32 tuiles, mais la progression du nombre de tuiles total, en fonction de la profondeur, serait plus faible.

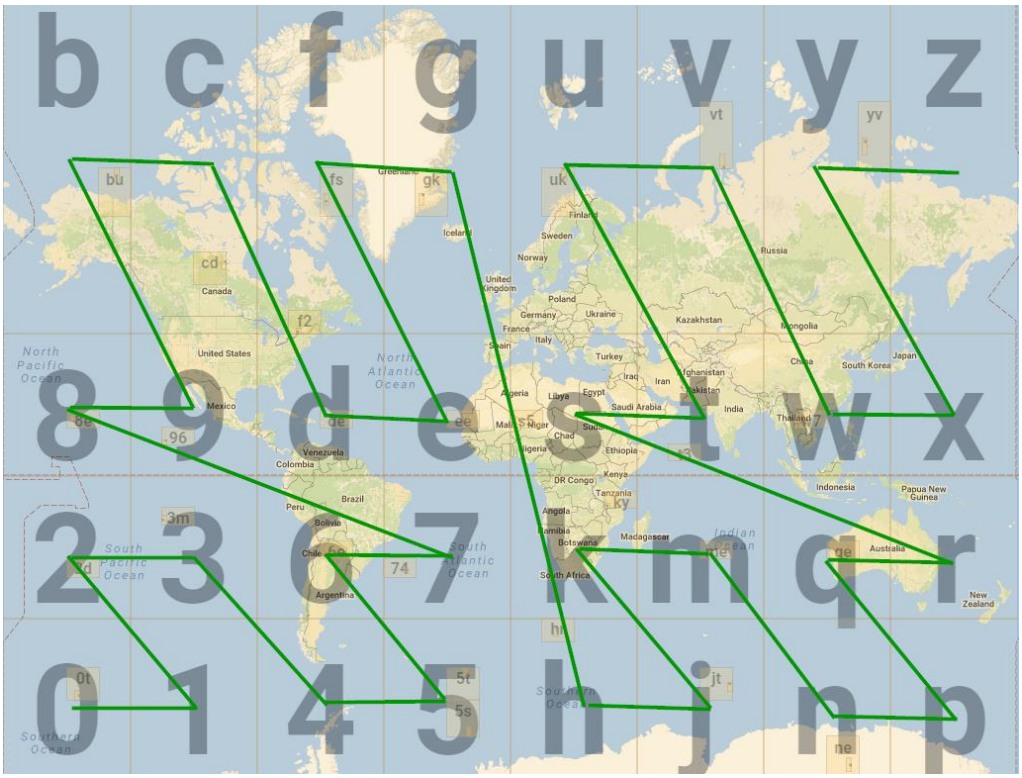


Figure 11 Géo-hachage sur une base 32 de profondeur 1

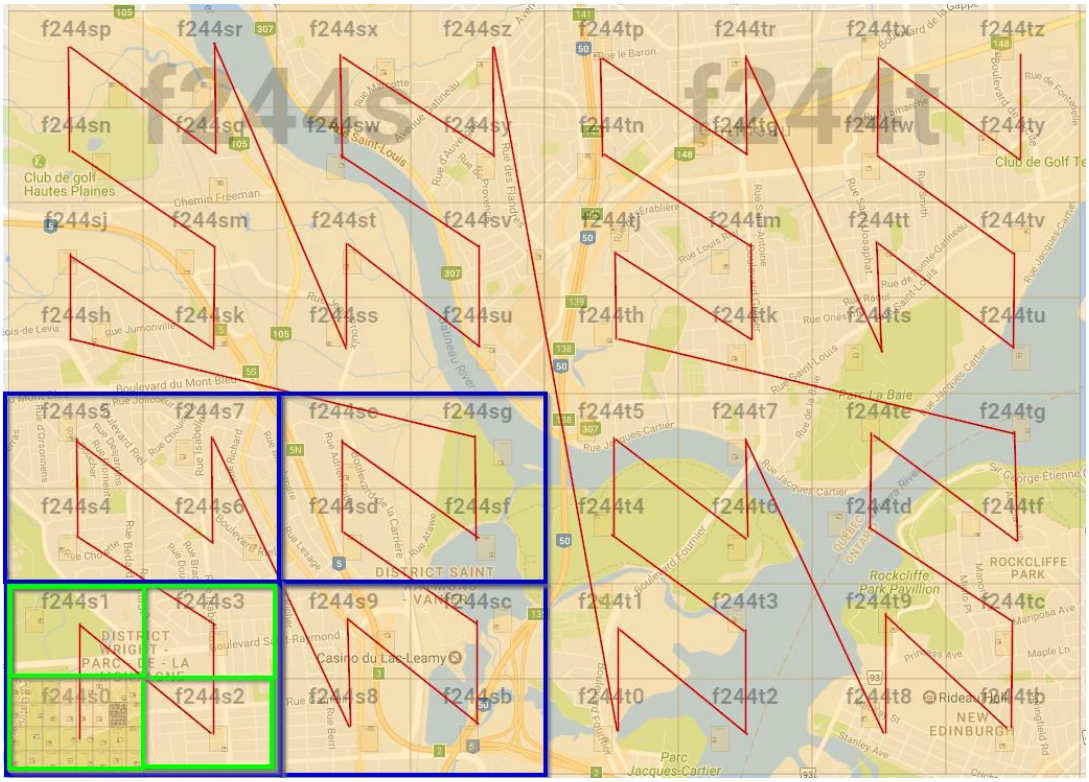


Figure 12 Géo-hachage sur une base 32 de profondeur 6

Cette technique de hachage permet de rapidement calculer une clé qui sera l'attribut relationnel des opérandes utilisé pour regrouper les données. Toutes les entités partageant une proximité spatiale posséderont la même clé et seront assignées à la même partition géographique. Les entités linéaires et surfaciques se trouvant sur plus d'une partition auront plusieurs clés. Celles-ci apparaitront donc dans plusieurs groupes, comme l'illustre la figure suivante.

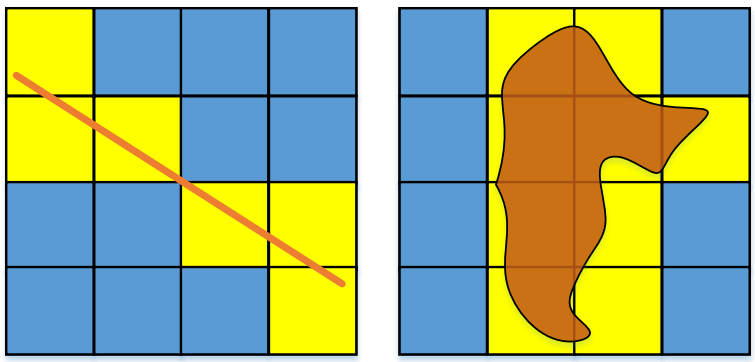


Figure 13 Partition multiple pour les entités chevauchant plusieurs partitions

Lors du partitionnement, il faut tenir compte des relations topologiques et aussi des relations de proximité. Dans le cas des relations topologiques, toutes les entités spatiales ayant une relation d'intersection partageront une même partition, donc une même clé de géo-hachage. Il sera donc possible de calculer exactement ces relations.

Pour les relations de proximité, une problématique survient. Les entités à proximité des frontières des cellules de géo-hachage seront ignorées puisqu'elles ne sont pas dans la même partition. Dans la figure suivante, les 3 plus proches voisins du point 1 sont les points 5, 6 et 7. Cependant, ces derniers ne partagent pas la même clé que le point 1. Comme les entités sont regroupées par clé de partitionnement, les trois plus proches voisins du point 1 seraient 2, 3 et 4, ce qui est erroné.

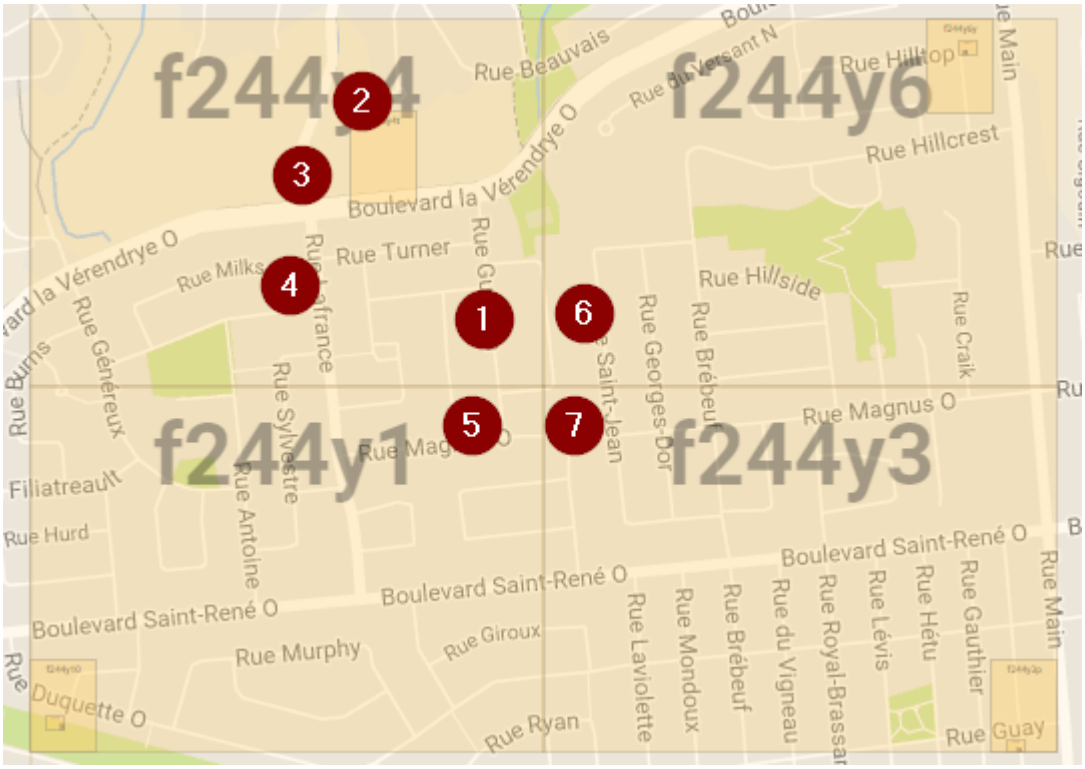


Figure 14 Problématique des frontières pour les relations de proximité

La solution consisterait à effectuer un certain nombre de translations virtuelles sur chacune des entités. Ces translations virtuelles ne serviraient qu'à calculer de nouvelles clés de hachage et non à déplacer « physiquement » l'entité. Les entités pourraient alors posséder plusieurs clés et donc faire partie de plusieurs partitions. La figure suivante illustre ces translations virtuelles.

Figure 15 Translations virtuelles pour les relations de proximité

Cette méthode a pour désavantage la multiplication des données. Le nombre de translations doit donc être calculé de façon optimale pour limiter la multiplication des clés de géo-hachage et des données. En effet, celles-ci étant réparties sur les différents nœuds sur la base de clé, elles devront être dupliquées. Dans certaines requêtes de proximité, il est courant de limiter la recherche dans un certain rayon. Les prochaines sections explorent le cas des "k" plus proches voisins.

Dans le cas où une analyse de proximité ne serait pas limitée par un rayon, le problème se complexifie. Les données étant partitionnées dans une grille, il faudrait chercher récursivement dans les grilles mères de plus haut niveau et adjacentes si le nombre de données recueillies n'atteint pas le nombre de valeurs désirées. Une autre méthode serait de construire un index (R-Tree par exemple) permettant de limiter le nombre de recherches. Rappelons que ces analyses portent sur des opérandes issus de flux de mégadonnées. L'indexation répétitive à chaque nouvelle fenêtre et la recherche récursive pourrait décupler le temps de calcul et la quantité de données échangées sur le réseau. Dans le cadre de ce travail, cette problématique n'est pas explorée.

Dans la section suivante, nous allons voir comment il est possible d'appliquer le géo-hachage à des traitements spatiaux. Dans un premier temps, nous analyserons le cas d'une relation topologique d'intersection et par la suite, le cas d'une relation de proximité à l'aide d'une analyse des « k » voisins les plus proches limités à l'intérieur d'un rayon.

2.5 Application du géo-hachage sur des traitements spatiaux

Deux cas spécifiques sont analysés dans cette section afin de tester les méthodes proposées plus haut. Le premier cas est une analyse topologique sur des opérandes issus de flux de mégadonnées. Le deuxième cas est une analyse de proximité simple de type "k" plus proches voisins limités à un rayon.

2.5.1 Cas d'étude sur un traitement de relation topologique

Dans ce premier traitement, le système reçoit un flux massif de données représentant la position de véhicules se déplaçant sur un réseau routier. Chacun des véhicules transmet son identifiant, sa position, sa direction et sa vitesse. On désire déterminer les trajets s'intersectant durant une certaine fenêtre de temps, par exemple les intersections durant les 30 dernières secondes. Dans l'exemple suivant, les informations des véhicules sont collectées à la seconde. Dans la Figure 17, les véhicules sont représentés par une barre de couleur (ex.). Les sources d'informations peuvent être multiples, c'est pourquoi le schéma en représente plusieurs. Par la suite, ces données sont accumulées à l'aide d'une fenêtre et groupées par identifiant pour former une ligne. Dans cette fenêtre, un opérateur crée une ligne à partir des points ayant le même identifiant. Cette opération est un exemple de partitionnement à l'aide d'un attribut non spatial. À la sortie de cette opération, des trajets sous la forme de lignes sont obtenus.

L'opération suivante consiste à calculer les clés de partition pour chacun des trajets. Chacune de ces partitions possède une clé de géo-hachage unique représentée par les chevrons de couleur (ex.). Chacune des lignes fera partie d'une ou plusieurs partitions. La Figure 16 illustre les partitions et les lignes représentant les trajets.

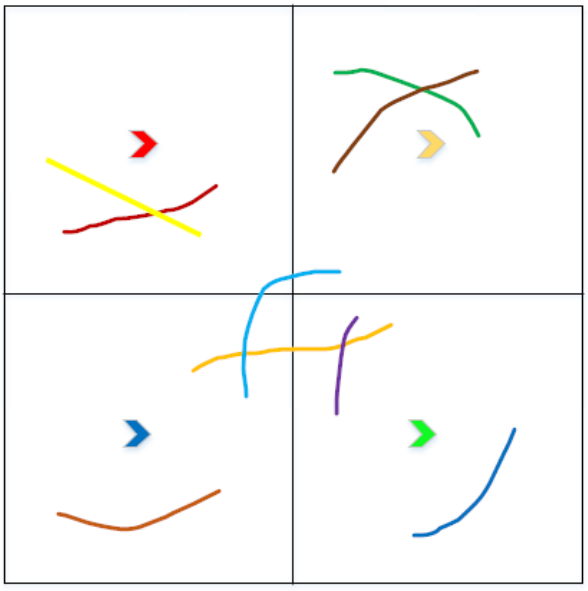


Figure 16 Partition et trajet

Les lignes sont alors regroupées ensemble pendant une fenêtre de temps selon leurs clés de partitionnement spatial. Finalement, les intersections sont calculées dans chacune des partitions et le résultat est transmis vers un système de sortie. Cette opération est schématisée dans la dernière section de la Figure 17.

L'opérateur binaire d'intersection prend deux opérandes issus du même flux. Les opérandes auraient aussi pu être issus de deux flux différents.

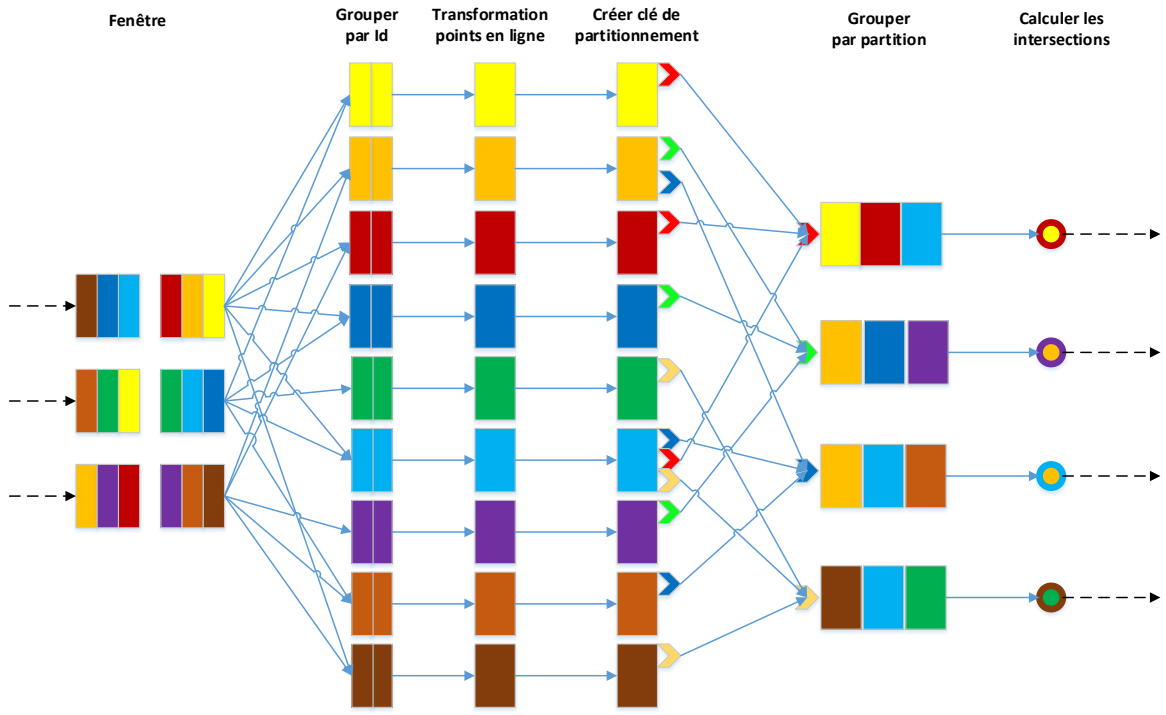


Figure 17 Séquence de traitement pour l'intersection de véhicule

2.5.2 Cas d'étude sur une relation de proximité

Dans ce deuxième traitement, le système doit déterminer les 3 plus proches voisins dans un certain rayon. La même source que l'exemple précédant est utilisée.

Le résultat désiré devra respecter les conditions suivantes. Pour chacun des véhicules, les trois voisins les plus proches seront calculés durant les 10 dernières secondes. Puisqu'un véhicule aura plus d'un point par fenêtre de 10 secondes, seule la distance minimale entre chaque paire de véhicules sera considérée. Le rayon de recherche maximal sera de 30 mètres.

Afin de simplifier les schémas suivants, seul un point par véhicule sera représenté. Ces conditions ne sont établies que pour les besoins de l'exemple.

Dans cet exemple, il faut appliquer la méthode de partition avec translation virtuelle, car c'est un cas de relation de proximité. L'étape suivante explique la façon d'effectuer ces translations.

Une méthode intuitive serait d'effectuer 8 translations de 30 mètres, une à chaque 45° en débutant avec l'angle de 0° pour chacun des points. Les points étant à une distance de 30 mètres ou moins d'une bordure (zone gris foncé sur la figure) obtiendront des clés de géo-hachages correspondant aux partitions voisines. Les points en bordure seront alors considérés dans la solution. Cette méthode crée plusieurs doublons, de 2 à 4 clés par entité contenues dans la zone grise. De plus, chacun des voisins à l'intérieur d'une distance de 30 mètres de sa bordure créera aussi des doublons. Par exemple, pour les quatre points présentés dans la Figure 18, chacun aura 4 clés, donc 16 points seront générés au total, dont 12 doublons.

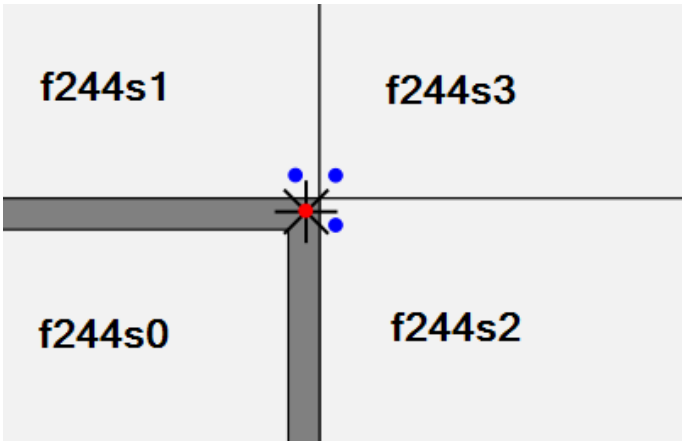


Figure 18 Méthode des 8 translations

La dimension des partitions est aussi un élément important. Elles ne doivent pas être inférieures à la longueur du rayon de recherche, sinon les translations pourraient enjamber une partition. De plus, si les partitions sont trop petites, une plus grande proportion des entités créera des doublons, car il y aura plus de points à la proximité des bordures. De trop grandes partitions diminueront les gains de traitement en parallèle. Il faut donc balancer avec soins le nombre de partitions.

Une autre méthode de translation permettant de minimiser les doublons consisterait à effectuer seulement deux translations virtuelles obliques dans une des directions horizontales ou verticales (ex. nord-est, sud-est) d'une distance de $r\sqrt{2}$. De cette façon, deux entités étant à une distance *r* l'une de l'autre se trouveraient toujours réunies dans une même partition. Cette partition pourrait être indigène (partition d'origine) de l'une des entités ou exotique (partition résultant d'une translation virtuelle et qui n'est pas la partition d'origine) aux deux entités.

Cette technique crée, pour quatre points situés dans un coin de leur partition, 8 doublons au maximum, donc 33 % moins que la méthode précédente. La Figure 19 représente cette méthode.

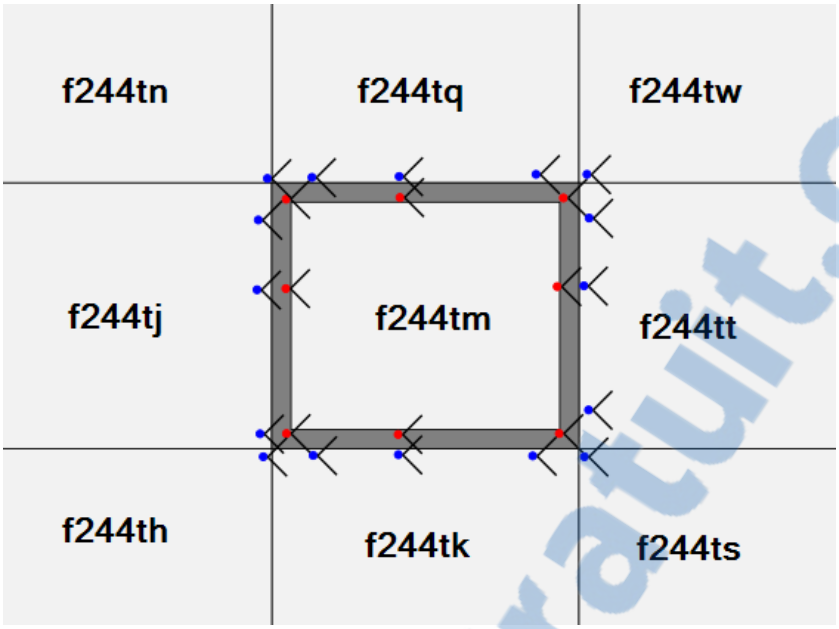


Figure 19 Méthode des 2 translations

La distance de translation de $r\sqrt{2}$ a pour but de s'assurer que la translation ait au moins une distance de *r* en *x* et *y*, comme l'illustre la figure suivante où $r\sqrt{2} = \sqrt{r^2 + r^2}$.

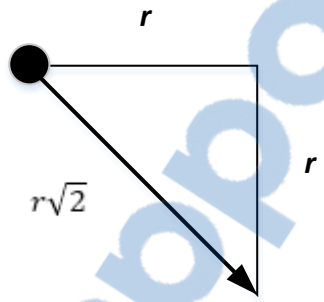


Figure 20 Distance pour la méthode à 2 translations

Reprenons l'exemple initial illustré par la Figure 21 et la Figure 22 . Les clés de géo-hachage sont associées à chacun des points. Après l'accumulation des données à l'aide de la technique des fenêtres, les données sont groupées en fonction des clés calculées. Rappelons que d'une à quatre clés par entité peuvent être créées.

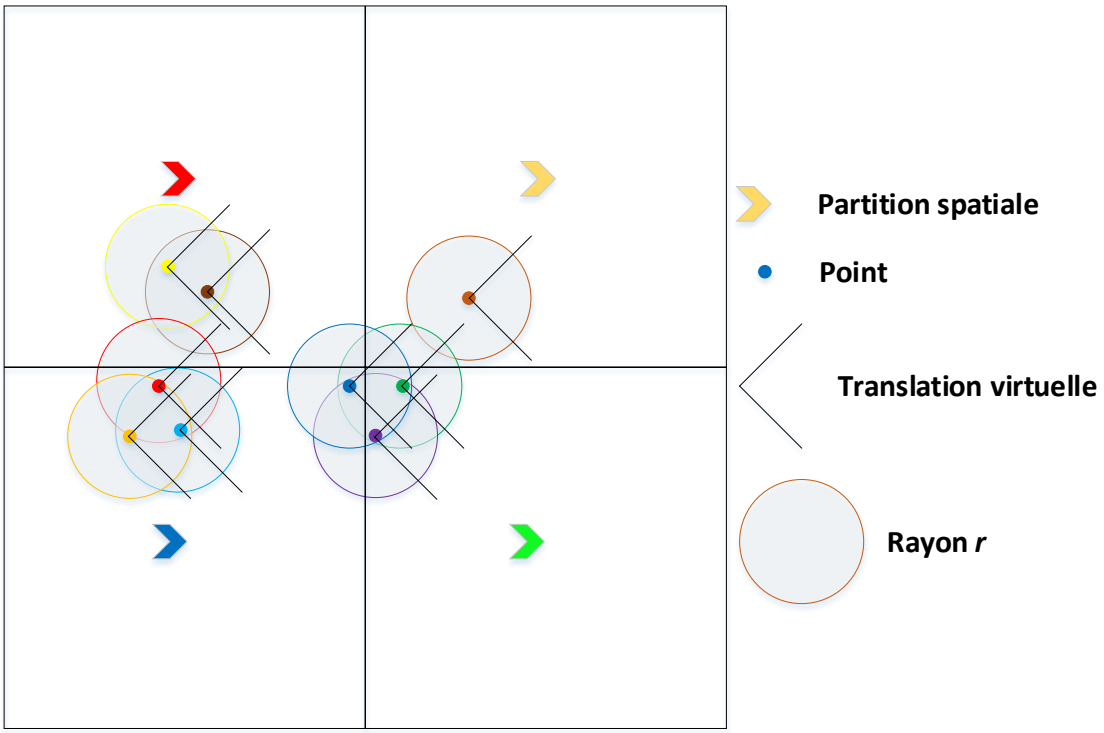


Figure 21 Point, translation et partition pour le knn

Ces étapes sont représentées par les premières parties du schéma. Lors du déclenchement de la fenêtre de traitement, la distance de chaque paire est calculée. Un index local, de type KD-Tree, pour calculer les knn pourrait être utilisé. Cet index local ne relève pas d'une stratégie de traitement en parallèle et ne sera donc pas discuté.

Suite à ce calcul, les résultats sont groupés par identifiant du véhicule et les "k" voisins les plus près sont conservés. Finalement, les résultats sont transmis vers un système de sortie.

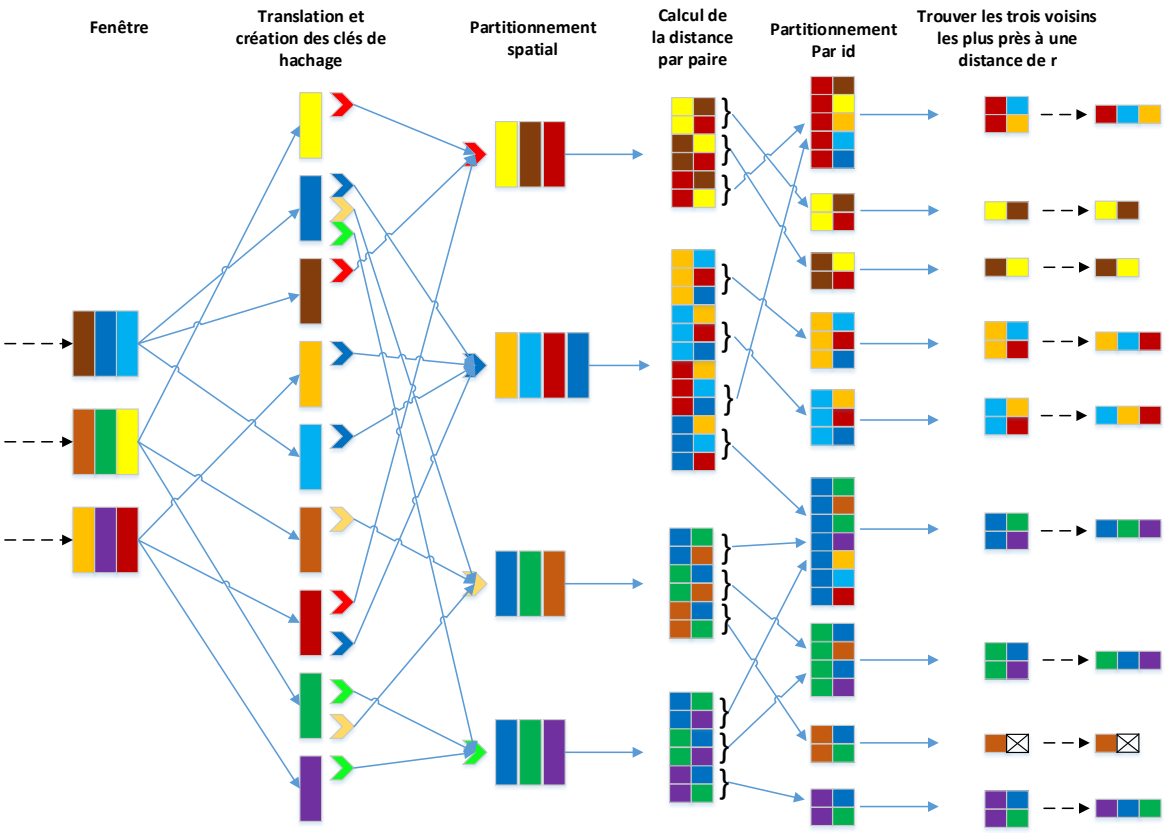


Figure 22 Séquence de traitement pour les knn

2.6 Inventaire des caractéristiques requises pour le système de traitement spatial de flux massifs

Les caractéristiques énoncées dans la section 1.2.1 ainsi que l'analyse des différents cas effectuée dans la section précédente ont permis de dégager certaines techniques requises pour l'adaptation du traitement de flux spatial dans un contexte de données massives. Les caractéristiques nécessaires sont les suivantes :

- Distribution et partitionnement aléatoire et balancé des données sur les différents nœuds du système;
- Partitionnement sur la base d'attribut spatial;
- Partitionnement sur la base d'attribut non spatial;
- Accumulation de données à l'aide de fenêtre;
- Permettre la jointure de deux flux massifs de données;
- Permettre la jointure d'un flux massif de données et d'un flux de données traditionnelles;
- Permettre la jointure d'un flux massif de données et de données massives persistante;

- Permettre la jointure d'un flux massif de données et de données persistantes traditionnelles;
- Permettre de diffuser des données persistantes traditionnelles à tous les nœuds;
- Permettre de diffuser des flux de données traditionnelles à tous les nœuds;
- Permettre le traitement immédiat de flux de mégadonnées avec un faible temps de latence
- Permettre l'arrivée des données dans un désordre potentiel.

Ces caractéristiques seront utilisées lors du choix technologique pour l'élaboration d'un prototype permettant de prouver la faisabilité et l'efficacité des méthodes proposées.

2.7 Conclusion

Ce chapitre a permis de dégager les caractéristiques influençant les méthodes d'adaptation du traitement spatial de flux de données. Ces caractéristiques sont :

- L'arité géométrique des opérateurs spatiaux;
- L'attribut relationnel entre les opérandes des opérateurs spatiaux;
- La propriété quantitative de la donnée, massive ou traditionnelle;
- La propriété temporelle de la donnée, persistante ou en flux.

L'analyse de ces caractéristiques a permis de déduire des techniques nécessaires pour effectuer les traitements. La répartition des données sur tous les nœuds, le partitionnement des données et l'utilisation de fenêtre sont quelques exemples des techniques proposées.

Une classification permettant de choisir les techniques de traitement selon le contexte des données et des opérateurs a été présentée ainsi qu'un classement selon le niveau de complexité. La classe de traitement impliquant des opérateurs binaires sur des opérandes issus de flux massif a été choisie afin d'explorer plus en détail les techniques de traitement. Les traitements clés pour cette classe ont été analysés et des solutions ont été proposées, notamment des techniques de partitionnement spatial. Par la suite, deux cas d'étude ont été analysés et des algorithmes ont été proposés pour effectuer le traitement en parallèle. Les cas étudiés sont un opérateur d'intersection et une analyse de proximité sous la forme des k plus proches voisins limités à un certain rayon.

Finalement, les caractéristiques requises d'une plateforme de traitement de flux massif ont été énoncées dans l'optique de la mise en œuvre d'un prototype devant démontrer la faisabilité et l'efficacité de la méthode proposée. Le prochain chapitre traitera de l'élaboration du prototype et permettra de matérialiser les opérateurs étudiés.

Chapitre 3 Développement d'un prototype

Afin de démontrer la faisabilité et l'efficacité des techniques de traitement proposées au chapitre précédent, un prototype a été conçu. Dans un premier temps, une technologie répondant le plus près aux caractéristiques établies par l'analyse des besoins a été choisie. Ensuite, une architecture permettant le traitement de flux massif a été créée. Puis, les algorithmes permettant d'effectuer les divers traitements proposés dans les cas d'étude ont été implantés. Finalement, des tests de performance ont été effectués sur les cas d'études. Ces tests visent à déterminer l'efficacité des techniques proposées en fonction du géo-hachage et du niveau de parallélisation.

3.1 Choix de la plateforme technologique

Afin de réaliser un prototype, il est nécessaire de choisir une plateforme permettant le traitement de flux massif de données. Des plateformes ont été évaluées en fonction de leur maturité et de leurs aptitudes à répondre aux caractéristiques énoncées à la fin du chapitre 2. Trois plateformes ont été retenues pour cette évaluation, car elles étaient les plus citées pour le traitement de flux massif de données. Ces plateformes sont : *Storm, Spark et Flink*. Toutes ces plateformes sont des projets *Open Source* de la famille *Apache*.

Dans un premier temps, les trois plateformes sont brièvement décrites et ensuite elles sont comparées entre elles au regard des caractéristiques énoncées au chapitre 2. Le lecteur doit garder à l'esprit que ces technologies évoluent très rapidement et que la description qui en est faite représente les fonctionnalités disponibles au moment de l'évaluation. Aussi, ces plateformes tendent à converger au niveau de leur fonctionnalité. À titre d'exemple, au début de cette recherche, en 2014, aucune de ces technologies n'était en mesure de résoudre les problématiques posées et la conclusion de l'article de Nittel (2015), écrit en 2015, résume bien cet état.

3.1.1 Apache Storm

Apache Storm est un système de traitement distribué en temps réel permettant de traiter des flux de données. Storm utilise le concept de topologie au travers duquel voyagent des tuples de données. Cette architecture est formée de *Spout* et de *Bolt*. Un *Spout* est une source de flux de données, tandis qu'un *Bolt* contient la logique de calcul. Un réseau de Spout et Bolt est représenté par un graphique acyclique dirigé appelé « topologies » (Landset *et al.*, 2015; *Apache Storm*, 2017) comme le montre la figure ci-dessous.

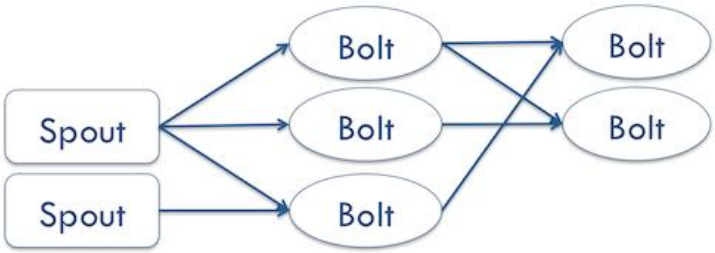


Figure 23 Topologie dans Apache Storm

Finalement, Apache Storm étant un système de traitement purement en flux, le concept de traitement par fenêtre n'existe pas. Cependant, Storm Trident, une extension de Storm, fonctionne sous un modèle de microlot, c'est-à-dire qu'il traite les informations en petits paquets, mais sans nécessairement inclure une sémantique élaborée pour ce type de traitement. Le traitement par fenêtre reste donc incertain pour l'instant.

3.1.2 Apache Spark

À la base, Apache Spark est un engin de traitement de données persistantes en parallèle. Une des caractéristiques qui fait de Spark un outil de traitement puissant est le concept de jeux de données distribués et résilients (resilient distributed dataset – RDD). Les RDD stockent les données en mémoire vive diminuant ainsi les opérations de lectures et d'écritures (Landset *et al.*, 2015; *Apache Spark*, 2017). Bien que Spark ne soit pas un système de traitement en temps réel, il supporte les flux (DStream) avec des traitements en microlots (micro batch). Spark supporte un certain nombre de traitements en fenêtre. Cependant, ces traitements de flux sous la forme de microlot impliquent un temps de latence équivalant à la durée du microlot.

3.1.3 Apache Flink

Apache Flink (*Apache Flink*, 2017) est une plateforme conçue pour le traitement distribué de flux et de données persistantes. À la base, Flink est conçu comme un engin de traitement de flux qui considère les données persistantes comme étant un flux d'éléments finis dans le temps. Le traitement de flux ne se fait donc pas sous forme de microlot. Flink peut donc atteindre des temps de latence très faibles. Des trois plateformes analysées, Flink est celle qui offre la plus grande variété d'options pour le traitement en fenêtre. Flink permet de grouper des éléments sur la base d'un attribut ou d'une fonction sur un attribut.

3.1.4 Comparaison des plateformes

Le tableau ci-dessous montre les différentes caractéristiques et comment elles sont supportées par les plateformes (Wei *et al.*, 2016; *Apache Flink*, 2017; *Apache Spark*, 2017; *Apache Storm*, 2017). Le symbole « ✓ », signifie que cette caractéristique est existante, le symbole « ~ » signifie qu'il serait possible d'obtenir cette caractéristique avec le système existant, le symbole « **? »** signifie qu'il est incertain que le système supporte

cette fonctionnalité et le symbole «✘» signifie que cette caractéristique est à développer. Le traitement par fenêtre de *Storm* étant incertain, cette plateforme n'a pas été retenue. Les choix restants sont Spark et Flink. Spark est une technologie qui est plus mature que Flink, mais du point de vue des traitements de données persistantes. Le traitement de flux dans Spark est plus récent et fonctionne sur la base de microlot. Ceci signifie que les flux ne sont pas traités avec une aussi faible latence que Flink ou Storm. D'un autre côté, Flink, bien que plus récent, offrait déjà la majorité des caractéristiques nécessaires. De plus, les fonctions de traitement de données persistantes ou de flux sont les mêmes, ce qui rend l'adaptation de l'un vers l'autre plus facile. La plateforme *Apache Flink* a finalement été retenue pour le prototype. Cependant, comme nous l'avons mentionné au début du chapitre, ces caractéristiques reflètent l'état de ces technologies lors de l'évaluation. Tous ces systèmes ont rapidement évolué et le choix final aurait peut-être été différent. Il est de notre avis qu'à l'heure actuelle, ces trois plateformes ont suffisamment évolué pour que les techniques proposées dans le cadre de ce travail puissent y être adaptées avec plus ou moins de difficulté.

Tableau 6 Caractéristiques supportées par les plateformes

ID	Caractéristique	Storm	Spark	Flink
A	Distribution et partitionnement aléatoire et balancé des informations sur les différents nœuds du système	✓	✓	✓
B	Partitionnement sur la base d'attribut spatial	✘	✘	✘
C	Partitionnement sur la base d'attribut non spatial	✓	✓	✓
D	Accumulation de données à l'aide de fenêtre	**?**	✓	✓
E	Permettre la jointure de deux flux massifs de données	✓	✓	✓
F	Permettre la jointure d'un flux massif de données et d'un flux de données traditionnelles	✓	✓	✓
G	Permettre la jointure d'un flux massif de données et de données massives persistantes	✘	✘	✘
H	Permettre la jointure d'un flux massif de données et de données persistantes traditionnelles	~	~	~
J	Permettre de diffuser des données persistantes traditionnelles à tous les nœuds	~	~	~
L	Permettre de diffuser des flux de données traditionnelles à tous les nœuds	~	~	~
M	Permettre le traitement sur le champ de flux de mégadonnées avec un faible temps de latence	✓	✘	✓
O	Permettre l'arrivée des données dans un désordre potentiel	✓	✓	✓
Caractéristique	**~ : Possible ✓ : Existante ? : Inconnue ✘: Non supportée nativement**			

3.2 Description d'une architecture Apache Flink

Avant de poursuivre la description de l'architecture du prototype, une description de la plateforme retenue est de mise. La figure suivante illustre une des multiples architectures possibles de traitement de flux avec Flink comme plateforme de traitement de flux.

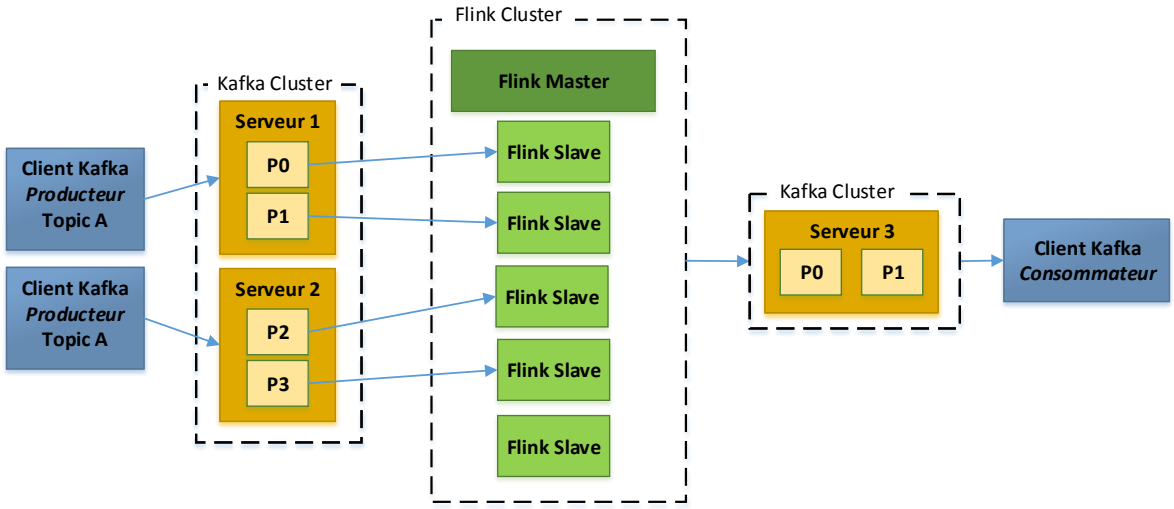


Figure 24 Architecture Flink incluant Kafka

3.2.1 Utilisation de Kafka comme source et destination

Dans cette configuration, des sources transmettent leurs données sous la forme de flux à une grappe d'ordinateur Kafka. Apache Kafka (*Apache Kafka*, 2017) est un agent de message temps réel ayant un temps de latence faible. Initialement créé par LinkedIn, Kafka est devenu open source en 2011. Cette plateforme est utilisée par plusieurs grandes entreprises telles que LinkedIn, Netflix, Spotify et Uber. Les données sont transmises d'un producteur vers un serveur Kafka. Celui-ci peut être configuré en plusieurs partitions (sans lien avec les partitions discutées dans ce mémoire) permettant une certaine redondance des informations. Dans cet exemple, Kafka sera la source de données du programme Flink. Chaque partition sera lue par un nœud et les données chemineront par la suite à l'intérieur de Flink selon le programme. Les résultats, sous forme de flux, seront émis vers un autre serveur Kafka et celui-ci pourra servir les données à des consommateurs. Cet exemple est vulgarisé afin de comprendre le fonctionnement de Flink. Il existe plusieurs sources et destinations (sink) possibles. Dans le cadre de ce projet, le prototype utilisera comme source et destination des fichiers accédés en lecture et écriture sous forme de flux.

3.2.2 Plateforme Apache Flink

Flink possède une riche librairie de fonctions de transformation de données. Le tableau suivant illustre seulement un échantillon des transformations possibles. Une liste complète est disponible sur le site de Flink (*Apache Flink*, 2017).

Tableau 7 Échantillon des opérateurs de transformation Flink

Transformation	Description

Map
DataStream→DataStream
	Prend un élément en entrée et produit un élément en sortie
	
	`DataStream<Integer> dataStream = //...`
	`dataStream.map(new MapFunction<Integer, Integer>() {`
	`  @Override`
	`  public Integer map(Integer value) throws Exception {`
	`  return 2 * value;`
	`  }`
	`});`

FlatMap
DataStream→DataStream
	Prend un élément en entrée et produit plusieurs éléments en sortie
	
	`dataStream.flatMap(new FlatMapFunction<String, String>() {`
	`  @Override`
	`  public void flatMap(String value, Collector<String> out)`
	`  throws Exception {`
	`  for(String word: value.split(" ")){`
	`  out.collect(word);`
	`  }`
	`  }`
	`});`

Union DataStream→ DataStream	Unit deux flux de données `dataStream.union(otherStream1, otherStream2, ...);`

Window CoGroup
DataStream ,DataStream
→DataStream
	Groupe deux flux sur la base d'une clé
	
	`dataStream.coGroup(otherStream)`
	`  .where(0).equalTo(1)`
	
	`.window(TumblingEventTimeWindows.of(Time.seconds(3)))`
	`  .apply (new CoGroupFunction () {...});`

Création de fenêtres à bascule
DataStream→
windowedStream
	Crée une fenêtre à bascule où les éléments sont groupés selon une clé pendant une période de temps.
	
	`DataStream<T> input = ...;`
	
	`input`
	`  .keyBy(<key selector>)`
	`  .window(TumblingProcessingTimeWindows.of(Time.seconds(5)))`
	`  .<windowed transformation>(<window function>);`

Window Apply
windowedStream→
DataStream
	Applique une fonction sur les éléments d'une fenêtre
	
	`windowedStream.apply (new WindowFunction<Tuple2<String,Integer>,`
	`Integer, Tuple, Window>() {`
	`  public void apply (Tuple tuple,`
	`  Window window,`
	`  Iterable<Tuple2<String, Integer>> values,`
	`  Collector<Integer> out) throws Exception {`
	`  int sum = 0;`
	`  for (value t: values) {`
	`  sum += t.f1;`
	`  }`
	`  out.collect (new Integer(sum));`
	`  }`
	`});`

L'exemple suivant illustre un traitement de flux dans Flink. Des véhicules produisent un flux de données comprenant, à chaque intervalle, leur position, l'identifiant unique du véhicule, leur vitesse et leur direction. Dès

l'arrivée dans Flink, le flux est transformé sous la forme d'un objet (A). Par la suite, une ligne, représentant le trajet futur pour la prochaine seconde, est créée pour chaque point à l'aide de la vitesse et de la direction (B). Ensuite, un code de partitionnement est créé selon l'emplacement spatial. Selon l'étendue de la ligne, un ou plusieurs codes de partitionnement sont créés (C). Les données sont ensuite accumulées pendant 100 millisecondes. Les données sont regroupées par leur clé de partitionnement (D et E). À la fin de la période d'accumulation, la fenêtre exécute la fonction d'intersection pour produire un flux représentant les positions à risque de collision. La Figure 25 représente l'emplacement spatial des trajets de l'exemple, tandis que la Figure 26 et le Code et pseudocode 1 illustrent la séquence de traitement. Il faut noter que, par défaut, Flink décide de la distribution des partitions sur les nœuds.

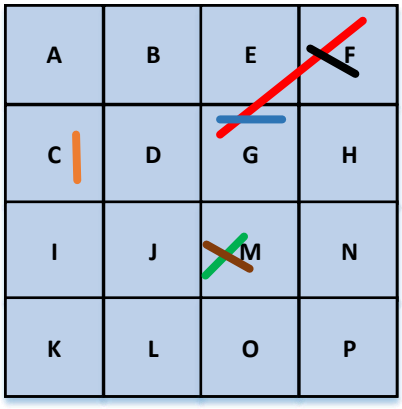


Figure 25 Emplacement spatial des trajets

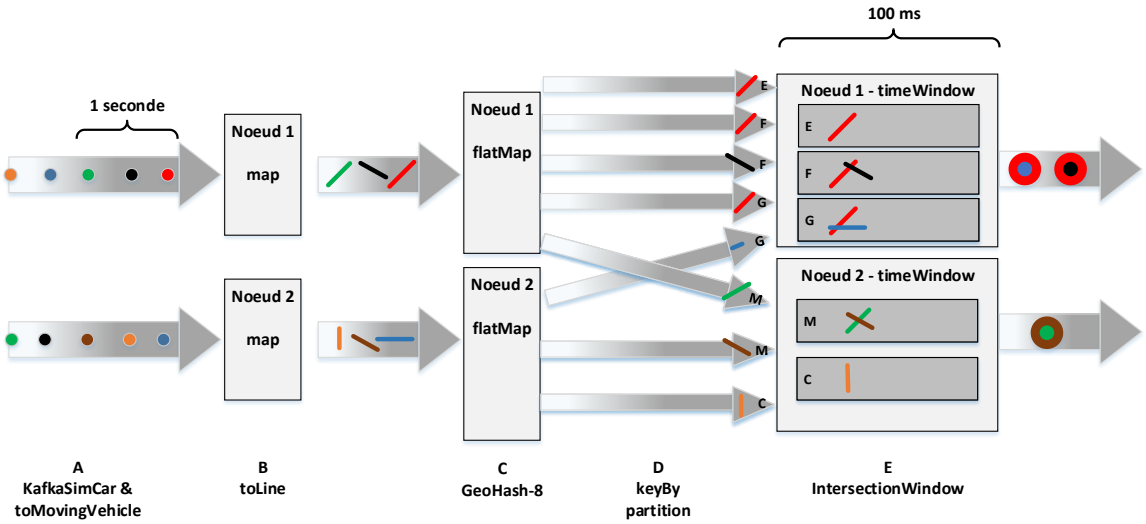


Figure 26 Traitement spatial dans Flink

```
DataStream<String> kafkaGeoStreamInputMV = env.addSource(new FlinkKafkaConsumer010<>(topic, new
SimpleStringSchema(), properties))
         .name("KafkaSimCar")
         .setParallelism(2)

DataStream<MovingVehicleSE> SimCarGeoStream  = kafkaGeoStreamInputMV
         .flatMap(new flatMaptoMovingVehicle())
         .name("toMovingVehicle")

DataStream<MovingVehicleSE> SimCarGeoStreamFuturePath  =SimCarGeoStream
         .map(new maptoLine(nbMilliSecondePourAnalyse))
         .name("toLine")

DataStream<MovingVehicleSE>SimCarGeoStreamFuturePathHash = SimCarGeoStreamFuturePath
         .flatMap(new flatMapGeoHash8(hashLevelAnalyse))
         .name("GeoHash-8")
         .setParallelism(parallelism)

DataStream<ColisionDensity> SimCarWarning = SimCarGeoStreamFuturePathHash
         .keyBy(new KeySelectorPartitionID())
         .timeWindow(Time.of(intersectionWindowMillisecond, TimeUnit.MILLISECONDS))
         .apply(new IntersectionWindowQuadTree(hashLevelReport))
         .name("IntersectionWindow")
         .setParallelism(parallelism)
```

Code et pseudocode 1 Code Java - Traitement spatial dans Flink

Flink utilise une méthode de contre-pression pour la gestion du flux. Lorsqu'un opérateur produit des données plus rapidement que ne peut le consommer l'opérateur en aval, il se produit une accumulation à l'entrée de cet opérateur. Flink étiquettera l'opérateur en amont comme subissant un effet de contre-pression. Si la contre-pression perdure dans le temps, alors elle se propage vers les autres opérateurs en amont et elle peut affecter le débit d'entrée des données dans le système. Si l'opérateur réussit à reprendre le dessus, alors la contre-pression disparait (Celebi, 2015; *Apache Flink*, 2017).

3.3 Architecture du prototype

Afin de tester les différents cas d'étude, une plateforme a été mise en place. Cette plateforme comprendra comme élément central le système de traitement de flux *Apache Flink*. Les données que nous voulons analyser doivent prendre la forme de flux. Nous avons retenu une source permettant de lire un fichier sous la forme d'un flux. Le choix est justifié dans la section suivante. Un fichier de simulation servira donc de point d'entrée. À la sortie, Flink transmettra les données dans un fichier de sortie sous la forme d'un flux. Finalement, un autre programme contrôlera la séquence de tests et la récupération des données de performance. Les sections suivantes décrivent brièvement les différentes parties de l'architecture du prototype.

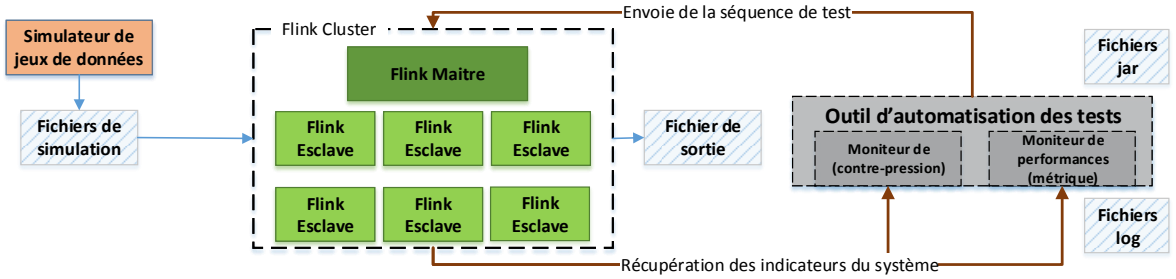


Figure 27 Architecture de la plateforme de tests

3.3.1 Simulateur de jeux de données

Pour obtenir des données permettant de réaliser les traitements décrits dans les divers cas d'études, un simulateur de données a été créé. Le simulateur permet de créer des points à un intervalle d'une seconde sur un trajet aléatoire suivant les réseaux routiers du Québec et de l'Ontario. Ces trajets ont été créés en prenant deux points aléatoires, mais ayant une distance minimale de 50 km, et en déterminant le chemin le court entre ceux-ci. Par la suite, des points, à un intervalle d'une seconde, ont été extraits le long du trajet en prenant une vitesse approximative de 50 km/h. Chacun des points du trajet a été aléatoirement décalé entre 1 à 2 mètres pour simuler la précision d'un GPS. Une portion de 15 minutes sur ce trajet a été extraite comme source d'information pour le simulateur. Environ 120 000 trajets de ce type ont ainsi été générés pour un total d'environ 100 millions de points.

Deux méthodes permettant d'introduire les points dans le système ont été évaluées. La première méthode consistait à utiliser la plateforme Kafka en amont comme source de données. Les points auraient alors été transmis par un simulateur en temps réel vers Kafka et Flink aurait puisé dans Kafka afin d'obtenir les données à traiter. L'autre méthode consistait à placer une copie du fichier de points sur les nœuds esclaves et à utiliser ce fichier comme source de flux. Les deux méthodes ont été testées, mais la deuxième est celle qui a finalement été retenue, car elle correspondait plus à un test de charge, car le débit était plus intense. Elle permettait aussi d'éliminer certains facteurs externes, comme la bande passante utilisée par la source des données ainsi qu'une plateforme externe à celle étant testée. Cependant, avec la méthode utilisée, le débit est supérieur à une position par véhicule par seconde. Comme nous le verrons plus tard, des débits supérieurs à 500 000 positions par secondes ont été obtenus. La figure suivante montre l'étendue du jeu de données utilisé par le simulateur.

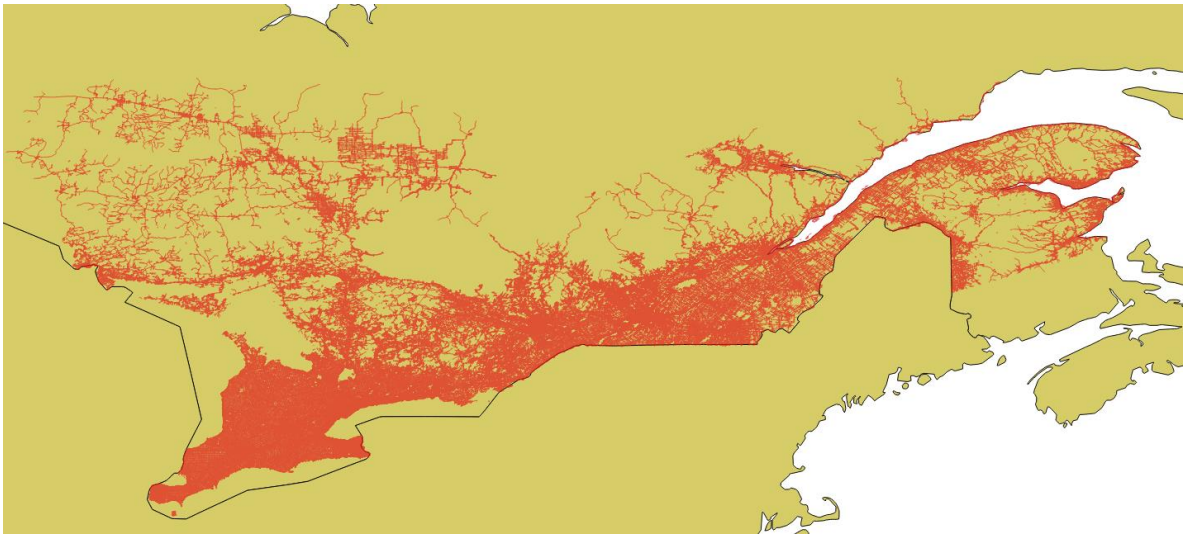


Figure 28 Étendue spatiale du jeu de données pour les tests

Le format du jeu de données est sous forme textuelle. Chaque enregistrement comprend les informations suivantes :

- L'identifiant du véhicule;
- Le numéro de séquence de la transmission;
- La latitude et la longitude en WGS84;
- La vitesse en mètre par seconde;
- La direction.

La figure suivante représente un extrait du jeu de données.

```
1;1;46.339257;-72.562498;17.3;24.2
2;1;42.947153;-78.933743;18.4;304.3
3;1;45.594745;-73.471840;25.4;205.1
4;1;42.422617;-82.201164;16.3;126.8
5;1;43.004536;-81.303064;11.6;141.4
6;1;43.845818;-79.443759;12.4;339.1
7;1;45.806587;-77.079164;11.8;309.6
8;1;46.901534;-71.277270;2.4;71.9
9;1;47.278439;-67.967070;12.8;198.8
10;1;44.150530;-80.478787;14.4;56.3
11;1;45.458828;-75.524858;24.3;52.6
12;1;43.466645;-80.524506;15.8;136.0
13;1;43.978569;-77.572681;10.0;87.2
14;1;46.661356;-72.493529;15.0;304.1
15;1;45.547195;-74.148025;11.8;308.8
```

Figure 29 Extrait du jeu de données utilisé pour le prototype

3.3.2 Cluster Flink

Le cluster Flink est composé d'un nœud maitre (master) et de 6 nœuds esclaves (slaves). Chacun des nœuds est installé sous une plateforme Linux CentOS 7 dans un environnement virtualisé à l'aide du logiciel Oracle VirtualBox. Windows 10 est le système d'exploitation des ordinateurs hôtes. Ceux-ci ont tous été installés à partir de la même image. Les ordinateurs sont de marque Dell Optiplex 790 avec un processeur i7-2600 à 3,40 GHz. Chacune des machines Linux virtuelles possède 8 Gb de mémoire vive et 4 processeurs. Les cartes réseau sont toutes de 1 Gb/s ainsi que le commutateur réseau. La version 1.2 de Flink a été utilisée pour ces tests.

3.3.3 Moniteur de performance

Afin de surveiller et d'archiver le déroulement des tests, nous avons créé deux programmes en Python. Le premier permet de récupérer les informations de contre-pression des opérateurs à l'aide de requêtes envoyées à l'interface REST du nœud maitre. Le deuxième programme permet de récupérer les métriques produites par le cluster par une adaptation Python de l'interface STATSD. Ces informations seront utilisées pour l'analyse des résultats. Finalement, pour normaliser et pour soustraire l'erreur humaine durant les tests, nous avons également conçu un programme servant à automatiser les tests en Python. Ce programme télécharge le code Flink (fichier jar) au serveur Flink. Il lance ensuite la commande de démarrage du programme et démarre les deux programmes du test précédents. Après quatre minutes, il lance une séquence d'arrêt, archive les résultats et démarre le prochain test.

3.4 Prototypes de traitement spatial

Deux prototypes d'adaptation de traitement spatial dans un contexte de flux de mégadonnées ont été testés. Le premier consiste à évaluer des opérations de relation spatiale en vérifiant des intersections. Le deuxième, quant à lui, a évalué des relations de proximité en évaluant les k plus proches voisins. Rappelons que l'objectif est de tester ces opérations lorsque les deux opérandes sont issus de flux massifs.

3.4.1 Cadre pour le développement des prototypes

Les prototypes ont pour but d'évaluer les méthodes d'adaptation des traitements spatiaux dans un contexte de flux massif. Les méthodes proposées sont le géo-hachage pour le partitionnement et le calcul d'opérateurs spatiaux à l'intérieur de fenêtre temporelle. Les opérations en amont servant à ingérer et préparer les données et celles en aval servant à synthétiser les résultats ne font pas partie des méthodes proposées et n'ont pas été analysées. Pour ces raisons, ces opérations se verront assigner des niveaux de parallélisme maximal pour éviter les goulots d'étranglement externes aux opérations à l'étude.

De plus, lors du traitement par fenêtre, des index spatiaux locaux sont utilisés pour améliorer les performances. Les index utilisés sont les Quad-Tree pour les relations spatiales et le KD-Tree pour l'analyse de proximité. Le choix du KD-Tree semblait plus performant, selon nos tests, que le Quad-Tree pour trouver le plus proche voisin. Nous avons adapté la librairie du KD-Tree afin que celle-ci puisse nous retourner les « k » plus proches voisins et non seulement le plus proche voisin. Diverses techniques d'indexation ont été testées durant ces travaux, mais les détails ne seront pas publiés ni discutés dans le cadre de ce mémoire. Nous considérons que les techniques d'indexation locale ne relèvent pas des traitements en parallèle, car leur exécution est locale et leur performance relative serait la même dans un contexte non parallèle.

Finalement, une analyse spatiale sur des flux de données peut être représentée par un graphe acyclique dirigé où les nœuds représentent les traitements et où les segments sont des flux de données. Dans ce mémoire, certains des traitements proposés se font à l'aide de fenêtre. Selon l'analyse requise, il sera possible d'avoir deux branches de traitement en parallèle qui se rejoignent dans un même nœud. Si l'une de ces branches comporte plusieurs fenêtres de traitement successives, alors un problème de synchronisation pourrait survenir. Les deux flux de données entrants dans le nœud pourraient alors être liés à des plages temporelles différentes. Cet effet a été constaté lors des tests initiaux. Cependant, dans le cadre de ce mémoire, ce sont les opérateurs spatiaux qui seront étudiés et non les effets de la topologie de traitement.

3.4.2 Relation spatiale

Afin de tester les opérateurs de relation spatiale, le test effectué consiste à trouver les intersections de véhicules en déplacement à partir du jeu de données mentionné précédemment. Pour chaque position reçue, une ligne représentant le déplacement futur dans la prochaine seconde est calculée à partir de la vitesse et de la direction. À chaque intervalle d'une seconde, les intersections entre les lignes sont calculées. Par la suite, les positions d'intersections sont accumulées pendant une minute et une densité est calculée par partition et le résultat est transmis sous forme de flux vers un système en sortie. La figure suivante représente la séquence de traitement.

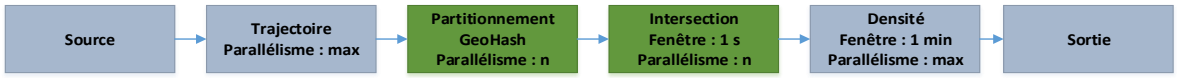


Figure 30 Séquence de traitement pour l'opérateur de relation spatiale

Les opérations en vert représentent les adaptations nécessaires pour traiter l'opérateur d'intersection ayant des opérandes issus des flux massifs de données. Ce sont ces opérations qui ont été analysées. Le niveau de parallélisme représente le nombre de cœurs disponibles pour le traitement. Pour les opérations visées (en vert), il y a eu une variation du nombre de cœurs alloués afin de voir la variation du débit. Pour les autres opérations,

le nombre de cœurs maximum du cluster a été assigné afin de ne pas causer des goulots d'étranglement. Le code ci-dessous représente la séquence de traitement des deux opérations en vert.

```
DataStream<MovingVehicleSE>SimCarGeoStreamFuturePathHash = SimCarGeoStreamFuturePath
                            .flatMap(new flatMapGeoHash8(hashLevelAnalyse))
                            .name("flatMapGeoHash8")
                            .setParallelism(parallelism)
                            .rescale();

DataStream<ColisionDensity> SimCarWarning = SimCarGeoStreamFuturePathHash
                            .keyBy(new KeySelectorPartitionID())
                            .timeWindow(Time.of(accidentWindowMillisecond, TimeUnit.MILLISECONDS))
                            .apply(new AccidentWindowQuadTree(hashLevelReport))
                            .name("AccidentWindow")
                            .setParallelism(parallelism)
                            .rescale();
```

Code et pseudocode 2 Code pour la séquence de traitement de l'opérateur de relation spatiale

Dans l'opération de partitionnement, un niveau de partitionnement était fourni en paramètre. La figure 31 illustre la procédure. Dans un premier temps, tous les codes de partition (jaune et bleu) couverts par le rectangle englobant (tiret rouge) de la ligne (orange) sont calculés. Par la suite, seules les partitions réellement traversées par la ligne sont retenues. Pour chaque code de partition (jaune), une copie de la ligne sera transmise en sortie de cette opération.

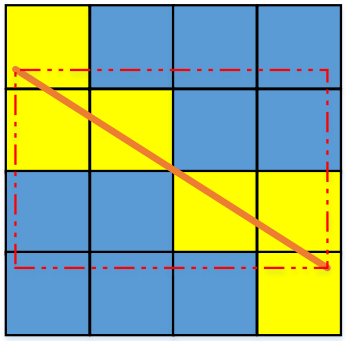


Figure 31 Géo-hachage d'une ligne

Toutes ces nouvelles données sont transmises à l'opération d'intersection qui fonctionne sous la forme d'une fenêtre. Les informations sont regroupées selon leurs codes de partition. C'est la fonction « keyBy » qui détermine l'attribut servant de base de regroupement. Quand le temps d'accumulation est écoulé, la fonction définie dans « apply » est activée. La fonction, représentée par le pseudo-code 3, consiste à créer un index local pour effectuer la recherche des intersections. Lors des tests effectués (non présentés dans le cadre de ce mémoire), il est apparu que cette méthode était plus efficace qu'un produit croisé.

```
Si nombre d'éléments dans la fenêtre est > 1
        Pour chaque élément
                Insérer l'élément dans un QuadTree

        Pour chaque élément (A)
                Obtenir l'enveloppe de A
                Enlever l'élément A du QuadTree
                Obtenir la liste des éléments recoupant l'enveloppe de A

                Pour chaque élément de cette liste (B)
                        Si A.id != B.id et A.intersecte(B)
                                Calculer le code GeoHash de l'intersection
                                Collecter l'information
```

Code et pseudocode 3 Pseudo-code de la fenêtre de traitement pour l'opérateur de relation spatiale

À la sortie de cette opération, un code de partition représentant le point d'intersection est transmis à l'opération en aval, soit le calcul de la densité.

3.4.3 Analyse de proximité

Pour tester l'opérateur de proximité, le test effectué a pour but de trouver les 3 voisins les plus près dans un rayon de 100 mètres autour de chacun des points. Les positions sont accumulées pendant une fenêtre de 30 secondes. À la fin de cette fenêtre de 30 secondes, l'opérateur est effectué afin de trouver les 3 voisins les plus près. Comme nous le présenterons dans la section des résultats, les performances n'ont pas été concluantes. Afin de tenter d'améliorer les résultats, deux méthodes ont été testées. Les deux méthodes sont présentées ci-dessous.

3.4.3.1 Méthode du partitionnement symétrique

Dans ce premier test, les points sont assignés à un code de partition basé sur une profondeur de partitionnement unique. La séquence de traitement est présentée ci-dessous.

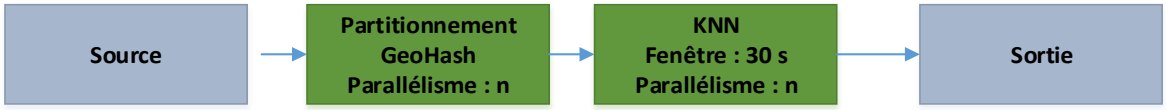


Figure 32 Séquence de traitement pour l'opérateur knn - partition symétrique

Les points subissent des translations comme celles expliquées précédemment. Pour chaque point en entrée, d'un à trois points par partition peuvent être transmis à la sortie de l'opération de partitionnement. Par la suite, les points sont groupés par partition et l'opération KNN est effectuée. Le code suivant représente la séquence de ces deux opérations.

```
DataStream<MovingVehicleSE> GeoHashedTranslationStream = SimCarGeoStream
                  .flatMap(new flatMapTranslationHash(hashCodeLevel,searchRadius))
                  .name("DoGeoHashing")
                  .setParallelism(parallelism);

DataStream<PerformanceTimeRecordPerHash> knnStreamPerformanceMV = GeoHashedTranslationStream
                  .keyBy(new KeySelectorPartitionID())
                  .timeWindow(Time.of(30, TimeUnit.SECONDS))
                  .apply(new KNNKDTreePerformanceWindow(nbNearestNeighbor))
                  .name("KNNKDTreePerformanceWindow")
                  .setParallelism(parallelism);
```

Code et pseudocode 4 Code pour la séquence de traitement pour l'opérateur knn - partition symétrique

Les items produits pour l'opération de partitionnement sont groupés par le code de partition. Ceux-ci sont accumulés pendant 30 secondes. Après ce délai, la fonction définie par « apply » est exécutée. Le pseudo-code suivant représente le traitement de l'opération KNN. Les éléments sont insérés dans un index local de type KD-Tree. Par la suite, une recherche des "k" plus proches voisins est effectuée. Pour analyser la performance, on retourne en sortie la durée du calcul et le nombre de points traités.

```
Début = heure de début
NbÉlement = Nombre d'éléments dans la fenêtre
Si le nombre d'éléments est > 1 dans la fenêtre
         Pour chaque élément (A)
                  Insérer A dans le KD-Tree

         Pour chaque élément (A)
                  Trouve les k plus proche voisin de A
                  Pour chaque k voisin
                           Calculer la distance avec A
Fin = heure de fin
Durée = Fin-Début
Collecter informations Durée, NbÉlement
```

Code et pseudocode 5 Pseudo-code de la fenêtre de traitement knn - partition symétrique

3.4.3.2 Méthode de partitionnement asymétrique

Dans ce deuxième test, les points sont premièrement assignés à un code de partition basé sur une profondeur de partitionnement unique. Par la suite, selon la densité, les points sont réassignés à des codes de partitionnement plus profond dans le but de diminuer le nombre de points par partition et d'augmenter la rapidité de traitement selon le principe de *Divide ut regnes* (diviser pour régner). Finalement, la même opération KNN est appliquée. La séquence de traitement est présentée ci-dessous.

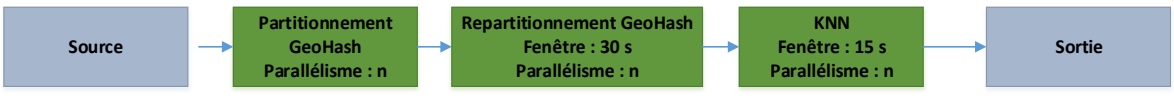


Figure 33 Séquence de traitement pour l'opérateur knn - partition asymétrique

L'enchainement de code de cette méthode est présenté ci-dessous.

```
DataStream<MovingVehicleSE> GeoHashedTranslationStream = SimCarGeoStream
                    .flatMap(new flatMapTranslationHash(hashCodeLevel,searchRadius))
                    .name("DoGeoHashing")
                    .setParallelism(parallelism)
                    .rebalance();

DataStream<MovingVehicleSE> reGeoHashedTranslationStream = GeoHashedTranslationStream
                    .keyBy(new KeySelectorPartitionID())
                    .timeWindow(Time.of(30, TimeUnit.SECONDS))
                    .apply(new reHashWindow4(maxDensity,hashCodeLevel, reHashCodeLevel,searchRadius))
                    .name("ReHashWindow")
                    .setParallelism(parallelism)
                    .rebalance();

DataStream<PerformanceTimeRecordPerHash> knnStreamPerformanceMV = reGeoHashedTranslationStream
                    .keyBy(new KeySelectorPartitionID())
                    .timeWindow(Time.of(15, TimeUnit.SECONDS))
                    .apply(new KNNKDTreePerformanceWindow(nbNearestNeighbor))
                    .name("KNNKDTreePerformanceWindow")
                    .setParallelism(parallelism)
                    .rescale();
```

Code et pseudocode 6 Code de la séquence de traitement pour l'opérateur knn - partition asymétrique

Le principe du repartitionnement consiste à vérifier si une partition contient plus d'un nombre maximal de points. Dans l'affirmative, chacun des points participant à la partition originale est repartitionné avec un code de partitionnement plus profond. Les zones plus denses se verront assigner des partitions plus petites que celles moins denses. La figure suivante présente le principe. Le point vert et le point orange se verront assigner deux partitions, soit A2 et A6 pour le vert et A10 et A6 pour l'orange. Le point rouge restera assigné à A12 et puisque ces translations sont à l'extérieur de la partition parent (A), ceux-ci seront gérés par la partition à la droite de la partition "A". Finalement, le point noir translaté nord-est sera assigné à la partition A14. Dans cet exemple, le nombre d'enregistrements augmentera de 50 %, mais la densité maximale passera de 4 à 2 et le nombre de partitions augmentera de 1 à 5 (seules celles ayant des points seront existantes).

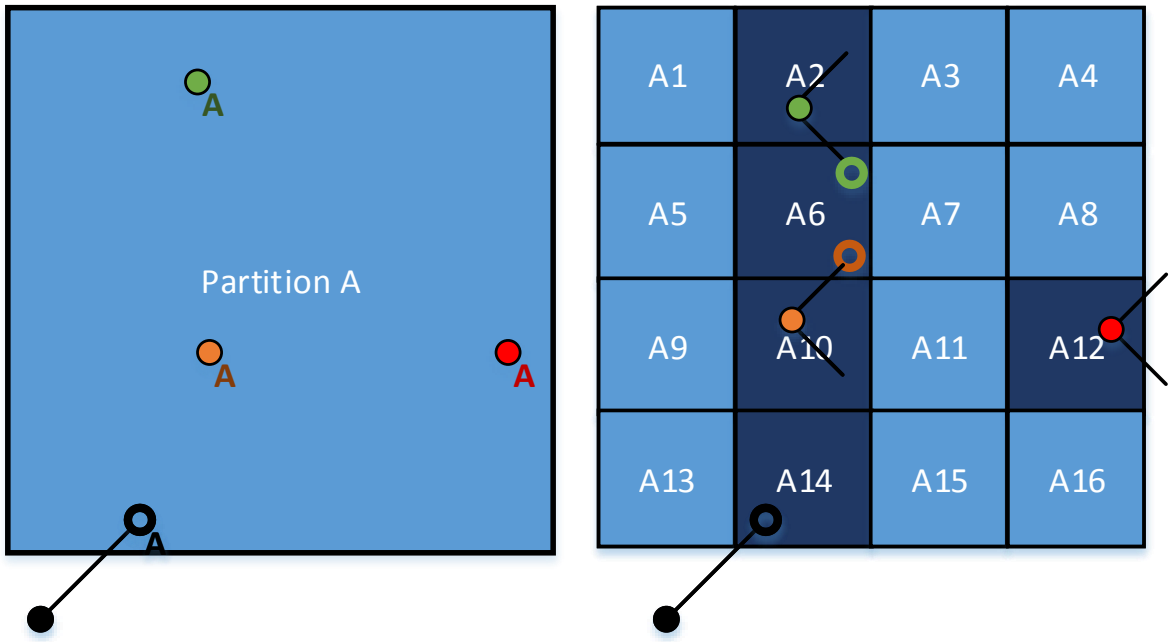


Figure 34 Partitionnement asymétrique

L'algorithme est illustré par le pseudo code suivant.

```
Si le nombre d'item < nombre maximal permis
        Pas de repartitionnement
        L'information à l'entrée est transférée au collecteur
Autrement
        Si le nombre d'item/10 < nombre maximal permis
                Profondeur de partitionnement = profondeur de partitionnement initial +2
        Autrement
                Profondeur de partitionnement = profondeur de partitionnement initial +4

        Pour chaque Élements (A)
                Si A est indigène à la partition
                        Calcule les nouveaux codes pour position originale (CodeOrig)
                        Calcule les nouveaux codes pour position translatée NE et SE (CodeNE, CodeSE)

                        Collecte A avec CodeOrig

                        Si CodeNE != CodeOrig et CodeNE est un enfant de CodeInitial
                                Collecte A avec CodeNE

                        Si CodeSE != CodeNE et CodeSE != CodeOrig et CodeSE est un enfant de CodeInitial
                                Collecte A avec CodeNE

                Autrement A est exotique à la partition
                        Si le code est issu d'une translation NE
                                Calcule le CodeNE pour la position translatée NE
                                Collecte A et CodeNE
                        Autrement Si le code est issu d'une translation EE
                                Calcule le CodeSE pour la position translatée SE
                                Collecte A et CodeSE
```

Code et pseudocode 7 Pseudo-code de la fenêtre de repartitionnement asymétrique

3.5 Banc de tests

Une application permettant l'automatisation des tests et la récupération des métriques a été mise en place pour faciliter les procédures de test ainsi que pour éliminer les erreurs et biais humains dans la soumission des tests. De plus, il faut garder à l'esprit que les mesures de performance dans les systèmes de traitement en flux sont différentes que celles mesurées pour les systèmes de traitement de données statiques. Puisqu'un flux de données est potentiellement infini, on parlera donc plus de nombre de données traitées par seconde, que de temps de traitement.

3.5.1 Cadre de tests

Pour ces tests, les données étaient transmises au système sous la forme d'un flux à partir d'un fichier disposé sur les nœuds esclaves. La lecture était faite par un seul cœur et les tests ont montré des débits d'environ 1,9 million d'enregistrements par seconde pour cette opération seule. Ce résultat est supérieur à tout autre débit mesuré lors des tests. Cette méthode d'ingestion des données n'a donc pas été un goulot d'étranglement.

Lors de la création des tests, l'utilisateur est en mesure de choisir le niveau de parallélisme désiré (nombre de cœurs). Cependant, c'est Flink qui détermine la distribution des opérateurs sur les cœurs. Bien que tous les

nœuds soient identiques, il se peut que des processus du nœud, externes au programme Flink, induisent des variations dans les performances. Pour minimiser cet effet, les tests ont été exécutés plusieurs fois.

Le débit mesuré peut varier en fonction du temps pour plusieurs raisons. Une de ces raisons pourrait être la variation spatiale des éléments analysés. Il est fort possible qu'une distribution spatiale égale des données soit plus facile à traiter que s'il y a de fortes zones de concentration. Cet aspect, bien qu'intéressant, n'a pas fait l'objet de cette étude. Le débit peut aussi varier en fonction du mécanisme interne de contre-pression expliqué précédemment.

Pour ces tests, la méthode choisie consiste à tester le système en envoyant le débit maximum. De cette façon si un goulot d'étranglement existe, le phénomène de contre-pression apparaitra et le débit à l'entrée sera ralenti par le système. C'est ce débit d'entrée qui est la valeur de performance mesurée. Pour chaque test, une valeur de profondeur de partitionnement et un niveau de parallélisme (nombre de cœurs) ont été assignés. Pour chacun des tests, trois essais ont été faits. La valeur finale de débit retenue pour un test est la moyenne de toutes les valeurs de débit pendant les trois essais. L'écart-type sera calculé et il représentera la stabilité du débit durant la période de test.

Certaines autres variables ont été analysées pour permettre d'expliquer les résultats, comme, le nombre d'enregistrements par cœur à l'entrée d'un opérateur et l'endroit où la contre-pression apparait. Ces valeurs seront présentées dans la section d'analyse des résultats.

3.5.2 Plan de test pour le prototype de relation spatiale

Pour ce prototype, les performances en débit à l'entrée, en termes de nombre d'enregistrements par seconde, étaient la valeur de performance mesurée. Les variables explicatives seront le niveau de partitionnement et le nombre de cœurs disponibles. Pour chaque combinaison de partitionnement – nombre de cœurs, trois tests de 4 minutes ont été effectués. Le tableau suivant résume les différentes combinaisons de tests.

Tableau 8 Plan de test pour le prototype de relation spatiale

Niveau de partitionnement (P)	Niveaux 1, 3, 7, et 11
Nombre de cœurs (C)	1, 4, 10, 16, 24 cœurs
Nombre de tests par combinaison partition-cœur	3 tests
Nombre de tests totaux	60 tests

3.5.3 Plan de test pour l'analyse de proximité

Pour ce test, c'est aussi le débit en nombre d'enregistrements par seconde qui est mesuré pour évaluer la performance. Les variables explicatives sont les mêmes que pour le test précédent. La durée et le nombre de

tests par combinaison sont aussi les mêmes. Les tableaux suivants montrent les combinaisons de tests pour les deux versions, soit le partitionnement symétrique et asymétrique.

Tableau 9 Plan de test pour le prototype d'analyse de proximité - géo-hachage symétrique

Niveau de partitionnement (P)	Niveaux 7, 9 et 11
Nombre de cœurs (C)	4, 10, 16, 24 cœurs
Nombre de tests par combinaison partition-cœur	3 tests
Nombre de tests totaux	36 tests

Tableau 10 Plan de test pour le prototype d'analyse de proximité - géo-hachage asymétrique

Niveau de partitionnement (P)	Combinaison 7-9-11
Densité maximale pour repartitionnement	2000
Nombre de cœurs (C)	4, 10, 16, 24 cœurs
Nombre de tests par combinaison partition-cœur	3 tests
Nombre de tests totaux	12 tests

Il faut noter que le partitionnement de niveau 1 et le nombre de cœurs de 1 n'ont pas été pris en compte dans les résultats, car l'ordinateur n'a pas été en mesure de traiter les informations dans un temps opportun. Cette configuration correspond à un traitement sans aucune parallélisation. À de faibles niveaux de partitionnement, le temps requis pour traiter la quantité de données d'une fenêtre de temps était très supérieur à la dimension de celle-ci et le système ne parvenait pas à prendre le dessus.

Chapitre 4 Résultats et analyses

Cette section présente les résultats, les observations et les analyses des deux prototypes, soit l'opérateur de relation spatiale et celui de l'analyse de proximité. Finalement, une conclusion sur les résultats des tests de performance réalisés termine ce chapitre.

4.1 Résultats du prototype de relation spatiale

Les résultats obtenus sont présentés dans le tableau 11 et la figure 35. Le tableau présente la moyenne du nombre d'enregistrements par seconde à l'entrée du traitement ainsi que l'écart-type. Ces valeurs sont en fonction du niveau de géo-hachage et du nombre de cœurs.

Tableau 11 Résultats du prototype de relation spatiale

Niveau de géo-hachage	**1**		**3**		**7**		**11**	
Nb de cœurs	Moyenne	Écart-Type	Moyenne	Écart-Type	Moyenne	Écart-Type	Moyenne	Écart-Type
1	124674	3524	126376	2921	138694	2671	135767	2686
4	134580	2642	208741	5258	263943	7363	251142	10061
10	139513	3125	235799	4770	403295	9719	395281	6979
16	135314	2443	239720	6794	471181	6529	467439	5252
24	139733	2125	240893	3391	563658	8713	567330	8376

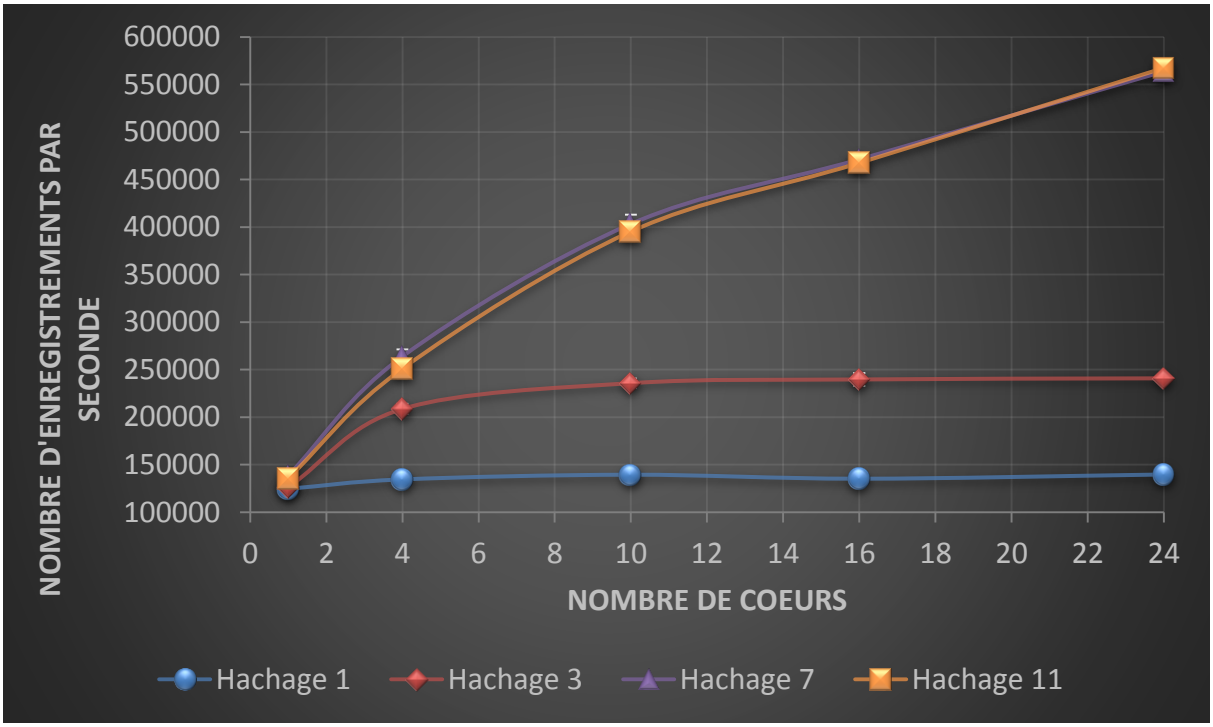


Figure 35 Évolution du débit selon le niveau de parallélisme et la profondeur de partitionnement

Les résultats permettent d'énoncer les observations suivantes :

- Les débits sont semblables pour toutes les profondeurs de géo-hachage lorsqu'il n'y a qu'un seul cœur.
- L'allure de la courbe de progression du débit en fonction du nombre de cœurs est différente selon la profondeur du géo-hachage.
- Le géo-hachage de niveau 11 ne présente aucun gain par rapport au géo-hachage de niveau 7.

4.2 Analyse des résultats du prototype de relation spatiale

Dans cette section, les trois observations précédentes sont analysées afin d'interpréter les résultats.

4.2.1 Première observation

Les débits sont semblables pour toutes les profondeurs de géo-hachage lors qu'il n'y a qu'un seul cœur. Ce cas représente le traitement sans parallélisme puisque toutes les opérations sont traitées sur un seul cœur. La faible différence observée dans les débits relativement au niveau de géo-hachage pourrait s'expliquer par l'effet « diviser pour régner ». La complexité algorithmique étant de O(n log n) pour la construction de l'index local (Heineman *et al.*, 2008), il est donc plus rapide de créer plusieurs index ayant moins de points qu'un seul index contenant la totalité des points.

4.2.2 Deuxième observation

L'allure de la courbe de progression du débit en fonction du nombre de cœurs est différente selon la profondeur du géo-hachage. Cette observation peut être divisée suivant les sous observation suivantes :

- Le géo-hachage de niveau 1 ne présente aucun gain de performance lors de l'augmentation du nombre de cœurs.
- Le géo-hachage de niveau 3 présente un gain lors du passage de 1 à 4 cœurs. Un gain plus faible est observé avec 10 cœurs et par la suite, aucun gain supplémentaire.
- Les géo-hachages de niveaux 7 et 11 présentent des gains systématiques avec l'augmentation du nombre de processeurs.

Pour comprendre ces observations, il faut regarder la répartition spatiale des données en fonction des partitions. Le jeu de données, en rouge dans l'illustration ci-dessous, est situé seulement dans la partition « e » quand le niveau de géo-hachage est de 1. Toutes les données seront donc traitées par le cœur étant assigné à cette partition et l'ajout de nouveaux cœurs n'aura aucun effet.

Cinq partitions sont présentes avec un niveau de géo-hachage de niveau 3, soit « ecd », « ece », « efb », « efc » et « eff ». Cinq cœurs au maximum pourront se diviser la tâche. Lors du passage de 1 à 5 cœurs, il y aura un partage constant des données. Par la suite, l'ajout de cœurs ne produira aucun effet sur le débit. L'allure de la courbe montre qu'il y a une grande augmentation lors du passage de 1 à 4 cœurs et une augmentation beaucoup plus faible entre 4 et 10 cœurs.

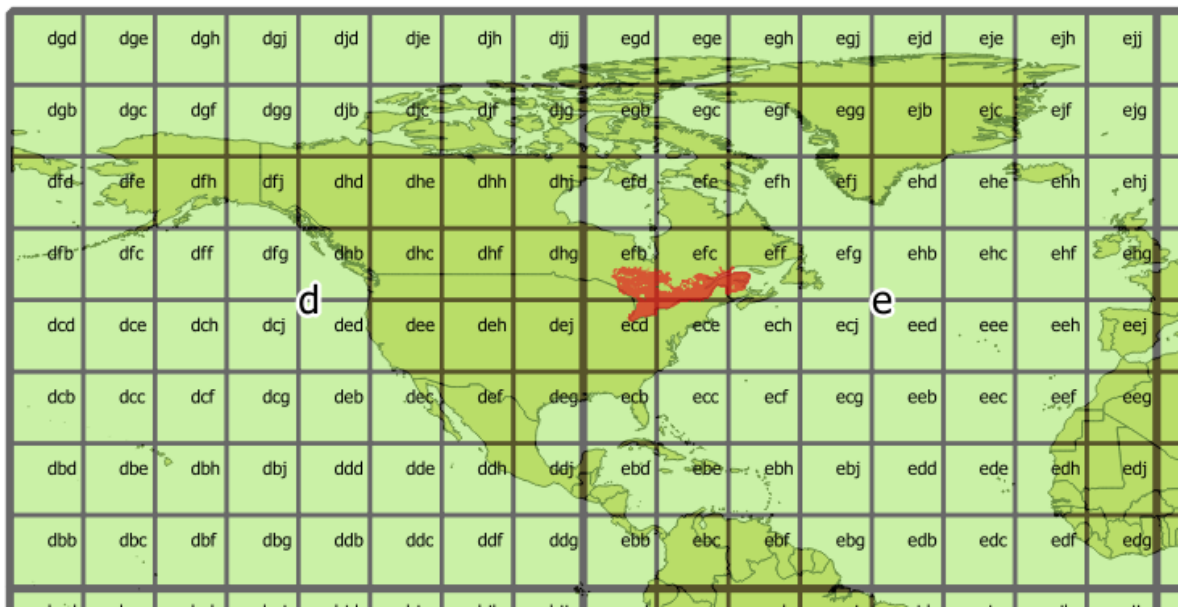


Figure 36 Jeu de données et géo-hachage de niveaux 1 et 3

Le nombre de partitions est beaucoup plus élevé pour les niveaux de hachage 7 et 11. L'image suivante montre le découpage de niveau 11 à gauche et 7 à droite (une seule partition). Le partitionnement utilisé divise chaque partition en huit à chaque niveau de profondeur. Par exemple, la partition « efc » contient 4096 partitions de niveau 7 et plus de 16 millions de partitions de niveau 11. Précisons que les partitions vides ne seront pas créées. Il y en aura suffisamment de partitions pour obtenir les deux effets suivants :

- Le nombre plus élevé de partitions que de cœurs permet de mettre à contribution tous les cœurs et c'est pourquoi on observe des gains sur les débits avec l'augmentation du nombre de cœurs.
- Le système pourra équilibrer les données sur les cœurs. Cet effet sera discuté plus en détail dans la troisième constatation.

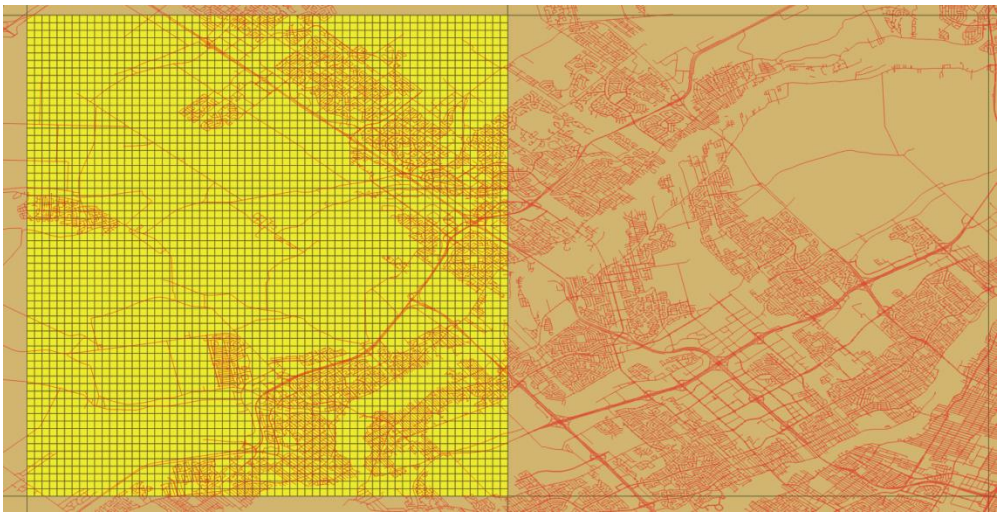


Figure 37 Géo-hachage de niveaux 7 et 11

4.2.3 Troisième observation

Le géo-hachage de niveau 11 ne présente aucun gain par rapport au géo-hachage de niveau 9. Le système Flink tente de répartir les partitions sur les cœurs de façon équilibrée. Comme il y a beaucoup de partitions de niveaux 7 et 11, chacun des cœurs recevra un nombre égal d'enregistrements. Si le niveau de partitionnement est tel que Flink parvient à bien faire l'équilibre, alors il n'y a plus de gain à augmenter le niveau de partitionnement. Le graphique suivant représente la distribution en pourcentage des enregistrements selon le niveau de partitionnement pour un parallélisme de 10 cœurs. Il est possible de constater un débalancement pour les hachages de niveaux 1 et 3, tandis que les données sont bien réparties pour les niveaux 7 et 10.

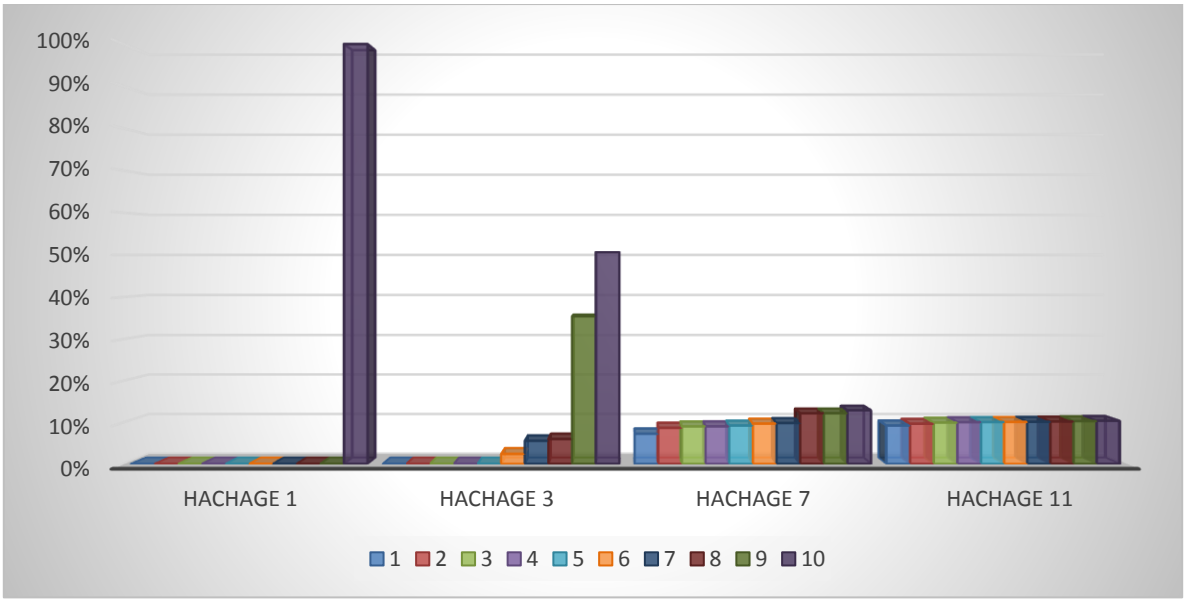


Figure 38 Distribution des données par cœurs selon le niveau de géo-hachage pour un parallélisme de 10 cœurs

4.3 Résultats du prototype de l'analyse de proximité

Les résultats de tests effectués pour le deuxième opérateur spatial, l'analyse des k plus proches voisins, sont présentés dans le tableau 12 et la figure 39. Le tableau présente la moyenne du nombre d'enregistrements par seconde à l'entrée du traitement ainsi que l'écart-type. Ces valeurs sont en fonction du niveau de géo-hachage et du nombre de cœurs.

Tableau 12 Résultats de prototype d'analyse de proximité

Niveau de géo-hachage	Symétrique 7		Symétrique 9		Symétrique 11		Asymétrique 7-9-11	
Nb. de cœurs	Moyenne	Écart-Type	Moyenne	Écart-Type	Moyenne	Écart-Type	Moyenne	Écart-Type
4	30730	35156	112970	11724	81140	5361	101035	26358
10	49266	48655	170659	10251	142289	7508	164316	19121
16	65014	33249	208342	10129	163200	6442	185082	9001
24	68699	34870	221670	10456	184956	7070	199430	14791

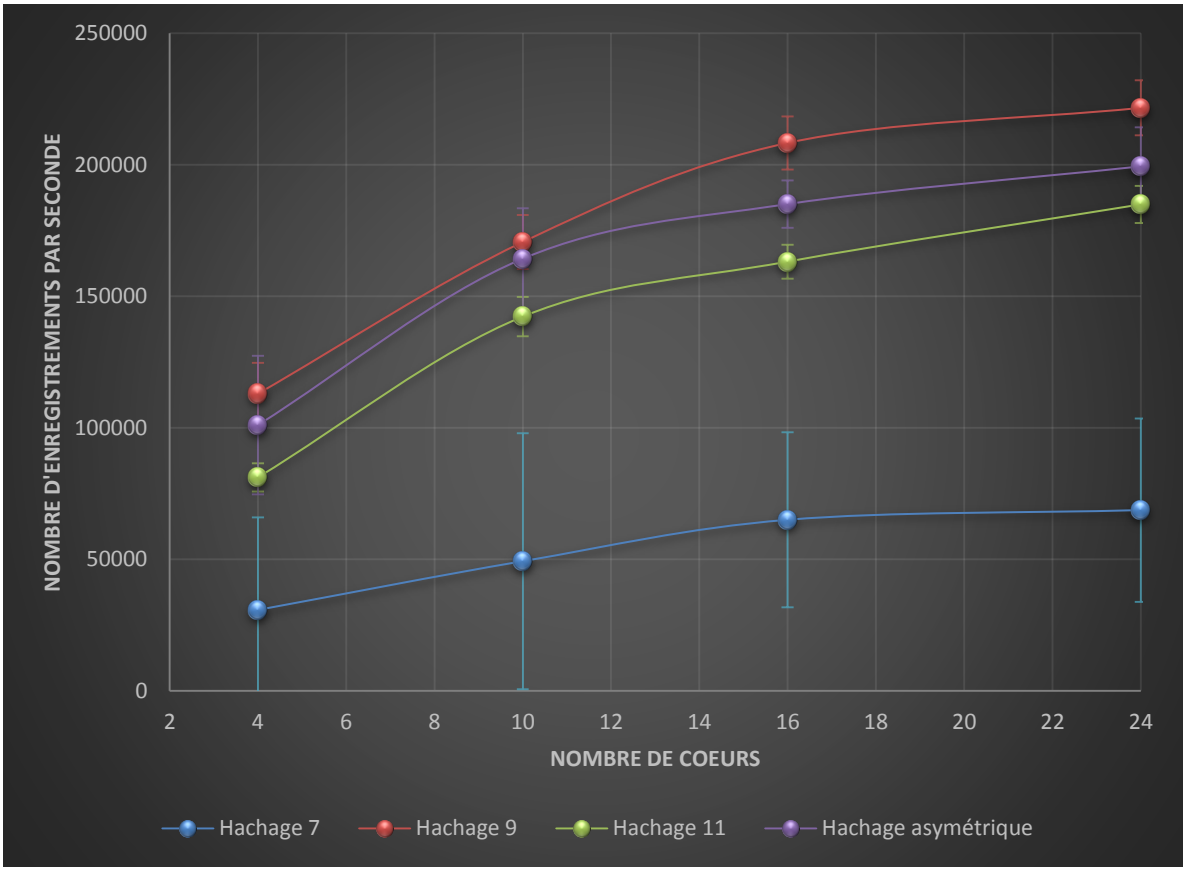


Figure 39 Évolution du débit selon le parallélisme comparé à la profondeur de partitionnement

Les observations suivantes peuvent être faites à partir des résultats :

- Le géo-hachage de niveau 7 présente de grandes valeurs d'écart-type;
- Le taux d'augmentation du débit semble décroitre avec l'augmentation du nombre de cœurs;
- Le géo-hachage de niveau 9 est plus performant que celui de niveau 11;
- Le géo-hachage asymétrique est moins performant que celui de niveau 9;

4.4 Analyse des résultats du prototype d'analyse de proximité

Dans cette section, les observations précédentes seront analysées afin d'interpréter les résultats. Cependant, pour bien comprendre l'interprétation, certaines notions sont rappelées.

4.4.1 Rappel de notions

4.4.1.1 Point en périphérie des partitions

L'analyse de proximité requiert une translation des points pour s'assurer que ceux situés en périphérie extérieure des partitions seront aussi évalués comme voisin. Pour ce faire, une translation théorique est effectuée comme

celle proposée dans la méthode d'adaptation du chapitre 2 (méthode à 2 translations). En conséquence, le nombre d'enregistrements sera plus élevé après la phase de géo-hachage. Plus le niveau de géo-hachage est profond, plus les partitions sont petites et, par conséquent, un plus grand nombre de points se trouveront en périphérie, ce qui augmentera le nombre d'enregistrements. La figure suivante représente le ratio entre le nombre d'enregistrements à l'entrée et le nombre à la sortie du géo-hachage ou du repartitionnement pour le cas asymétrique.

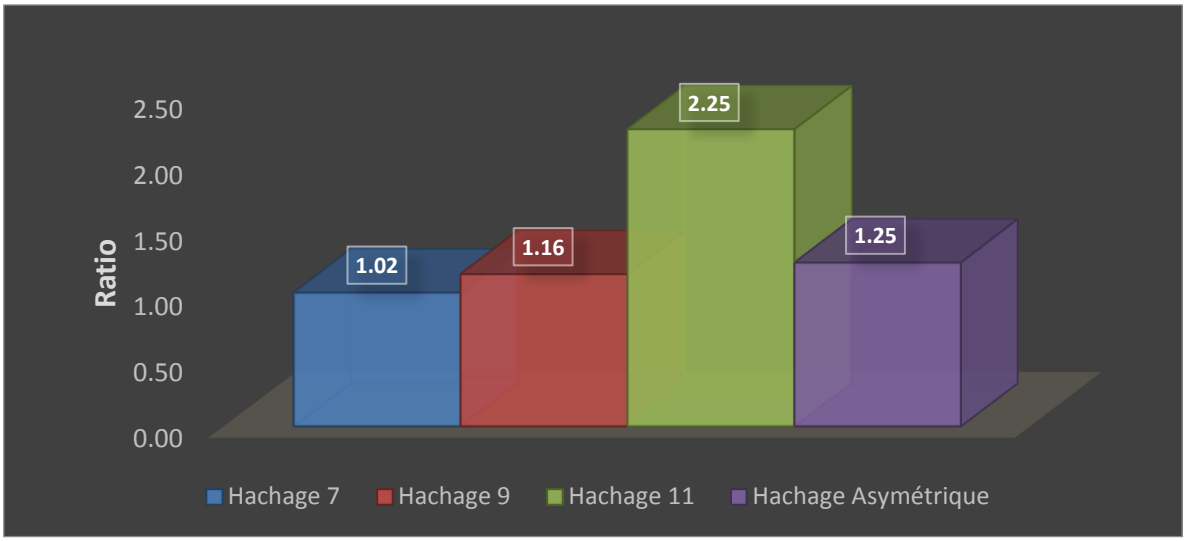


Figure 40 Ratio du nombre d'enregistrements après le géo-hachages vs les enregistrements initiaux

4.4.1.2 Coût de base intrinsèque au système

Un facteur qui n'est pas à négliger est le coût de base en temps de traitement et en ressource d'une exécution de code. Par exemple, lorsque Flink traite une fenêtre de temps contenant "n" données, il faut compter le temps de traitement de celles-ci (discuté plus bas), mais aussi le temps pour l'allocation et les ressources pour créer ces fenêtres. Il y aura autant de fenêtres que de partitions, les données étant groupées (keyBy) par partition. En augmentant le niveau de géo-hachage, le nombre de partitions augmente aussi, ce qui demande plus de ressources. La figure suivante montre le nombre moyen de partitions devant être traitée par intervalle de 30 secondes sous la forme de fenêtre. Ces données ont été obtenues à l'aide de données de performance calculées dans l'opération KNN. Elles permettent de connaître l'étiquette de temps de la fenêtre, son nombre de points et le temps de traitement.

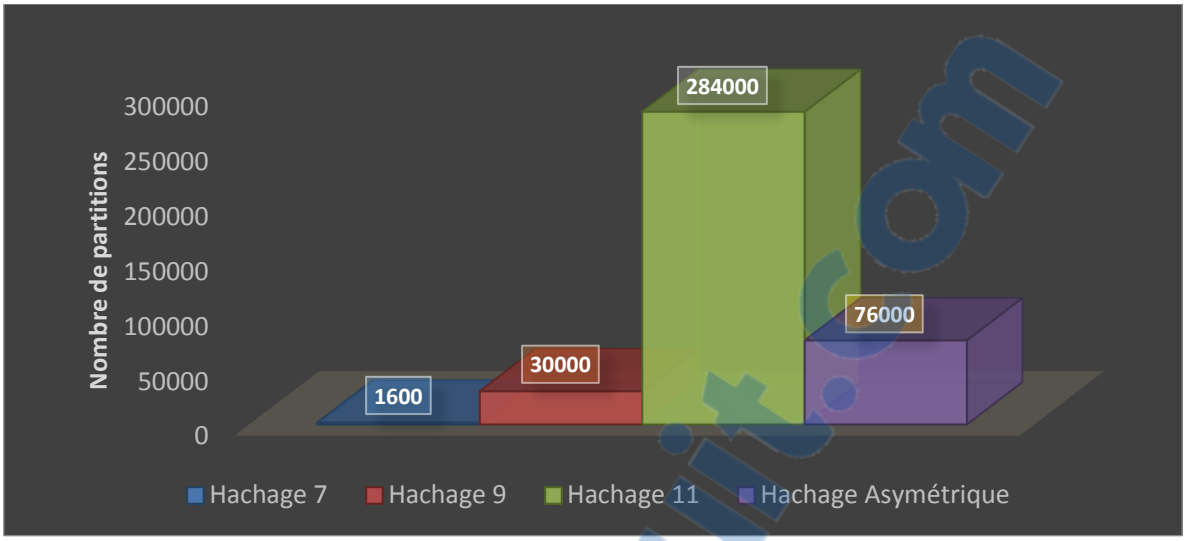


Figure 41 Nombre de partitions traitées par fenêtre de 30 secondes en fonction du niveau de géo-hachage

4.4.1.3 Progressivité du temps de traitement en fonction du nombre d'éléments

Les algorithmes n'ont pas tous le même type de complexité. L'analyse des k plus proches voisins est traitée dans une fenêtre dans notre prototype. Il est intéressant de connaître le type de progression du temps de traitement en fonction du nombre de données à traiter. Le type de complexité peut être trouvé par une analyse algorithmique ou empirique. Pour déterminer la complexité, une approche empirique a été retenue. Le résultat présenté à la figure suivante montre une complexité quadratique (n^2) pour la fenêtre d'opération analysant les kNN. Diviser une partition avec un niveau de géo-hachage plus élevé devrait donc permettre de diminuer le temps de traitement global. Le résultat ci-dessous a été obtenu par une simulation de l'algorithme faite à l'extérieur du prototype et n'a pour objectif que de déterminer l'allure de celle-ci. Cependant, la Figure 43 montre que le poids de chacun des coefficients a un effet sur la croissance du temps de traitement. Celui-ci serait constante, linéaire ou quadratique selon le nombre de points.

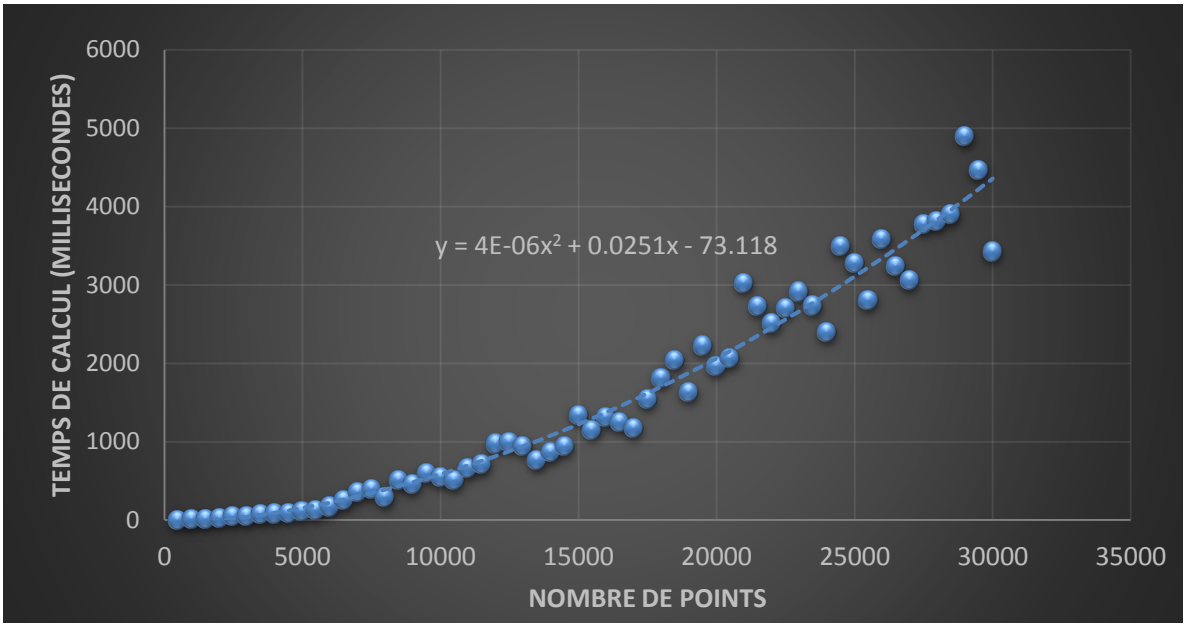


Figure 42 Progression du temps de calcul de l'algorithme pour la fenêtre KNN utilisé dans Flink

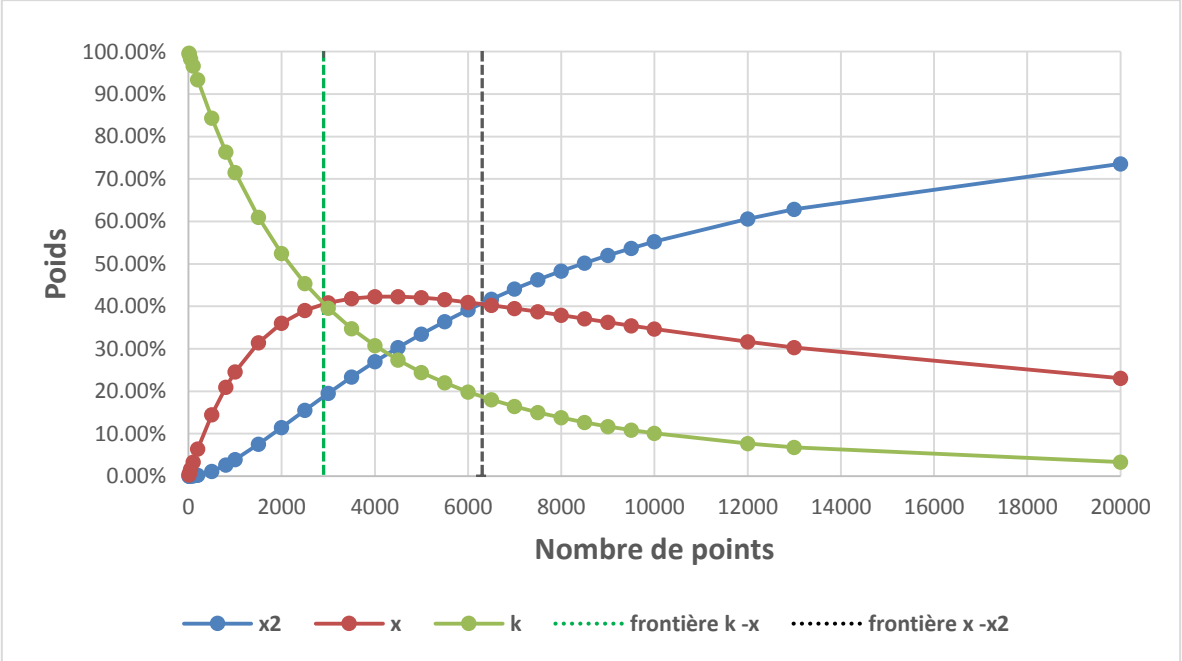


Figure 43 Poids des coefficients sur le temps de calcul en fonction du nombre de points

4.4.1.4 Fil d'exécution

Les calculs des opérations s'exécutent en fil d'exécution (thread) et la rapidité d'exécution sera dépendante du nombre de fils d'exécution actifs sur un cœur. La progression montrée aux graphiques précédents représente donc une tendance, puisque dans le prototype il y a plusieurs fils d'exécution en même temps. De plus, les fils

d'exécution peuvent ne pas tous être démarrés en même temps. Afin de ne pas surcharger le système, celui-ci peut démarrer un certain nombre de fils d'exécution et démarrer les autres au fur et à mesure que les premiers fils d'exécution se terminent. Le temps pour exécuter toutes les fenêtres sera supérieur au temps de la fenêtre la plus longue.

4.4.2 Analyse des observations

Les quatre observations de l'analyse de proximité sont présentées dans cette section.

4.4.2.1 Première observation

Le géo-hachage de niveau 7 présente de grandes valeurs d'écart-type, celles-ci représentent une instabilité dans le débit. La figure 44 montre des délais de traitement entre 2,5 et 10 secondes, certaines allant même jusqu'à 50 secondes. Ces délais entrainent un phénomène important de contre-pression qui se traduit par un arrêt momentané dans l'ingestion des données. Ces fluctuations sont à la source des écarts-types qui sont observés.

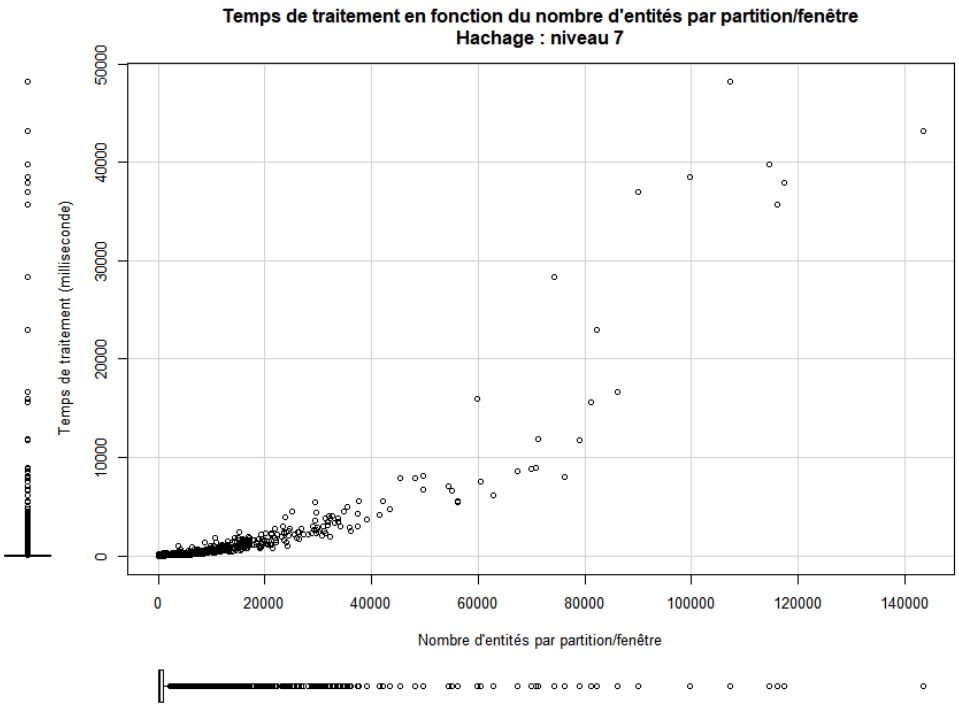


Figure 44 Temps de traitement vs nombre d'entités – géo-hachage 7

4.4.2.2 Deuxième observation

Le taux d'augmentation du débit semble décroitre avec l'augmentation du nombre de cœurs. Nous pensons que la diminution du taux de croissance du débit serait causée par des zones de grande densité. Ces zones pourraient varier dans le temps au gré des déplacements des véhicules, causées notamment par des goulots

d'étranglement routier ou par une simulation de densité non réaliste pour un réseau routier. Mais peu importe le degré de réalité de la simulation, ce phénomène pourrait exister dans d'autres contextes. Les partitions devant être plus grandes que le rayon de recherche, ces zones denses seraient imperméables à l'augmentation de la profondeur de géo-hachage. Ces zones rendraient inefficace, localement, le traitement en parallèle et ralentiraient les fils d'exécution du système jusqu'à bloquer les opérateurs en amont en causant de la contre-pression. Lors des tests, il a été observé que certains opérateurs pouvaient prendre jusqu'à une minute pour s'arrêter après l'envoi de la commande d'annulation. De plus, il n'est pas clair si ces opérateurs se sont terminés d'eux-mêmes ou s'ils ont été supprimés par le système après un certain délai. Il est aussi arrivé de perdre un nœud après une commande d'annulation dû à un opérateur qui ne se terminait pas. L'analyse de certains fichiers de sortie a permis d'observer ces zones de haute densité. Par exemple, des partitions qui ont pu être complétées possédaient pour des niveaux de géo-hachage de 7, 9 et 11 un nombre d'entités de 150 000, 40 000 et 7000, respectivement. Afin de vérifier le phénomène, un test préliminaire a été effectué. Le graphique suivant montre l'évolution de la méthode normale et l'évolution d'une méthode où les partitions ayant plus de 2000 points ont été ignorées. Avec la méthode originale, on remarque une diminution du taux d'augmentation, comme ce que l'on observe dans les résultats. Cependant, la courbe où les partitions supérieures à 2000 points ont été éliminées possède une progression linéaire. Le graphique suggère donc un lien entre le taux de progression et la densité des données. De plus, la Figure 43 montre une croissance constante, linéaire ou quadratique du temps de calcul des fenêtres selon le nombre de points. Des stratégies d'indexation locale pourraient être envisagées. Ce phénomène aurait le mérite d'être étudié plus à fond, car la méthode de géo-hachage ne semble pas adaptée à ce contexte particulier. Ce test a été fait avec un ordinateur différent (Dell Aurora R6 – i7-7700K) et conséquemment, seules les tendances doivent être regardées.

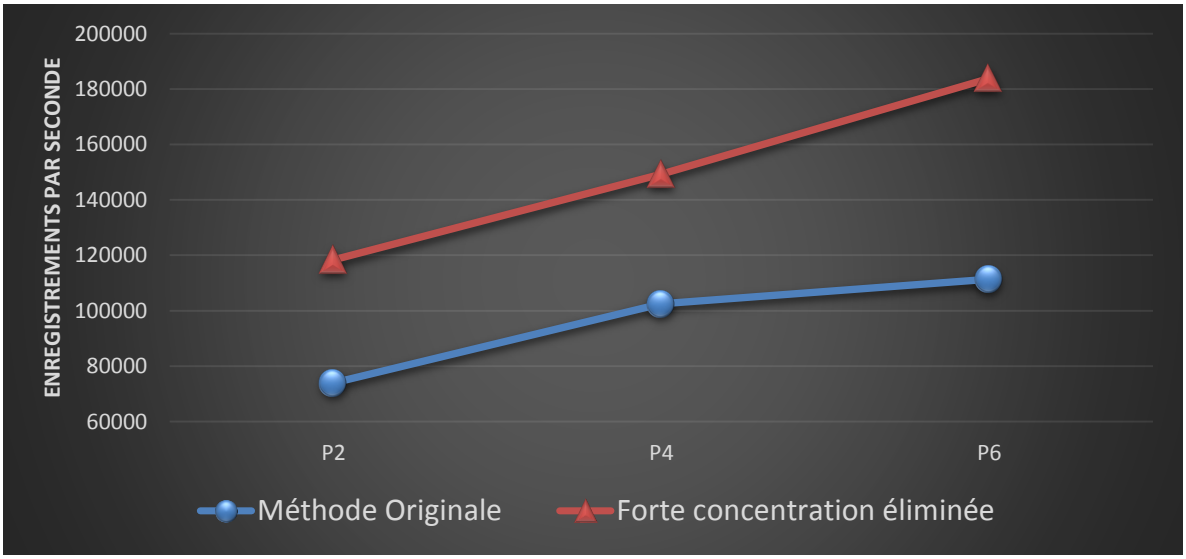


Figure 45 Élimination des zones de forte densité vs méthode originale – progression en fonction du nombre de cœurs

4.4.2.3 Troisième observation

Le géo-hachage de niveau 9 est plus performant que celui de niveau 11. L'observation de la Figure 46 et de la Figure 47 permet de constater une diminution du nombre de points par partition et de la durée de calcul lors du passage du niveau de géo-hachage 9 à 11. Cette observation pourrait laisser croire que le géo-hachage de niveau 11 aurait produit un meilleur débit. Cependant, il faut tenir compte de certains autres phénomènes. Premièrement, le géo-hachage de niveau 11 produira un plus grand nombre de points puisque les partitions sont plus petites (Figure 40). Deuxièmement, le géo-hachage de niveau 11 produira 10 fois plus de partitions que celui de niveau 9 (Figure 41). Ces deux phénomènes entraineront donc une plus grande pression sur les ressources nécessaires au traitement. À terme, bien qu'il soit plus rapide de calculer la fenêtre de KNN avec un nombre de points inférieur, la quantité de partitions créées augmentera le nombre de fils d'exécution devant être traités. Le débit en fonction du niveau de géo-hachage passe donc par un maximum avant de redescendre.

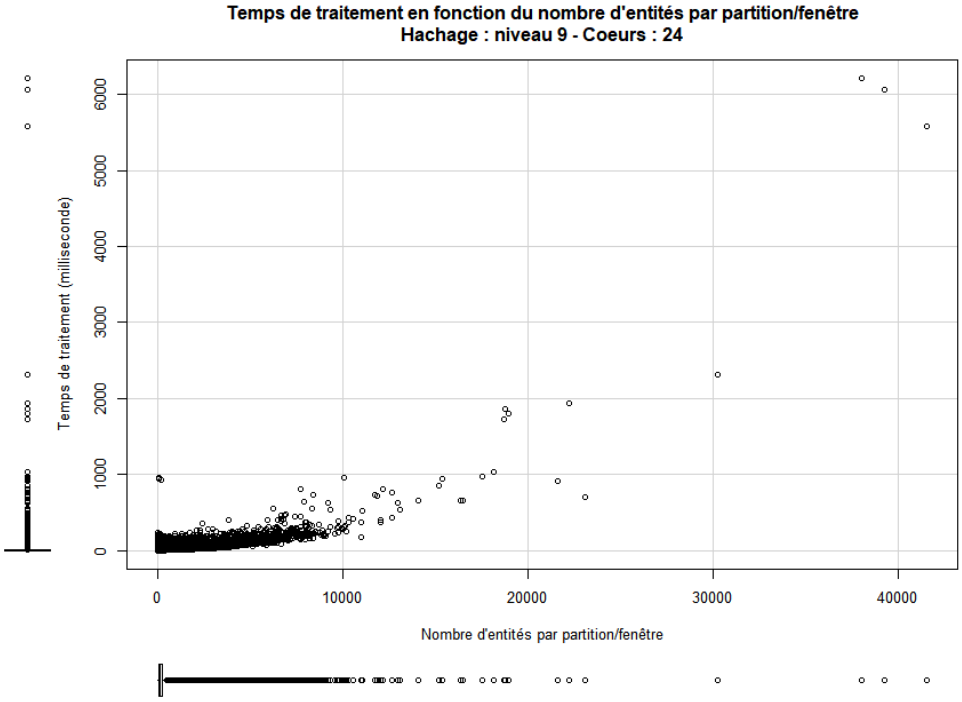


Figure 46 Temps de traitement vs nombre d'entités – géo-hachage 9

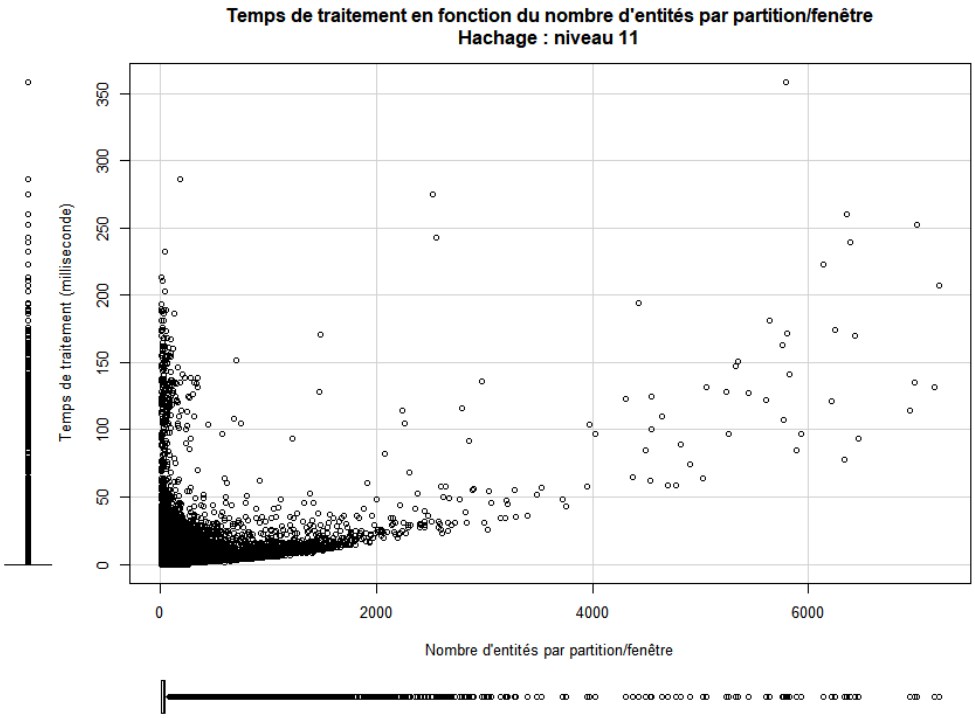


Figure 47 Temps de traitement vs nombre d'entités – géo-hachage 11

4.4.2.4 Quatrième observation

Le géo-hachage asymétrique est moins performant que celui de niveau 9. Dans le géo-hachage asymétrique, les partitions sont géo-hachées à divers niveaux selon le nombre d'éléments dans la partition mère. Le géo-hachage asymétrique a été introduit afin de profiter de l'effet bénéfique de la réduction du nombre de points sur le temps de calcul lors de l'augmentation de la profondeur du géo-hachage. Il a aussi été réfléchi pour minimiser le nombre de points créés par les points en périphérie de partition et pour minimiser le nombre de partitions. Les Figure 40, 41 etFigure 48 montrent bien ces compromis.

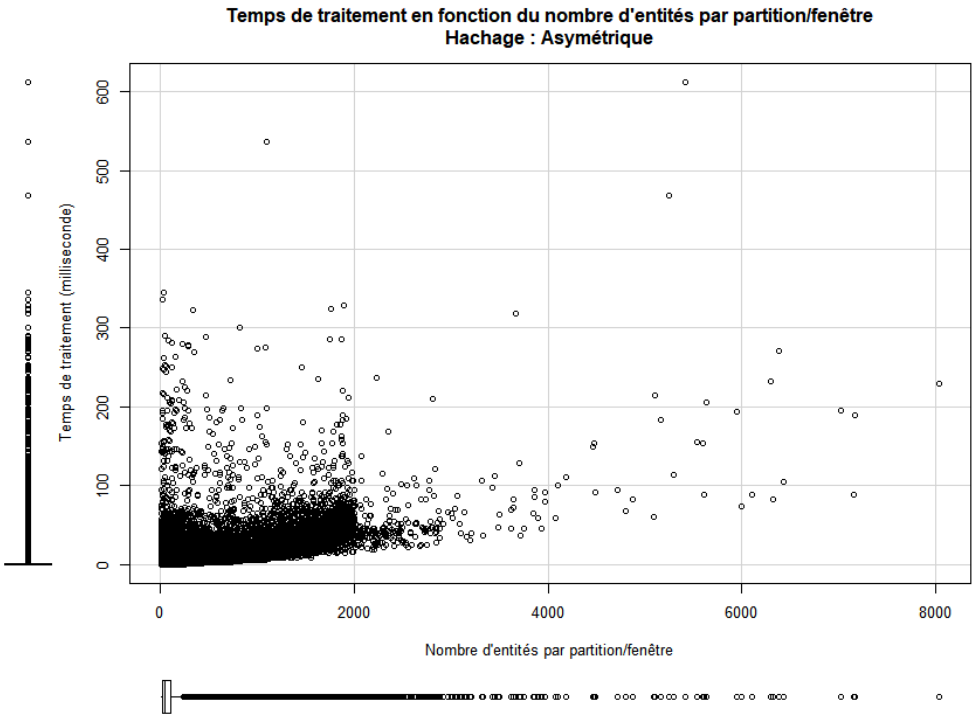


Figure 48 Temps de traitement vs nombre d'entités – géo-hachage asymétrique

Le temps de calcul est majoritairement en bas de 100 millisecondes. De plus, comparé avec le niveau 9 de géo-hachage qui produit 1,16 fois le nombre d'enregistrements initial, le géo-hachage asymétrique en produit 1,25 fois. Finalement, 2,5 fois plus de partitions sont créées comparativement au niveau 7 versus 10 fois plus pour le niveau 11.

Bien que cette méthode performe mieux que le géo-hachage de niveau 11, elle est en deçà des performances du niveau 9. L'analyse des métriques montre que, pour le géo-hachage symétrique, la contre-pression se situe au niveau de la fenêtre de traitement calculant le KNN, tandis que pour le géo-hachage asymétrique, la contre-pression se situe au niveau du repartitionnement. L'analyse du code n'a pas permis d'observer la raison du goulot d'étranglement. Cependant, en exécutant la séquence d'opérations, mais en supprimant la fenêtre KNN, les données ont permis de constater une augmentation du débit de plus de 2 fois. Cette observation laisse croire que le goulot d'étranglement observé n'est pas intrinsèquement dû à la fonction de repartitionnement. Cette fonction de repartitionnement a certainement un effet sur l'allocation des ressources. On peut en déduire que l'ajout de l'opération de repartitionnement n'a pas un effet bénéfique global sur le débit du système malgré les gains observés sur le nombre de points par partition, la quantité de partitions créées et sur le nombre d'enregistrements.

4.5 Conclusion

Le prototype avait pour objectif d'évaluer les variations de performance, mesurée en nombre d'enregistrements par seconde, en fonction du degré de partitionnement et du nombre de cœurs alloués aux traitements. Le contexte étudié était le traitement d'opérateurs spatiaux binaires dont les opérandes étaient tous deux issus de flux massifs.

Les traitements étudiés dans ce contexte étaient une opération d'intersection ainsi qu'une opération de proximité sous la forme des k plus proches voisins à l'intérieur d'un rayon maximal fixé.

L'analyse des résultats de l'opérateur d'intersection a permis de constater les choses suivantes :

- Des gains systématiques avec l'augmentation du nombre de cœurs, en utilisant un niveau de partitionnement permettant le balancement adéquat des données.
- Aucun gain n'est perceptible avec un niveau de partitionnement plus élevé après que le balancement ait été atteint.

L'analyse des résultats du prototype de l'opérateur de proximité a mis en lumière les éléments suivants :

- Un opérateur ayant un temps de latence plus élevé que le temps qui lui est alloué entrainera des effets de contre-pression pouvant mener à de grandes fluctuations dans l'ingestion des données.
- La densité des données à traiter a un impact sur les capacités de traitement en parallèle utilisant un modèle de partitionnement par grille.
- Le gain lié à l'augmentation du niveau de partitionnement passe par un maximum au-delà duquel les performances diminuent. Ce maximum est causé par deux effets qui s'opposent. D'une part, la diminution du nombre de points par fenêtre augmente les performances. D'autre part, l'augmentation du nombre de partitions crée de la pression sur les ressources.
- Les gains réalisés par le partitionnement asymétrique sur le nombre de points par partition et sur les ressources sont moins élevés que le coût des calculs supplémentaires requis.

Dans ces deux traitements, on constate une augmentation des performances en fonction du nombre de cœurs. Cependant, le niveau de partitionnement n'a pas le même impact. La période d'accumulation étant différente pour chacun d'entre eux, il est difficile de les comparer et de tirer des conclusions définitives. À cet égard, d'autres recherches sont donc nécessaires.

Une analyse de la variation des performances en fonction du temps d'accumulation pourrait être faite, afin de voir l'effet de celle-ci sur les performances. De plus, lors des cycles de validation, nous avons constaté un impact sur les performances liées aux formats internes des données et aux différentes librairies spatiales utilisées. Ces facteurs constituent autant de voies à explorer.

Chapitre 5 Conclusion générale

5.1 Conclusion

Dans le cadre de cette recherche, nous avons fait ressortir que les facteurs suivants sont des éléments clés permettant une classification des cas de figure pouvant être rencontrés dans les traitements spatiaux de flux massifs de données géolocalisées :

- L'arité géométrique des opérateurs spatiaux;
- L'attribut relationnel des opérandes d'un opérateur;
- La propriété temporelle des opérandes;
- La propriété quantitative des opérandes.

Sur la base de ces facteurs, une classification a été proposée et la complexité de traitement a été évaluée en fonction des méthodes générales de traitement proposées.

Après avoir réalisé les deux premiers objectifs, le troisième consistait à l'élaboration de méthodes de traitement détaillées pour les cas de traitement d'opérateurs binaires ayant des opérandes issus de flux massifs. Cette famille de traitement requérait un traitement par fenêtre et un partitionnement, spatial ou non spatial, des données. Le partitionnement spatial dans un contexte de flux, a été le sujet analysé, les deux autres étant déjà des éléments connus dans la littérature. La méthode proposée a été un partitionnement par clé de géo-hachage. Cette méthode a été choisie pour sa rapidité et le fait qu'il soit possible d'obtenir une clé de partitionnement sans faire un prétraitement des données, ce qui est un avantage dans les systèmes à faible temps de latence.

Afin de tester ces méthodes, 2 opérateurs spatiaux ont été développés et testés à l'aide d'un prototype. Les résultats montrent l'importance de calibrer le niveau de partitionnement pour le balancement des données sur les cœurs afin de profiter au maximum de la capacité de parallélisation des nœuds du système. Ils ont aussi montré que la densité des données traitées avait un impact sur les capacités du traitement en parallèle utilisant un modèle de partitionnement par grille.

5.2 Contribution

Nos travaux de recherche ont fait ressortir les facteurs influençant l'adaptation des traitements spatiaux dans un contexte de traitement en parallèle de flux massif de données. Nous avons déterminé que les méthodes d'adaptation peuvent se décliner en classes sur la base des caractéristiques de l'opérateur, mais aussi de la nature des données mises en relation par l'opérateur.

Nous avons proposé des méthodes générales de traitement pour chacune des classes identifiées afin de guider les stratégies d'adaptation. Pour l'une de ces classes, le traitement d'opérateur binaire ayant des opérandes issus de flux massifs, nous avons détaillé une méthode d'adaptation permettant l'utilisation d'opérateurs spatiaux.

Afin de tester l'efficacité et la validité de la méthode proposée, nous avons appliqué cette méthode à un opérateur relationnel d'intersection et un opérateur d'analyse de proximité, soit les "k" plus proches voisins. Ces tests ont permis de vérifier la validité et de quantifier l'efficacité des méthodes proposée par rapport à l'évolution, ou scalabilité, horizontale du système (l'augmentation du nombre de cœurs). Nos tests ont aussi permis de quantifier l'effet de la variation du niveau de partitionnement sur les performances du débit de traitement. Notre contribution permettra, nous l'espérons, de servir de point de départ pour l'adaptation d'opérateur spatial plus complexe.

5.3 Limites de la recherche

Ce mémoire avait pour objectif de développer des méthodes permettant d'effectuer des traitements spatiaux provenant de flux massif de données. Le cadre de cette recherche se limitait aux données de type vectoriel. Il est donc peu probable que les méthodes proposées soient suffisantes ou adaptées à la donnée de type matriciel ou à des nuages de points.

De plus, la classification des familles de traitement est basée sur l'analyse de traitements spatiaux de base. Bien que certains types de traitements spatiaux plus complexes aient été brièvement abordés dans ce mémoire, ils ont été exclus du cadre de cette recherche.

Les méthodes de traitement proposées comportent aussi certaines limites. Premièrement, la technique de géo-hachage utilisé ne permet pas d'obtenir des partitions équivalentes lorsque la zone d'observation est étendue, par exemple à l'échelle planétaire ou continentale, due aux déformations causées par la projection. Cependant, cet effet peut être amoindri en utilisant un niveau de partitionnement plus profond ou si les observations sont effectuées sur une zone plus restreinte. Deuxièmement, la technique de géo-hachage proposée comporte aussi une autre problématique lorsque les observations se situent aux pôles. À ces endroits, le système de coordonnées et de calcul des index est soumis à une singularité géographique et les calculs de translation permettant de calculer les « k » plus proches voisins ne fonctionneraient plus. Ces problèmes pourraient être résolus avec un système de géo-hachage basé sur des cellules de Voronoï ou une tessellation de l'ellipsoïde terrestre.

Bien que nous ayons expliqué que l'architecture du banc de test ne causait pas de problèmes de performance sur les traitements observés, celle-ci ne représente pas fidèlement la réalité à plusieurs égards. Les données

de flux étaient chargées à partir d'un fichier afin de tester le système en charge maximum. Le flux était généré par la vitesse de lecture du fichier. Il aurait été intéressant que la source des données soit aussi sous forme de flux, cependant les limites de bande passante de l'architecture réseau et le nombre d'ordinateurs disponibles n'ont pas permis d'effectuer les tests de cette façon.

Lors de nos tests préliminaires, nous avons testé différentes structures de données, tant pour la source des données que pour leur structure interne. Bien que l'analyse des différences de performance entre différentes structures ait été exclue du cadre de discussion de ce mémoire, nous avons remarqué un certain impact sur le débit maximal du flux de traitement. Cet impact se situe principalement au niveau de la bande passante. Certains traitements nécessitent une répartition des données sur les nœuds et le poids de celles-ci a un impact direct sur le niveau d'utilisation maximal de la bande passante.

Finalement, les données sources ont été obtenues à l'aide d'une simulation de véhicules voyageant sur un réseau routier. Les positions initiales et finales ont été générées aléatoirement, elles ne reflètent donc pas la réalité. De plus, certaines densités de partition obtenue sont physiquement impossibles. Néanmoins, l'exemple du déplacement de véhicules sur un réseau routier n'était qu'un prétexte pour tester les méthodes proposées. Cet exemple, bien que non réaliste, ne diminue pas la qualité des résultats obtenus.

5.4 Perspectives

Les méthodes proposées permettent déjà d'effectuer des traitements spatiaux de base sur des flux massifs de données. Il est possible de chainer successivement ces traitements afin d'en effectuer des plus complexes. Certains tests préliminaires, non présentés dans ce mémoire, nous ont permis, par exemple, d'effectuer des analyses de regroupement (clustering) et de produire des polygones concaves (α-shape) représentant ces regroupements.

Il serait intéressant de poursuivre l'analyse pour inclure d'autres types de traitements spatiaux plus complexes étant utilisés dans un contexte temporel. Ces traitements incluent, entre autres, la détection les patrons spatio-temporels tels que :

- l'attroupement (flock);
- les rencontres (meet);
- la répétition (periodic pattern);
- les emplacements fréquents (frequent location).

Une analyse des opérateurs spatiaux plus complexes ferait certainement ressortir d'autres facteurs d'influence qui permettraient de raffiner la classification proposée.

Présentement, nous croyons que les sources de données pouvant satisfaire les critères d'un flux massif de données spatiales sont principalement celles provenant d'objets connectés et mobiles, ou celles provenant d'objets fixes, mais récoltants des données sur les objets mobiles passant à proximité. Une grande proportion de ces données est représentée par le déplacement d'individus. Ceci pose un problème d'éthique sur le droit à la vie privée des individus et les questions d'anonymisation des informations dans un contexte de positionnement géographique sont tous à propos.

Dans ce mémoire, nous avons seulement exploré les données vectorielles. Cependant, les données de type matriciel (image satellite) et les nuages de points (point cloud) sont des candidates idéales pour les traitements spatiaux de flux massif de données. Imaginons seulement le traitement, en temps réel, de point lidar provenant de centaines de véhicules autonomes afin d'obtenir un modèle tridimensionnel fluctuant dans le temps pour un secteur donné.

L'intelligence artificielle est une grande consommatrice de données massives. Il deviendra important de pouvoir mettre en relation des flux massifs et des données massives statiques. Notre classification a permis de faire ressortir le niveau de complexité de ce type de traitements.

Finalement, la représentation cartographique et dynamique des flux de données générés par ces traitements est une question importante qui n'a pas été abordée dans ce mémoire. Bien que le traitement de données massives ait principalement pour objectif de résumer et d'extraire l'information, il demeure que la quantité de données produites peut être volumineuse. De plus, le traitement de flux de données produit des résultats sous la forme de flux. La cartographie dynamique devra donc aborder les caractéristiques temporelles et celles du volume.

Références

Aggarwal, A. *et al.* (1988) « Parallel computational geometry », *Algorithmica*, 3(1‑4), p. 293‑327.

Aji, A. *et al.* (2013) « Hadoop GIS: A High Performance Spatial Data Warehousing System over Mapreduce », *Proc. VLDB Endow.*, 6(11), p. 1009–1020.

Ali, M. *et al.* (2010) « Real-time Spatio-temporal Analytics Using Microsoft StreamInsight », dans *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*. New York, NY, USA: ACM (GIS '10), p. 542–543.

Apache Flink (2017). Disponible à: https://flink.apache.org/ (Consulté le: 17 juillet 2017).

Apache Kafka (2017). Disponible à: https://kafka.apache.org/.

Apache Spark (2017). Disponible à: http://spark.apache.org/ (Consulté le: 17 juillet 2017).

Apache Storm (2017). Disponible à: http://storm.apache.org/ (Consulté le: 17 juillet 2017).

Arasu, A. *et al.* (2004) « Characterizing Memory Requirements for Queries over Continuous Data Streams », *ACM Trans. Database Syst.*, 29(1), p. 162–194.

Babcock, B. *et al.* (2002) « Models and Issues in Data Stream Systems », dans *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. New York, NY, USA: ACM (PODS '02), p. 1–16.

Berg, M. de *et al.* (2008) *Computational Geometry: Algorithms and Applications*. 3rd ed. Santa Clara, CA, USA: Springer-Verlag TELOS.

Bifet, A. *et al.* (2011) « MOA-TweetReader: Real-time Analysis in Twitter Streaming Data », dans *Proceedings of the 14th International Conference on Discovery Science*. Berlin, Heidelberg: Springer-Verlag (DS'11), p. 46–60. Disponible à: http://dl.acm.org/citation.cfm?id=2050236.2050243.

Boxer, L. et Miller, R. (1988) « Dynamic Computational Geometry on Parallel Computers », dans *Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications - Volume 2*. New York, NY, USA: ACM (C³P), p. 1212–1219.

Celebi, U. (2015) *How Apache Flink™ handles backpressure*. Disponible à: https://data-artisans.com/blog/how-flink-handles-backpressure.

Clementini, E. et Di Felice, P. (1997) « A Global Framework for Qualitative Shape Description », *GeoInformatica*, 1(1), p. 11‑27.

Clementini, E. et Di Felice, P. (2000) « Spatial Operators », *SIGMOD Rec.*, 29(3), p. 31–38.

Costa, J. *et al.* (2011) « Blending OLAP Processing with Real-Time Data Streams », dans Yu, J., Kim, M., et Unland, R. (éd.) *Database Systems for Advanced Applications*. Springer Berlin Heidelberg (Lecture Notes in Computer Science), p. 446‑449.

Dean, J. et Ghemawat, S. (2008) « MapReduce: Simplified Data Processing on Large Clusters », *Commun. ACM*, 51(1), p. 107–113.

Dehne, F. (1994) « Scalable parallel computational geometry », dans Cosnard, M., Ferreira, A., et Peters, J. (éd.) *Parallel and Distributed Computing Theory and Practice*. Springer Berlin Heidelberg (Lecture Notes in Computer Science), p. 115‑119.

Dehne, F. et Sack, J.-R. (1989) « A survey of parallel computational geometry algorithms », dans Wolf, G., Legendi, T., et Schendel, U. (éd.) *Parcella '88*. Springer Berlin Heidelberg (Lecture Notes in Computer Science), p. 73‑88.

Dumoulin, M. (2014) *Personalized Large Scale Classification of Public Tenders on Hadoop*. Université Laval. Disponible à: http://theses.ulaval.ca/archimede/fichiers/30887/30887.pdf (Consulté le: 17 juillet 2017).

Dyson, G. (2013) *George Dyson: No Time Is There— The Digital Universe and Why Things Appear To Be Speeding Up*. San Francisco. Disponible à: http://longnow.org/seminars/02013/mar/19/no-time-there-digital-universe-and-why-things-appear-be-speeding/ (Consulté le: 17 juillet 2017).

Egenhofer, M. J. (1994) « Spatial SQL: A Query and Presentation Language », *IEEE Transactions on Knowledge and Data Engineering*, 6, p. 86–95.

Eldawy, A. *et al.* (2013) « CG_Hadoop: Computational Geometry in MapReduce », dans *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. New York, NY, USA: ACM (SIGSPATIAL'13), p. 294–303.

Eldawy, A. (2014) « SpatialHadoop: Towards Flexible and Scalable Spatial Processing Using Mapreduce », dans *Proceedings of the 2014 SIGMOD PhD Symposium*. New York, NY, USA: ACM (SIGMOD'14 PhD Symposium), p. 46–50.

Eldawy, A. et Mokbel, M. F. (2013) « A Demonstration of SpatialHadoop: An Efficient Mapreduce Framework for Spatial Data », *Proc. VLDB Endow.*, 6(12), p. 1230–1233.

Eldawy, A. et Mokbel, M. F. (2015) « The Ecosystem of SpatialHadoop », *SIGSPATIAL Special*, 6(3), p. 3–10.

Galić, Z. *et al.* (2014) « Geospatial data streams: Formal framework and implementation », *Data & Knowledge Engineering*, 91, p. 1‑16.

Galić, Z. (2016a) *Spatio-Temporal Data Streams*. 1st ed. 2016 edition. New York, NY: Springer.

Galić, Z. (2016b) « Spatio-Temporal Data Streams and Big Data Paradigm », dans *Spatio-Temporal Data Streams*. New York, NY: Springer New York, p. 47–69.

Galić, Z. (2016c) « Spatio-Temporal Data Streams and Big Data Paradigm », dans *Spatio-Temporal Data Streams*. Springer, New York, NY (SpringerBriefs in Computer Science), p. 47–69. Disponible à: https://link.springer.com/chapter/10.1007/978-1-4939-6575-5_3.

Gama, J. (2010) *Knowledge Discovery from Data Streams*. 1st éd. Chapman & Hall/CRC.

Ghemawat, S. *et al.* (2003) « The Google file system », dans *ACM SIGOPS operating systems review*. ACM, p. 29–43. Disponible à: http://dl.acm.org/citation.cfm?id=945450 (Consulté le: 17 juillet 2017).

Giannotti, F. *et al.* (2011) « Unveiling the Complexity of Human Mobility by Querying and Mining Massive Trajectory Data », *The VLDB Journal*, 20(5), p. 695–719.

Golab, L. et Özsu, M. T. (2003) « Issues in Data Stream Management », *SIGMOD Rec.*, 32(2), p. 5–14.

Guller, M. (2015) *Big Data Analytics with Spark: A Practitioner's Guide to Using Spark for Large Scale Data Analysis*. 1st ed. edition. New York, New York: Apress.

Güting, R. H. *et al.* (2000) « A Foundation for Representing and Querying Moving Objects », *ACM Trans. Database Syst.*, 25(1), p. 1–42.

He, Y. *et al.* (2014) « MR-DBSCAN: a scalable MapReduce-based DBSCAN algorithm for heavily skewed data », *Frontiers of Computer Science*, 8(1), p. 83–99.

Heineman, G. T. *et al.* (2008) *Algorithms in a Nutshell*. 1 edition. Beijing ; Sebastopol Calif.: O'Reilly Media.

Hershberger, J. *et al.* (2009) « Summarizing Spatial Data Streams Using ClusterHulls », *J. Exp. Algorithmics*, 13, p. 4:2.4–4:2.28.

Huang, Y. et Zhang, C. (2008) « New Data Types and Operations to Support Geo-streams », dans Cova, T. et al. (éd.) *Geographic Information Science*. Springer Berlin Heidelberg (Lecture Notes in Computer Science), p. 106‑118.

Hurwitz, J. *et al.* (2013) *Big Data For Dummies*. 1st éd. Hoboken, NJ: John Wiley & Sons inc.

International Data Corporation (2014a) *The digital universe of opportunities*. International Data Corporation, p. 17.

International Data Corporation (2014b) *The Internet of Things*. Disponible à: https://canada.emc.com/leadership/digital-universe/2014iview/internet-of-things.htm (Consulté le: 17 juillet 2017).

Kazemitabar, S. J. *et al.* (2010) « Geospatial Stream Query Processing Using Microsoft SQL Server StreamInsight », *Proc. VLDB Endow.*, 3(1‑2), p. 1537–1540.

Kazemitabar, S. J. *et al.* (2011) « Geostreaming in Cloud », dans *Proceedings of the 2Nd ACM SIGSPATIAL International Workshop on GeoStreaming*. New York, NY, USA: ACM (IWGS '11), p. 3–9.

Landset, S. *et al.* (2015) « A survey of open source tools for machine learning with big data in the Hadoop ecosystem », *Journal of Big Data*, 2(1), p. 1–36.

Laney, D. (2001) *3D Data Management: Controlling Data Volume, Velocit, and Variety*. 949. META Group Inc., p. 4.

Lee, J.-G. et Kang, M. (2015) « Geospatial Big Data: Challenges and Opportunities », *Big Data Research*, 2(2), p. 74‑81.

Li, S. *et al.* (2015) « Geospatial big data handling theory and methods: A review and research challenges », *{ISPRS} Journal of Photogrammetry and Remote Sensing*, p.

Morton, G. M. (1966) *A computer Oriented Geodetic Data Base; and a New Technique in File Sequencing*. IBM Ltd.

Neis, P. et Zipf, A. (2012) « Analyzing the Contributor Activity of a Volunteered Geographic Information Project — The Case of OpenStreetMap », *ISPRS International Journal of Geo-Information*, 1(2), p. 146–165.

Nittel, S. (2015) « Real-time Sensor Data Streams », *SIGSPATIAL Special*, 7(2), p. 22–28.

Nittel, S. *et al.* (2012) « Real-time Spatial Interpolation of Continuous Phenomena Using Mobile Sensor Data Streams », dans *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*. New York, NY, USA: ACM (SIGSPATIAL '12), p. 530–533.

OECD (2015) *Big Data and Transport - Understanding and assessing options*. OECD, p. 66.

Open Geospatial Consortium (2010) « OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option ». Open Geospatial Consortium Inc.

Open Geospatial Consortium (2011) « OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture ». Open Geospatial Consortium Inc.

Sharifzadeh, M. et Shahabi, C. (2009) « Approximate Voronoi Cell Computation on Spatial Data Streams », *The VLDB Journal*, 18(1), p. 57–75.

Shekhar, S. *et al.* (2014) « Benchmarking Spatial Big Data », dans Rabl, T. et al. (éd.) *Specifying Big Data Benchmarks*. Springer Berlin Heidelberg (Lecture Notes in Computer Science), p. 81‑93.

Siddalingaiah;, J. V. S. W. M. (2014) *Pro Apache Hadoop by Jason Venner*. 2nd ed. 2014 edition. New York, NY: Apress.

de Smith, M. J. *et al.* (2015) *Geospatial Analysis - 5th edition*. 5th éd. eBook. Disponible à: http://www.spatialanalysisonline.com/.

Stonebraker, M. *et al.* (2005) « The 8 Requirements of Real-time Stream Processing », *SIGMOD Rec.*, 34(4), p. 42–47.

Tang, L.-A. *et al.* (2012) « A Framework of Traveling Companion Discovery on Trajectory Data Streams », *ACM Transaction on Intelligent Systems and Technology*. Disponible à: http://research.microsoft.com/apps/pubs/default.aspx?id=161707.

Wang, S. et Yuan, H. (2014) « Spatial Data Mining: A Perspective of Big Data », *International Journal of Data Warehousing and Mining (IJDWM)*, 10(4), p. 50‑70.

Wei, M. *et al.* (2016) *Streaming Report - Fonctional Comparison and Performance Evaluation*. Intel Corporation, p. 50.

Whitman, R. T. *et al.* (2014) « Spatial Indexing and Analytics on Hadoop », dans *Proceedings of the 22Nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. New York, NY, USA: ACM (SIGSPATIAL '14), p. 73–82.

Wikipedia (2014) *Courbe de Lebesgue*. Disponible à: https://fr.wikipedia.org/w/index.php?title=Courbe_de_Lebesgue&oldid=103768257 (Consulté le: 17 juillet 2017).

Zhang, C. et Huang, Y. (2010) « Querying Streaming Point Clusters As Regions », dans *Proceedings of the ACM SIGSPATIAL International Workshop on GeoStreaming*. New York, NY, USA: ACM (IWGS '10), p. 43–50.

Zheng, Y. *et al.* (2009) « Mining Interesting Locations and Travel Sequences from GPS Trajectories », dans *Proceedings of the 18th International Conference on World Wide Web*. New York, NY, USA: ACM (WWW '09), p. 791–800.

Zheng, Y. *et al.* (2011) « Recommending Friends and Locations Based on Individual Location History », *ACM Trans. Web*, 5(1), p. 5:1–5:44.

Zheng, Y. et Xie, X. (2011) « Learning Travel Recommendations from User-generated GPS Traces », *ACM Trans. Intell. Syst. Technol.*, 2(1), p. 2:1–2:29.