

# Analyse de données à grande échelle : application au dépôt RIPE Atlas

Mémoire réalisé par Hayat BELLAFKIH  
pour l'obtention du diplôme de Master en Sciences Informatiques

Année académique 2017 – 2018

**Directeurs :** B. Quoitin  
M. Goeminne

**Service :** Département de l'Informatique

**Rapporteurs :** D. Hauweele



# Résumé

RIPE Atlas est un projet créé et géré par l'organisme RIPE NCC, ce projet a donné naissance à de simples dispositifs, appelés sondes, qu'on connecte à un routeur. Ces sondes consomment une quantité légère d'électricité et de bande passante, mais elles sont capables de changer complètement des implantations physiques dans certaines infrastructures. La répartition abondante des sondes Atlas engendre quotidiennement une quantité importante de données qui dépasse la capacité des outils traditionnels à stocker et à traiter ces dernières avec efficacité. On parle des données massives.

Certaines problématiques dans le domaine des réseaux informatiques nécessitent une exploration plus profonde des données réseaux afin d'aboutir à des résultats importants, voire d'en tirer les connaissances. Dans ce travail, des technologies adaptées aux données massives ont été évaluées pour étudier un des problèmes liés aux performances des réseaux informatiques. C'est une évaluation d'un outil existant qui utilise un nombre très important de traceroutes.

**Mots clés :** RIPE Atlas, Traceroute, Big Data, données massives, MongoDB, Amazon Web Service, Apache Spark.

# Table des matières

<b>I</b>	<b>RIPE Atlas</b>	<b>6</b>
I.1	Introduction	6
I.2	RIPE NCC	6
I.3	Présentation du projet Atlas	6
I.3.1	Les mesures actives et passives de l'Internet	7
I.3.2	Généralités sur les sondes	7
I.3.3	Les générations des sondes	8
I.3.4	Les versions du firmware des sondes	9
I.3.5	Connexion des sondes à Internet	9
I.3.6	Architecture du système Atlas	10
I.4	Les mesures des sondes	13
I.4.1	Les mesures intégrées : Built-in	13
I.4.2	Les crédits d'Atlas	13
I.4.3	Les mesures personnalisées : User Defined measurement	15
I.4.4	La sélection des sondes	15
I.4.5	Les ancres VS sondes	16
I.4.6	Les sources de données Atlas	16
I.5	Limitations du projet Atlas	17
I.5.1	Les sondes et la vie privée	17
I.5.2	La sécurité dans Atlas	18
I.5.3	Les limitations d'Atlas	18
I.5.4	Confiance aux données Atlas	19
I.6	Projets existants de mesures d'Internet	19
I.6.1	Test Traffic Measurement Service	19
I.6.2	ProbeAPI	20
I.6.3	Archipelago	21
I.6.4	DIMES	21
I.6.5	SamKnows	21
I.7	Quelques cas d'utilisation des données collectées par les sondes	21
I.7.1	Détection des coupures d'Internet	22
I.7.2	Aide à la prise de décision	22
I.7.3	Le suivi des censures	22
I.7.4	Le suivi des performances d'un réseau	23
I.7.5	Le suivi des détours dans un trafic local	26
I.7.6	Visualisation : indicateurs et dashboard	26
I.8	Conclusion	26
<b>II</b>	<b>Détection des anomalies dans les délais d'un lien</b>	<b>27</b>
II.1	Présentation générale	27

II.2	Pourquoi analyser les délais des liens ?	27
II.3	Etude des délais des liens	28
II.3.1	Les données utilisées dans l'analyse des délais	28
II.3.2	Le principe de détection des changements des délais	29
II.4	L'étude des délais des liens en pratique : l'évolution du RTT différentiel des liens	31
II.4.1	Les étapes principales de détection	31
II.4.2	Description des paramètres de l'analyse des délais	31
II.4.3	Processus de détection des anomalies en deux phases	33
II.4.4	Exemple illustratif des étapes de la détections des alarmes	38
<b>III</b>	<b>Introduction au Big Data</b>	<b>44</b>
III.1	Introduction	44
III.2	Processus d'analyse de données massives	44
III.3	Quelques concepts associés au Big Data	45
III.3.1	Définition du Big Data : Volume, Vitesse, Variété et Véracité	45
III.3.2	L'architecture standard du Big Data	46
III.3.3	Les bases de données NoSQL (Not Only SQL)	48
III.3.4	Extraction, Transformation, Loading (ETL)	51
III.3.5	Schema on Write VS Schema on Read	53
III.3.6	L'informatique distribuée et l'analyse de données massives	54
III.4	Parcours de quelques technologies du Big Data	55
III.4.1	MongoDB	55
III.4.2	Amazon DynamoDB	55
III.4.3	Amazon S3, Amazon Glue et Amazon Athena	56
III.4.4	Hadoop ecosystem	58
III.4.5	Apache Spark	58
III.5	Conclusion	62
<b>IV</b>	<b>Implémentation en Spark/Scala</b>	<b>63</b>
IV.1	Introduction	63
IV.2	Implémentation	63
IV.2.1	Description de l'environnement	63
IV.2.2	Brève présentation de l'implémentation	63
IV.2.3	Création d'une application Spark/Scala	64
IV.2.4	Exécution d'une application Spark	79
<b>V</b>	<b>Application de quelques technologies Big Data sur l'analyse des traceroutes</b>	<b>82</b>
V.1	Introduction	82
V.2	Critères d'évaluation des technologies Big Data	82
V.3	Caractéristiques de l'environnement de test	83
V.4	Application 1 : MongoDB	83
V.5	Application 2 : Amazon DynamoDB	84
V.6	Application 3 : Amazon S3, Amazon Glue et Amazon Athena	85
V.7	Application 4 : Spark Apache avec Scala	90
V.7.1	Mode Standalone	91
V.7.2	EMR Cluster	94
V.8	Récapitulatif	95
V.9	Conclusion	97

<b>A</b>	<b>Amazon Athena</b>	<b>100</b>
A.1	Création de la table traceroutes . . . . .	100
A.2	Partitionnement de données sur Amazon Athena . . . . .	101
A.2.1	Présentation du partitionnement . . . . .	101
A.2.2	Application du partitionnement sur les traceroutes Atlas . . . . .	101
A.2.3	L'interrogation de données sur Amazon Athena . . . . .	102
<b>B</b>	<b>Important à savoir</b>	<b>104</b>
B.1	Pourquoi à partir des temps RTTs bruts on arrive à détecter les anomalies ? . . .	104
B.2	Rappel . . . . .	104
B.3	Processus de détection des anomalies . . . . .	104
B.3.1	Les phases de la détection . . . . .	104
B.3.2	Exemple illustratif . . . . .	110
B.4	Implémentation Spark/Scala . . . . .	115
<b>C</b>	<b>Exemple d'une réponse traceroute</b>	<b>117</b>

# Introduction

L'analyse de données, en particulier des données à grande échelle, attire de plus en plus les entreprises à s'y investir. Cette analyse peut affecter potentiellement la stratégie de ces entreprises. Les défis de l'analyse de données massives varient en fonction du processus suivi. Par exemple, les défis peuvent être liés à la définition des objectifs d'une analyse, le choix de données, la collecte de données, etc.

L'idée de ce travail est d'exploiter l'existence d'un dépôt de données en vue d'évaluer la mise en place ainsi que les performances de quelques technologies conçues pour la manipulation des données massives, appelées aussi Big Data. Les données considérées sont des données collectées par des dispositifs appelés sondes Atlas, et l'accès à ces données est publique. La manipulation de ces dernières nécessite l'utilisation des outils convenables dépassant les capacités des outils traditionnels ; l'exemple des bases de données relationnelles. Ainsi, le choix d'utilisation d'une technologie Big Data donnée dépend de plusieurs critères.

L'objectif du présent mémoire est de montrer la capacité des nouvelles technologies du Big Data à fournir des solutions efficaces capables d'assurer le stockage des données massives et d'effectuer des tâches de traitement sur de grandes quantités de données. Dans notre cas, ce sont des données collectées par les sondes Atlas. Ces données apportent des informations utiles et pertinentes de l'état des réseaux informatiques.

Dans un premier temps, nous avons étudié le projet RIPE Atlas (Réseaux IP Européens Atlas) afin de maîtriser le contexte général de données d'une part, et d'autre part de bien choisir les données de travail. Le choix a été fait sur les données générées par l'utilitaire Traceroute, en particulier, nous allons réutiliser un outil conçu dans le cadre d'un travail basé aussi sur les données collectées par les sondes Atlas. Cet outil permettant de détecter les anomalies dans les délais des liens dans les réseaux informatiques. Ensuite, nous avons passé en revue les technologies du Big Data disponibles afin de sélectionner celles qu'on puisse évaluer leurs performances et convenances pour les données choisies. A l'issue de cette étape, nous avons abordé l'évaluation des technologies Big Data sélectionnées pour réutiliser l'outil de la détection des anomalies.

Le présent document est organisé en quatre chapitres. Le premier chapitre présente le projet RIPE Atlas : la présentation des caractéristiques techniques et fonctionnelles des sondes Atlas ainsi qu'une liste non exhaustive des cas d'usage de ces sondes. Le deuxième chapitre reprend l'algorithme de la détection à évaluer par les technologies Big Data. Le troisième chapitre énumère quelques concepts liés au Big Data ainsi qu'une liste non exhaustive des technologies Big Data. Pour finir, le quatrième chapitre aborde l'application de l'outil de détection en pratique pour les technologies sélectionnées.

# Chapitre I

## RIPE Atlas

### I.1 Introduction

Le présent chapitre est une présentation détaillée du projet RIPE Atlas créé par l'organisme RIPE NCC (Réseaux IP Européens - Network Coordination Centre). RIPE Atlas a introduit l'utilisation des dispositifs pour effectuer des mesures des réseaux informatiques dans le monde. Ce chapitre présente dans un premier temps les caractéristiques des sondes Atlas, ensuite, il reprend une liste non exhaustive de quelques outils similaires aux sondes Atlas en matière d'objectifs, puis, il expose quelques limites du système RIPE Atlas. Enfin, ce chapitre liste brièvement quelques travaux basés sur le projet RIPE Atlas illustrant les cas d'usage du présent projet. Dans la suite de ce document, Atlas désigne le projet RIPE Atlas et une sonde désigne une sonde en provenance du projet Atlas.

### I.2 RIPE NCC

Le RIPE NCC est un organisme qui alloue les blocs d'adresses IP et des numéros des Systèmes Autonomes dans l'Europe et une partie de l'Asie, notamment au Moyen-Orient. Un *Système Autonome*, appelé AS, est un ensemble de réseaux et de routeurs sous la responsabilité d'une même autorité administrative. Chaque Système Autonome est identifié par un code sur 16 bits uniques. Les protocoles qui tournent au sein d'un Système Autonome peuvent être différents.

RIPE NCC fournit aussi différents services relatifs à la gestion des réseaux informatiques. Il maintient de multiples projets pour de nombreux protocoles comme DNS (Domain Name System) (DNSMON<sup>1</sup>), BGP (Routing Information Service ou RIS<sup>2</sup>) et d'autres projets et services. En particulier, nous sommes intéressés par le projet RIPE Atlas géré aussi par RIPE NCC. L'objectif du projet RIPE Atlas est de déployer des dispositifs dans le monde qui sont capables d'effectuer des mesures de performance des réseaux informatiques.

### I.3 Présentation du projet Atlas

RIPE NCC a créé le projet RIPE Atlas en 2010. L'objectif de ce projet est de créer et de distribuer des dispositifs matériels destinés à effectuer des mesures au sein d'un réseau informa-

---

1. Source : <https://atlas.ripe.net/dnsmon/>, consultée le 27/12/2018.

2. Source : <https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris>, consultée le 27/12/2018.



tique. Le nombre de sondes déployées est en augmentation continue notamment en Europe, et ce malgré que ces dernières sont déployées par des volontaires. Une carte géographique reprenant la distribution de ces sondes est disponible<sup>3</sup>. Actuellement, plus de 10.000 sondes Atlas sont actives, ces dernières produisent environ 450 millions de mesures par jour, ce qui correspond à 5, 000 résultats par seconde [25].

### I.3.1 Les mesures actives et passives de l'Internet

Il existe plusieurs approches pour analyser l'état d'un réseau informatique. Par exemple, la mesure du temps de réponse, le suivi du chemin des paquets, etc. Les deux approches les plus répandues sont : passive et active. L'approche passive fait référence au processus de mesure d'un réseau, sans créer ou modifier le trafic sur ce réseau. L'approche active repose sur l'injection des paquets sur le réseau et la surveillance de ce flux. Les données issues des deux approches permettent de proposer des améliorations de l'Internet. Le projet RIPE Atlas est un des outils s'inscrivant dans l'approche active. Ce sont des dispositifs, appelés sondes, hébergés par des volontaires, ils sont distribués et maintenus par RIPE NCC. Les données collectées par ces dispositifs sont disponibles au public sur le dépôt de RIPE Atlas [4].

### I.3.2 Généralités sur les sondes

- Les sondes mesurent les performances de la couche IP. Une sonde envoie des paquets réels et observe la réponse en temps réel indépendamment des applications en dessus de la couche IP.
- Les sondes ne sont pas des observatrices des données comme le trafic du routage BGP, ainsi, elles n'observent pas le trafic de leurs hébergeurs.
- Les sondes sont déployées dans différents emplacements autour du monde, cette répartition permet de diversifier les mesures et par conséquent, d'enrichir les études basées sur les données collectées<sup>4</sup>.
- Les sondes sont déployées volontairement dans une maison, un bureau, un entrepôt de données, etc.
- Les mesures peuvent être lancées à tout moment et pour n'importe quelle période<sup>5</sup>.
- La participation au projet Atlas est ouverte à toute personne qui s'y intéresse, cela inclut les résultats de mesures, les outils d'analyse, l'hébergement des sondes elles-mêmes, les travaux, etc.
- Le projet Atlas simule le comportement de la couche IP. Par exemple, avec ces sondes, il est possible de :
  - Suivre l'accessibilité d'une destination<sup>6</sup> depuis différents emplacements dans le monde et depuis différents réseaux informatiques. Car les sondes sont réparties dans plusieurs pays et déployées dans différents réseaux.
  - Étudier des problèmes du réseau remontés en effectuant des vérifications de connectivité ad-hoc via les mesures effectuées par les sondes.

---

3. Source : <https://atlas.ripe.net/results/maps/density/>, consultée le 02/05/2019.

4. Voir les sections I.4.1 et I.4.3 concernant les mesures des sondes.

5. Si le nombre de crédits (voir la section des crédits I.4.2) disponibles le permet et qu'il n'y a pas de dépassement du nombre de mesures autorisé.

6. Une destination représente une adresse IP.

- Tester la connectivité IPv6.
- Vérifier l'infrastructure DNS.

La section I.7 reprend quelques cas d'usage du projet Atlas.

### I.3.3 Les générations des sondes

Depuis leur création en 2010, les sondes ont connu trois générations du matériel : v1, v2 et v3. Le Tableau I.1 présente quelques caractérisations de ces trois générations et la Figure I.1 montre le matériel utilisé dans chaque génération.

	v1	v2	v3
<b>Matériel informatique</b>	Lantronix XPort Pro [8]	Lantronix XPort Pro [8]	tp-link tl-mr3020
<b>Début d'utilisation</b>	2010	2011	2013
<b>Mémoire RAM</b>	8 Mo	16 Mo	32 Mo
<b>Mémoire Flash</b>	16 Mo	16 Mo	4 Mo
<b>CPU</b>	32-bit	32-bit	32-bit
<b>Support du Wi-Fi</b>	Non	Non	oui
<b>Support du NAT</b>	oui	oui	oui
<b>Vitesses supportées</b>	10 Mbit/s et 100 Mbit/s	10 Mbit/s et 100 Mbit/s	10 Mbit/s et 100 Mbit/s

TABLE I.1 – Les caractéristiques du matériel des trois générations des sondes



Génération 1

Génération 2

Génération 3

FIGURE I.1 – Les trois générations des sondes

Source : <https://atlas.ripe.net/docs/>, consultée le 05/08/2018.

Pour précision, les générations 1 et 2 présentent une très faible consommation d'énergie, cependant, elles ont un temps de redémarrage et coûts de production élevés.

En 2015, plusieurs utilisateurs des sondes ont montré un intérêt aux sondes virtuelles. Les sondes virtuelles présentent des avantages et aussi des inconvénients. Parmi les avantages, la conception des sondes virtuelles permet d'explorer des emplacements qui sont difficilement accessibles. En effet, cela permet d'étendre le réseau des sondes. D'autre part, les sondes virtuelles peuvent être installées sans contraintes physiques ou organisationnelles. Parmi les inconvénients, une complexité sera ajoutée au système Atlas, plus de ressources seront demandées. De plus, il peut y avoir le problème de la qualité de données ; le manque de données peut faire référence à une perte de paquets ou bien la machine qui héberge la sonde n'est plus disponible pour continuer les mesures.

### I.3.4 Les versions du firmware des sondes

En principe, toutes les sondes Atlas collectent la même information, indépendamment de leur version du firmware. On trouve les mêmes attributs<sup>7</sup> dans toutes les versions sauf de légers changements : ajout d'un ou de plusieurs attributs, la modification des noms des attributs, etc. Pour la simplification, nous donnons un identifiant entier pour chaque version (celui entre les parenthèses dans la liste ci-dessous). Cet identifiant sera utilisé dans la suite de ce document.

Il existe plusieurs versions du firmware :

- version 1 est identifiée par 1 **(1)**;
- version 4400 est identifiée par une valeur entre 4400 et 4459 **(2)**;
- version 4460 est identifiée par une valeur entre 4460 et 4539 **(3)**;
- version 4540 est identifiée par une valeur entre 4540 et 4569 **(4)**;
- version 4570 est identifiée par une valeur entre 4570 et 4609 **(5)**;
- la dernière version du firmware<sup>8</sup> est 4610 **(6)**.

### I.3.5 Connexion des sondes à Internet

Les générations 1 et 2 des sondes ont une interface Ethernet (RJ-45). La génération 3 dispose techniquement des capacités Wi-Fi. Cependant, ces sondes ne sont pas suffisamment prêtes au niveau logiciel pour supporter le Wi-Fi. Une fois la sonde se connecte au port d'Ethernet, elle acquiert une adresse IPv4, un résolveur DNS en utilisant DHCP et la configuration IPv6 via *Router Advertisement*. Ensuite, elle essaie de rejoindre l'infrastructure du RIPE Atlas. Pour ce faire, elle utilise le résolveur DNS et se connecte à l'infrastructure à travers SSH sur le port TCP de sortie 443 comme il est illustré dans la Figure I.2. L'architecture du système RIPE est détaillée dans la section I.3.6.

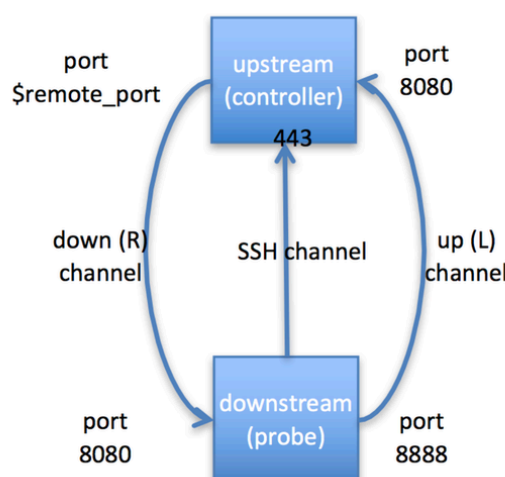


FIGURE I.2 – La connexion des sondes à l'infrastructure RIPE Atlas [24]

7. Attribut dans le sens du JSON : chaque résultat de mesure est enregistré comme étant un objet JSON.

8. A la date de consultation 25/01/2018.

### I.3.6 Architecture du système Atlas

Il existe deux catégories d'outils de surveillance des réseaux informatiques : des outils matériels et d'autres logiciels. Les sondes sont parmi les outils matériels. Le choix d'utilisation d'un outil matériel au lieu d'un outil logiciel dépend de plusieurs facteurs, par exemple l'indépendance du système d'exploitation, la facilité de déploiement, la disponibilité des sondes tout le temps (au lieu d'être dépendante de la machine qui l'héberge) et d'autres facteurs liés à la sécurité.

Le système Atlas est conçu pour qu'il soit opérationnel de façon distribuée. La plupart des composantes ont assez de connaissances pour remplir leurs rôles, sans nécessairement avoir besoin de connaître les états des autres composantes du système. Cela garantit que le système soit capable d'assurer la plupart des fonctionnalités en cas d'un problème temporaire. Par exemple, si une sonde est déconnectée de l'infrastructure, elle continue les mesures planifiées et les données sont renvoyées dès sa reconnexion au système. La Figure I.3 montre une vue d'ensemble de l'architecture d'Atlas.

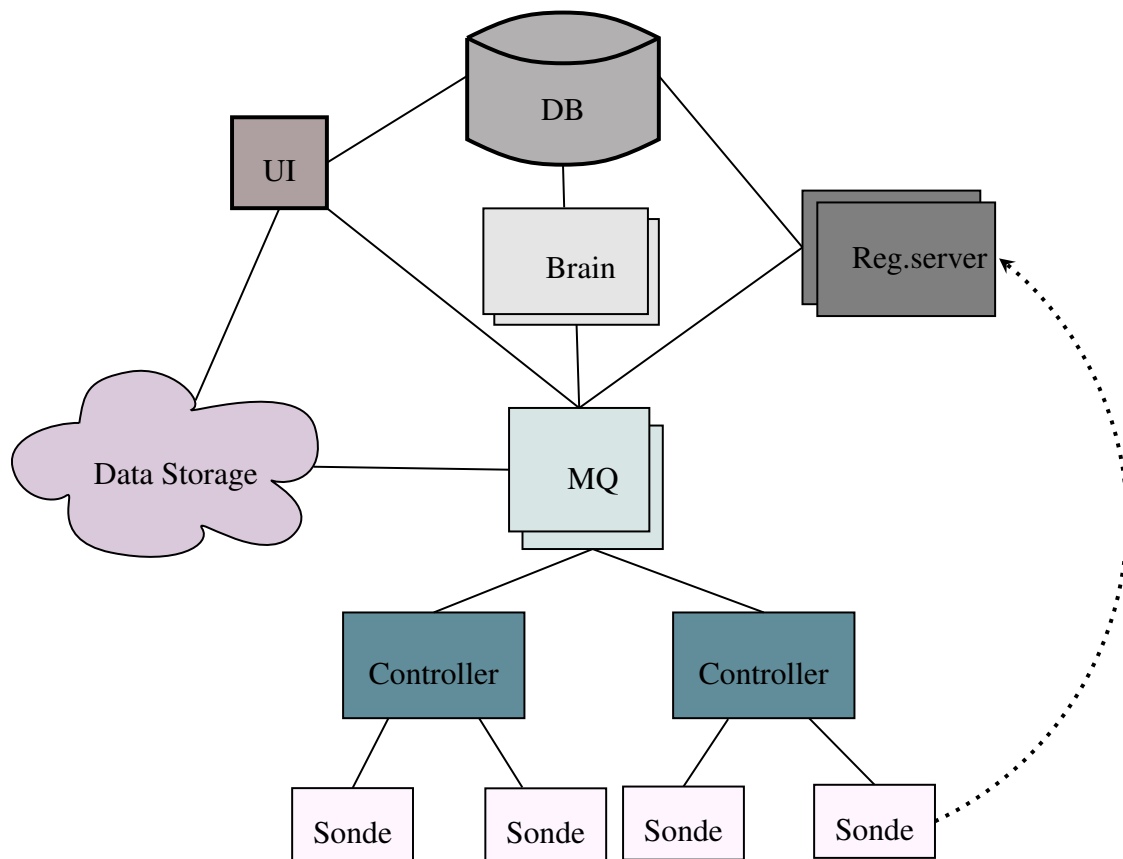


FIGURE I.3 – L'architecture du système Atlas

Source : Schéma repris du travail de Kisteleki [25]

L'architecture du système Atlas est constituée par les composantes suivantes :

**Registration server** (Reg.server) : c'est le seul point d'entrée de confiance pour les sondes. Son rôle est de recevoir toutes les sondes désirant se connecter au système Atlas. Ensuite, il redirige chaque sonde vers le contrôleur adéquat ; celui le plus proche de la sonde et qui est suffisamment non occupé. Le serveur d'enregistrement a un aperçu de haut niveau du système.

**Controller** : un contrôleur accepte d'établir une connexion avec une sonde parmi celles dont il a reçu leurs clés du serveur d'enregistrement (Reg.server). Une fois la connexion est établie entre une sonde et un contrôleur, ce dernier garde cette connexion active pour prévenir la sonde des mesures à effectuer et recevoir les résultats de ces mesures. Le rôle du contrôleur est de communiquer avec les sondes, associer les mesures aux sondes en se basant sur leur disponibilité et sur autres critères, et enfin, collecter les résultats intermédiaires des mesures.

**Message Queue (MQ)** : tout d'abord définissons MQ :

*« Message Queue ou file d'attente de message : est une technique de programmation utilisée pour la communication interprocessus ou la communication de serveur-à-serveur. Les files d'attente de message permettent le fonctionnement des liaisons asynchrones normalisées entre deux serveurs, c'est-à-dire de canaux de communications tels que l'expéditeur et le récepteur du message ne sont pas contraints de s'attendre l'un l'autre, mais poursuivent chacun l'exécution de leurs tâches<sup>a</sup>. »*

<sup>a</sup>. Source : [https://fr.wikipedia.org/wiki/File\\_d'attente\\_de\\_message](https://fr.wikipedia.org/wiki/File_d'attente_de_message), consultée le 05/08/2018.

Un cluster de serveurs MQ agit comme un système nerveux central au sein de l'architecture d'Atlas. Il gère la connectivité entre les composantes de l'infrastructure et il garantit l'échange de messages avec un délai minimal. C'est cette composante qui élimine le besoin que les autres composantes de l'infrastructure soient au courant des états des autres composantes de l'infrastructure. En plus, chaque composante peut être ajoutée ou retirée sans devoir synchroniser cette information avec l'infrastructure entière. Si c'est le cas d'une déconnexion d'une composante, les messages seront sauvegardés sur différents niveaux jusqu'au moment de la reconnexion.

**UI (User Interface)** : elle s'occupe des interactions de l'utilisateur. Elle sert les pages pour l'interface graphique de mesures [3], elle traite les appels en provenance de l'API<sup>9</sup> et sert les demandes de téléchargement en provenance de l'API.

**Brain** : il effectue des tâches de haut niveau dans le système, notamment la planification des mesures. Cette planification est basée sur les demandes reçues via l'interface graphique Web de mesures (UI) ou bien via l'API. La planification passe par la présélection des sondes et la négociation avec les contrôleurs pour voir la disponibilité des sondes.

**DB** : c'est une base de données SQL contenant toutes les informations du système Atlas : les informations sur les sondes et leurs propriétés, les meta-data des mesures, les utilisateurs, les crédits, etc.

**Data Storage** : c'est un cluster Hadoop/HBase pour le stockage à long terme de tous les résultats des mesures. Cette technologie permet aussi d'effectuer des calculs d'agrégation périodiques et d'autres tâches.

---

9. Source : <https://atlas.ripe.net/docs/api/v2/manual/>, consultée le 05/08/2018.

**Hadoop MapReduce** est un modèle de programmation qui permet de traiter les données massives suivant une architecture distribuée dans un cluster.

**HBase** est une base de données non relationnelle et distribuée. Elle est adaptée au stockage de données massives.

La Figure I.4 illustre les étapes d'établissement de la connexion entre une sonde et l'infrastructure Atlas.

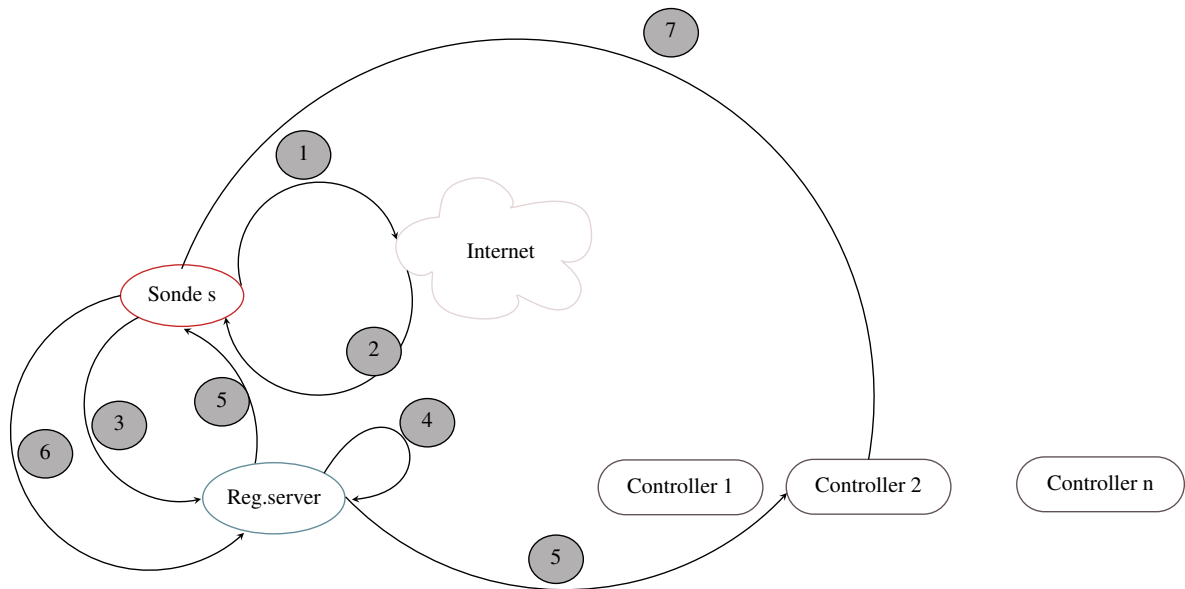


FIGURE I.4 – Les étapes d'établissement d'une connexion entre une sonde et l'architecture Atlas

Les étapes suivantes décrivent le déroulement de la connexion d'une sonde *s* à l'infrastructure d'Atlas.

- ① La sonde se connecte à Internet via le câble Ethernet RJ45.
- ② La sonde acquiert différentes informations : une adresse IPv4, une adresse IPv6 via Router Advertisement et les informations du résolveur DNS via DHCP.
- ③ Les informations précédemment acquises permettent à la sonde de se connecter au serveur d'enregistrement (Reg.server). C'est la première entrée vers l'infrastructure.
- ④ En se basant sur la géolocalisation de la sonde, la charge des différents contrôleurs et d'autres options, le serveur d'enregistrement décide le contrôleur qui va être associé à la sonde.
- ⑤ Suite à la décision du serveur d'enregistrement, le contrôleur reçoit l'identifiant de la sonde à gérer et la sonde reçoit l'identifiant du contrôleur à qui elle sera associée.
- ⑥ Une fois l'association entre la sonde et le contrôleur est faite, la sonde se déconnecte du serveur d'enregistrement.
- ⑦ La connexion entre la sonde et le contrôleur est maintenue le plus longtemps possible. Les contrôleurs gardent le contact avec les autres composants via Message Queue. Dans le cas où une des composantes se déconnecte du système, les événements sont conservés jusqu'au moment où la connexion est restaurée.

La connexion précédemment établie permet aux sondes d'envoyer leurs rapports de mesures aux serveurs de stockage. C'est la même connexion qui permet de passer les commandes aux sondes pour qu'elles puissent effectuer les mesures et les mises à jour de leur firmware.

## I.4 Les mesures des sondes

Les sondes ont la capacité d'effectuer plusieurs requêtes réseaux. On distingue deux catégories de mesures. La première catégorie reprend les mesures à effectuer, par défaut, par toutes les sondes, appelées mesures intégrées. Tandis que la deuxième catégorie, concerne les mesures personnalisées suivant des besoins spécifiques. Ces dernières nécessitent la possession des crédits.

### I.4.1 Les mesures intégrées : Built-in

Une fois une sonde connectée, elle lance automatiquement un ensemble de mesures prédéfinies, appelées *Built-in Measurements*. Les mesures peuvent être effectuées selon l'adressage IPv4 ou bien IPv6. Le choix du mode IPv4, IPv6 ou les deux, dépend de la capacité du réseau qui héberge la sonde. Il existe deux types de mesures : celles qui s'exécutent une seule fois, appelée *One-Off*, et celles qui s'exécutent périodiquement, à chaque intervalle fixe de temps.

Par défaut, les sondes assurent les mesures intégrées suivantes :

- Les informations sur la configuration du réseau dans lequel la sonde est déployée.
- L'historique de la disponibilité de la sonde.
- Les mesures du RTT (Round Trip Time) via une requête traceroute.
- Les mesures ping vers un nombre de destinations prédéfinies.
- Les mesures traceroute vers un nombre de destinations prédéfinies.
- Les requêtes vers les instances des serveurs DNS racines.
- Les requêtes SSL/TLS (Secure Socket Layer/Transport Layer Security) vers un nombre de destinations prédéfinies.
- Les requêtes NTP (Network Time Protocol).

Chaque mesure a un identifiant ID unique, cet identifiant indique le type de la mesure, s'il s'agit du ping, traceroute ou autres. Plus de détails sur la signification des identifiants des mesures sont disponibles sur RIPE Atlas <sup>10</sup>.

### I.4.2 Les crédits d'Atlas

Les crédits d'Atlas est une sorte de reconnaissance de la contribution des participants à ce projet. Un hébergeur d'une sonde reçoit un nombre de crédits en contrepartie de la durée pendant laquelle sa sonde reste connectée. D'autre part, il gagne d'autres crédits suivant les résultats de mesures générés par cette sonde. Les crédits gagnés peuvent être utilisés dans la création des mesures personnalisées, appelées *User Defined Measurements* (voir la section I.4.3). Les personnes ayant gagné des crédits peuvent les transférer vers une autre personne dont il a besoin. Les crédits peuvent être obtenus via :

---

10. Source : <https://atlas.ripe.net/docs/built-in/>, consultée le 10/08/2018.



- L'hébergement d'une sonde ; à chaque utilisation d'une sonde, son hébergeur reçoit un nombre de crédits. La connexion d'une sonde au système durant une minute apporte 15 crédits.
- L'hébergement d'une ancre <sup>11</sup>.
- La recommandation à une personne d'héberger une sonde.
- En étant un sponsor du RIPE NCC. Le parrainage des sondes est possible pour les organisations et les individus. Le sponsor reçoit le même nombre de crédits que les hébergeurs de ces sondes.
- En étant un registre Internet local (Local Internet Registry).
- La réception des crédits d'une autre personne via un transfert de crédits.

Le lancement des mesures personnalisées exploite les ressources de l'infrastructure Atlas d'une part, du réseau hébergeur de la sonde d'autre part. Par conséquent, les mesures sont organisées afin d'éviter toute surcharge du système.

Le coût d'une mesure dépend de son type et des options spécifiées. Le système calcule le nombre de crédits nécessaires pour effectuer une mesure donnée. Le nombre de crédits est déduit à chaque résultat reçu. Ci-dessous le coût unitaire des différents types de mesures.

#### **Ping et ping6 :**

$$\text{Coût unitaire} = N \times (\lfloor \frac{S}{1500} \rfloor + 1)$$

Où  $N$  est le nombre de paquets dans le train (par défaut 3) et  $S$  est la taille du paquet (par défaut : 48 octets).

#### **DNS et DNS6 :**

Coût unitaire pour UDP : 10 crédits/résultat  
Coût unitaire pour TCP : 20 crédits/résultat

#### **Traceroute et traceroute6 :**

$$\text{Coût unitaire} = 10 \times N \times (\lfloor \frac{S}{1500} \rfloor + 1)$$

Où  $N$  est le nombre de paquets dans le train (par défaut 3) et  $S$  est la taille du paquet (par défaut : 40 octets).

#### **SSLCert et SSLCert6 :**

Coût unitaire = 10 crédits/résultat.

**Exemple :** la planification d'une mesure ayant les caractéristiques suivantes nécessite 14,400 crédits.

---

11. Les ancres Atlas sont décrites dans la section [I.4.5](#).



La fréquence	: deux fois par heure
La durée	: deux jours (48 heures)
Le nombre de sondes	: 5
Type de mesure	: <i>traceroute</i>

Tel que :

$$5 \times 2 \text{ mesures/heure} \times 48 = 480 \text{ ligne résultat}$$

$$30 \text{ credits/result} \times 480 \text{ results} = 14.400 \text{ crédits}$$

### I.4.3 Les mesures personnalisées : User Defined measurement

Les sondes peuvent effectuer des mesures personnalisées qui peuvent être lancées via l'interface Web [3] ou bien via HTTP REST API. Ce sont les mêmes types de mesures que ceux des mesures intégrées. A savoir, ping, traceroute, HTTP Get, SSLCert, DNS, NTP et TLS. Il faut avoir assez de crédits pour lancer ces mesures. L'interface web dédiée à la création d'une nouvelle mesure offre toute les possibilités comme la personnalisation des éléments suivants :

- le type de la mesure ;
- la sélection des sondes réalisant la mesure ;
- la fréquence de la mesure et sa durée.

Chaque mesure est suivie via son état dont on distingue plusieurs : *specified, scheduled, ongoing, stopped, Forced to stop, no suite probes* et enfin *failed*.

### I.4.4 La sélection des sondes

La sélection des sondes pour effectuer une des mesures repose sur un des critères suivants :

- numéro d'AS ;
- zone géographique via l'attitude et la longitude ;
- pays (ou zone géographique comme Europe) ;
- préfixe IP ;
- manuellement, avec les identifiants des sondes ;
- reprendre ceux d'une mesure précédente.

Il existe une autre manière de grouper les sondes, il s'agit d'un groupement par étiquettes. Les étiquettes servent comme indicateur des propriétés, des capacités, de la topologie du réseau et d'autres caractéristiques des sondes. On distingue les étiquettes système et utilisateur.

Les étiquettes utilisateurs sont associées à une sonde librement par son hébergeur. Or, les étiquettes système sont attribuées uniquement par l'équipe Atlas et sont mises à jour périodiquement, à priori chaque 4 heures. Des exemples d'étiquettes système sont disponibles sur RIPE Atlas<sup>12</sup>.

12. Source : <https://atlas.ripe.net/docs/probe-tags/>, consultée le 23/01/2018.

### I.4.5 Les ancrs VS sondes

Les ancrs Atlas, auxquelles je référerai dans la suite du document seulement par le terme «ancres», sont des dispositifs agissant comme cibles aux différentes mesures lancées par les sondes. Il est possible de planifier des mesures entre les ancrs, ces mesures permettent de vérifier l'état des réseaux qui hébergent ces ancrs. De plus, les ancrs peuvent être considérées comme cibles aux mesures suivantes :

- Ping.
- Traceroute.
- DNS : les ancrs ont été configurées avec BIND pour qu'elles agissent en tant que serveur DNS faisant autorité.
- HTTP et HTTPS : l'ancre fait tourner un serveur Web, ce dernier utilise un gestionnaire personnalisé de réponses aux requêtes HTTP(S) ayant comme seule option la taille du payload. Cette taille peut prendre une valeur maximale de 4096 et la réponse est fournie sous format JSON. L'exemple d'une requête HTTP avec une taille de 536 depuis une sonde vers une ancre est :

```
http://nl-ams-as3333.anchors.atlas.ripe.net/536
```

Les ancrs sont configurées avec un certificat SSL auto-signé en utilisant une clé de 2048 bit et un temps d'expiration de 100 ans. Le Tableau I.2 reprend une comparaison de certaines caractéristiques communes entre les sondes et les ancrs.

	Sonde	Ancre
<b>Mesures originaires de</b>	oui	oui
<b>Mesures à destination de</b>	— <sup>13</sup>	ping, traceroute, DNS, HTTP(S).
<b>Nomination</b>	—	structurée <sup>14</sup>
<b>Crédit gagnés</b>	$N$	$10 * N$
<b>Besoin en bande passante</b>	léger	important
<b>Coût : gratuite</b>	oui	non <sup>15</sup>

TABLE I.2 – Comparaison entre les sondes et les ancrs

### I.4.6 Les sources de données Atlas

Le projet Atlas donne accès à trois catégories de données : les détails de connexion des sondes, la description des mesures effectuées par les sondes (meta-data) et enfin les résultats des mesures intégrées et personnalisées. La première catégorie comprend les données descriptives des sondes, ce que inclut les informations de la connexions, de la déconnexion, des réseaux et autres. Les détails des connexions des sondes ne sont disponibles qu'à partir du 13 mars 2014

13. — : Non disponible.

14. Exemple de *de-mai-as2857.anchors.atlas.ripe.net* avec la structure suivante : *pays-ville-ASN.anchors.atlas.ripe.net*.

15. Le matériel est au frais de l'hébergeur.

jusqu'à ce jour<sup>16</sup>, ces informations sont organisées dans un fichier JSON par jour. Les données de certains jours sont manquantes. Toutes les archives se trouvent dans [5] et la taille d'une seule archive est entre 120 KB et 921 KB<sup>17</sup>.

La deuxième catégorie concerne les méta-datas décrivant les mesures, ces données sont sauvegardées chaque semaine dans un fichier JSON, où une ligne, est objet JSON, décrit une mesure publique. Au moment de la consultation, la taille de chaque archive était entre 124 Mo et 1.5 Go. On a l'accès à ces données de deux manières : à travers le téléchargement direct depuis un serveur FTP ou bien via *streaming API*. Les noms des archives sont bien structurés afin de faciliter l'automatisation des téléchargements.

Enfin, dans la troisième catégorie on trouve les résultats des mesures. Seules les données des derniers 30 jours qui sont conservées en archives et disponibles sur la page Web dédiée<sup>18</sup>. Les autres périodes sont accessibles en utilisant l'API d'Atlas. Les fichiers ont été nommés comme suit<sup>19</sup> :

`$TYPE-$IPV-$SUBTYPE-$DATE.bz2`

- \$TYPE peut être traceroute, ping, dns, ntp, http, sslcert.
- \$IPV version du protocole IP v4 ou v6.
- \$DATE date au format YEAR-MONTH-DAY. (exemple : 2017-06-13)
- \$SUBTYPE type de mesure builtin ou udm.

En considérant toutes les possibilités des types, la quantité de données générées quotidiennement est environ 25 Go<sup>20</sup> et la taille des archives est entre 281M et 3.2G.

Depuis 15 mars 2018, les résultats des mesures sont regroupés différemment. 24 archives par jour, une seule archive pour chaque heure et type de mesure. L'archive ne distingue pas entre mesures IPv4 et IPv6, entre mesures intégrées et personnalisées. Il existe un attribut "**af**" qui distingue entre IPv4 et IPv6 et l'identifiant de la mesure pour distinguer les mesures intégrées et celles personnalisées. Par exemple, les mesures personnalisées ont un identifiant > 1,000,000.

Streaming API permet de récupérer les résultats de mesures en temps réel en provenance des sondes publiques. Ainsi, elle fournit continuellement de nouveaux résultats en temps réel, et ce via une connexion de type HTTPS web-socket active tout le temps.

## I.5 Limitations du projet Atlas

### I.5.1 Les sondes et la vie privée

La sonde n'a pas l'accès au trafic de son hébergeur. Elle maintient sa connexion avec l'infrastructure centrale et elle exécute les mesures planifiées vers les destinations publiques sur Internet.

Les sondes peuvent révéler l'adresse IP de leur hébergeur. Bien que, les informations personnelles telles que les adresses MAC et les adresses e-mail ne seront jamais affichées. Cependant, l'adresse IPv6 peut exposer l'adresse MAC.

---

16. 15/08/2018.

17. A la date de consultation.

18. Source : <https://data-store.ripe.net/datasets/atlas-daily-dumps/>, consultée le 05/04/2018.

19. A partir du 15 mars 2018.

20. Source : <https://ftp.ripe.net/ripe/atlas/data/README>, consultée le 26/03/2018.

## I.5.2 La sécurité dans Atlas

La connexion entre les composantes de l'infrastructure Atlas est maintenue le plus longtemps possible comme c'est décrit dans la section I.3.6. De ce fait, la sécurité des différentes connexions est primordiale. Afin de réduire la surface d'attaque contre les sondes, les précautions suivantes sont prises :

- Les hébergeurs des sondes ne disposent d'aucun service qui leur permet de se connecter aux sondes (dans le sens de TCP/IP).
- Les sondes n'échangent aucune clé d'authentification entre elles. En effet, chaque sonde dispose de sa clé qu'elle utilise pour se connecter à l'infrastructure.
- Comme les sondes sont chez les hébergeurs, il est impossible qu'elles soient résilientes au démontage. Cependant, si c'est le cas, cela ne devrait pas affecter les autres sondes.
- Toutes les communications au sein de l'infrastructure Atlas se font d'une manière sécurisée. Les connexions entre les composantes sont maintenues grâce aux *secure channels* avec *mutual authentication*.
- Le logiciel qui tourne dans les sondes peut être facilement mis à niveau ; la sonde est capable de vérifier l'authenticité d'une nouvelle version du firmware et cela via les signatures cryptographiques.

Le système Atlas est un système comme les autres, il n'est pas résilient à 100 % aux attaques. Cependant, l'équipe Atlas propose régulièrement des améliorations et des fixations de bugs surmontées par la communauté Atlas.

## I.5.3 Les limitations d'Atlas

De nombreux travaux ayant exploité les données générées par les sondes. Néanmoins, ce système connaît des bugs et des limitations. Les membres de la communauté Atlas s'engagent à remonter les bogues liés aux sondes, ces derniers sont répertoriés dans une rubrique dédiée [6].

Le projet Atlas connaît des limitations liées à la visualisation. Actuellement, Atlas supporte la visualisation des mesures de type ping ayant utilisé au maximum 20 sondes. Cette limitation concerne aussi le type traceroute, toutefois, il est possible de visualiser seulement les mesures IPv6 built-in.

Afin d'éviter la surcharge des sondes et de l'infrastructure, l'équipe Atlas a limité le nombre de mesures périodiques de 10 à la fois et de 10 mesures de type one-off vers n'importe quelle cible à un moment donné. De plus, il n'est pas possible d'utiliser plus de 500 sondes par mesure.

Pour les mesures one-off (non périodiques), une sonde peut effectuer au plus 10 mesures en parallèle. L'équipe Atlas limite aussi la fréquence des mesures personnalisées. Un hébergeur d'une sonde peut effectuer :

- Ping chaque 60 secondes (par défaut 240 secondes) ;
- Traceroute chaque 60 secondes (par défaut 900 secondes) ;
- SSL chaque 60 secondes (par défaut 900 secondes) ;
- DNS chaque 60 secondes (par défaut 240 secondes).

Dans le cas d'une déconnexion, la sonde continue à effectuer les mesures. En ce qui concerne les versions 1 et 2, la sonde est capable de sauvegarder les 6 dernières heures de données. Tandis qu'avec la version 3, une sonde est capable de sauvegarder les résultats de plusieurs mois. Une fois la sonde est connectée, elle envoie les données à l'infrastructure centrale. Concernant la consommation des crédits par jour, l'équipe Atlas limite cette consommation à 1.000.000 crédits.

### I.5.4 Confiance aux données Atlas

Etant donné le nombre important des travaux basés sur les données issues du projet Atlas, peut-on faire confiance à la qualité de ces données ? sont-elles complètes ?

La question de la complétude des données est plus présente pour les mesures périodiques. W. Shao et al. [35] ont traité les résultats des mesures manquantes. Leur approche repose sur l'étude de la corrélation entre l'absence de certains résultats de mesures et les périodes durant lesquelles les sondes ont été déconnectées. Les informations de connexion/déconnexion des sondes font partie des données sauvegardées par Atlas. Parmi les travaux ayant étudié la qualité des données collectées par les sondes, on trouve le travail de W. Shao et al. [35], ces derniers ont étudié les mesures en provenance des sondes v3, effectuées entre le 01/06/2016 et le 01/07/2016 (UTC). Les auteurs ont combiné les informations relatives à la connexion/déconnexion des sondes et leurs mesures planifiées en se basant sur l'attribut *timestamp*; ce dernier est présent dans chaque résultat de mesure et dans les états de connexions.

Malgré les règles mises en place par l'équipe Atlas en terme de mesures autorisées par jour, cela n'empêche pas la situation où un nombre important de mesures doit être effectué au même moment. En outre, plusieurs utilisateurs peuvent s'intéresser à une même sonde. C'est l'objet du travail [22] de T. Holterbach et al. ; si les mesures lancées par les autres utilisateurs affectent les résultats obtenus par un autre utilisateur, et si c'est le cas, comment s'y entreprendre. Les expériences réalisées ont montré la présence de l'interférence entre les mesures à destination des sondes et cela de deux manières. Premièrement, les mesures depuis et à destination des sondes augmentent le temps reporté par la sonde et ils ont conclu que l'amélioration du CPU a permis de limiter les interférences sur le temps mesuré par les sondes. Deuxièmement, ils ont conclu que les mesures perdent la synchronisation avec l'infrastructure d'Atlas, pendant plus d'une heure, à cause de la charge concurrentielle que subit le système Atlas. Dans ce cas, l'amélioration du matériel ne peut pas résoudre le problème.

## I.6 Projets existants de mesures d'Internet

Dans les sections précédentes, nous avons présenté le projet Atlas comme étant une plateforme pour la collecte de données des réseaux informatiques. Toutefois, il existe d'autres projets similaires à Atlas. Les sections suivantes reprennent une liste non exhaustive des projets similaires à Atlas.

### I.6.1 Test Traffic Measurement Service

Avant l'arrivée d'Atlas, Le RIPE NCC a assuré le suivi de la connectivité entre les réseaux informatiques via d'autres plateformes, comme la plateforme Test Traffic Measurement Service (TTM). C'est un projet qui permet de mesurer la connectivité entre un nœud source et un nœud destination sur Internet. C'était une des manières pour suivre la connectivité entre le réseau source et le réseau destination.

L'idée était la mise en place d'un dispositif, test-box, qui génère du trafic. Ce dernier n'affecte pas l'infrastructure réseau en matière de bande passante et il n'a pas l'accès aux données du réseau dans lequel il est mis en place. Ce service a été assuré et géré, durant 6 ans, par une équipe au sein du RIPE NCC. Les fonctionnalités fournies par ce service comprennent le test de l'accessibilité à une destination via le *ping*. Les mesures effectuées étaient indépendantes des applications, elles dépendaient plutôt du réseau lui-même. RIPE NCC a arrêté la maintenance du TTM depuis le 1 juillet 2014 [31].

## I.6.2 ProbeAPI

*ProbeAPI* [37] est une plateforme de mesure de l'état d'un réseau, elle couvre 170 pays et des milliers d'ISPs (Internet Service Provider). *ProbeAPI* est utilisée par les développeurs, les administrateurs des réseaux et les chercheurs, ils peuvent lancer des mesures d'un réseau depuis différents réseaux. Le logiciel *ProbeAPI* s'exécute dans plusieurs systèmes : ordinateurs (Win32/64), Android via une installation dans les mobiles et les tablettes et dans des routeurs au sein du DD-WRT.

**DD-WRT** est un micrologiciel libre et gratuit, il est destiné aux routeurs sans fil et aux points d'accès. Il fonctionne avec un système d'exploitation Linux. Le rôle du DD-WRT est de remplacer le micrologiciel intégré aux routeurs par leurs fabricants. Ainsi, il est possible d'étendre des fonctionnalités du routeur en ajoutant d'autres fonctions supplémentaires.

ProbeAPI s'agit d'un logiciel qui tourne dans la machine de l'hébergeur. C'est pourquoi le suivi des réseaux dépend de la disponibilité de la machine qui le fait tourner. Cette dépendance affecte la disponibilité de la sonde logicielle, sa configuration et aussi les résultats de mesures.

Une étude comparative [38] entre les sondes et les sondes *ProbeAPI* est résumée dans le Tableau I.3. En fin de cette étude, ils concluent qu'en comparant les résultats des mesures ICMP effectuées par les deux plateformes, des contrastes intéressantes ont été constatées. Les sondes ont montré un comportement stable lors de la réalisation des mesures, les résultats sont peu variables car les sondes sont indépendantes de l'utilisateur. Cependant, il était constaté qu'une forte variabilité au cours du temps pour les sondes logicielles (*ProbeAPI*), car elles dépendent fortement de l'hébergeur; sa configuration réseau, sa disponibilité, etc. Enfin, la force des sondes logicielles comme *ProbeAPI* réside dans sa capacité à effectuer des mesures depuis la couche application, la plus proche de l'utilisateur. L'exemple de l'évaluation du Time To First Byte et le taux de transfert dans deux pays.

« *Le Time to First Byte (TTFB) est le temps de chargement du premier octet, c'est la mesure qui nous permet d'évaluer la vitesse d'accès à un serveur. Plus la mesure est basse et plus le serveur commencera à servir les ressources rapidement.* »<sup>a</sup>

a. Source : <https://www.skyminds.net/calculer-le-time-to-first-byte-ttfb-dun-serveur/>, consultée le 10/08/2018.

Malgré le niveau de couverture assuré par *ProbeAPI*, ces sondes se connectent et se déconnectent fréquemment, ce qui montre une forte volatilité. Cette volatilité est liée à la dépendance des sondes *ProbeAPI* de leur hébergeur; tant qu'il est connecté, la sonde *ProbeAPI* est prête pour effectuer les mesures. Toutefois, si l'hébergeur est déconnecté, la sonde *ProbeAPI* ne peut pas effectuer des mesures, d'où le basculement fréquent entre les deux états : connectée et déconnectée.

SONDE ATLAS	PROBEAPI
Matériel homogène a un comportement pré-visible	Matériel hétérogène a un comportement imprévisible
Connexions stables vu l'indépendance du software utilisateur	Connexions instables vu la dépendance du software utilisateur
Indépendance de l'OS et ses limitations ou vulnérabilités	Liaison à l'OS et ses limitations ou vulnérabilités, cependant utile pour les mesures au niveau application
La distribution des sondes est coûteuse, difficile de couvrir certaines régions	Mise en place du logiciel est rapide et moins chère, avec facilité de couvrir plusieurs régions
Les mesures HTTP se limitent aux ancres pour des raisons de sécurité	HttpGet, DNS et page-load sont disponibles via des bibliothèques Mozilla et chromium, et ce pour toutes les destinations

TABLE I.3 – Comparaison entre sondes Atlas et ProbeAPI

### I.6.3 Archipelago

Archipelago (Ark) [1] est l'infrastructure de mesures actives du CAIDA [2]. Elle est au service des chercheurs en réseau depuis 2007. Pour précision, c'est un Raspberry Pi 2nd gen. L'objectif de ce projet est de couvrir un maximum de régions afin de collecter un maximum de résultats de mesures, ensuite, produire des visualisations facilitant l'amélioration de l'Internet.

### I.6.4 DIMES

DIMES [36] est un logiciel qui devrait être installé dans une machine. Une fois installé, il fonctionne de sorte que la consommation d'énergie soit minimale et qu'il n'existe aucun impact sur les performances de la machine ou sur la connexion. L'objectif de *DIMES* est de collecter un maximum de données afin d'explorer la topologie d'Internet.

### I.6.5 SamKnows

SamKnows [7] est une plateforme globale de mesure des performances d'Internet, elle regroupe des ISPs, des ingénieurs, des universitaires, des codeurs et des organismes de régulation. Son objectif est d'évaluer les performances du haut débit des utilisateurs finaux et de trouver les problèmes avant que les clients ne commencent à se plaindre.

## I.7 Quelques cas d'utilisation des données collectées par les sondes

Plusieurs travaux ont exploité les données collectées par les sondes. Ces travaux peuvent être classés de plusieurs manières, par exemple, par thème, par type de mesures utilisé, etc. Nous distinguons les travaux ayant exploité les données collectées par les sondes à travers les mesures *built-in* ou bien ceux ayant utilisé les données des mesures personnalisées. Pour les premiers, ils permettent d'exploiter au mieux ces données sans surcharger le système Atlas, car ces données sont collectées, par défaut, quotidiennement. Tandis que les autres peuvent



introduire une charge supplémentaire sur ces sondes. D'autre part, certains travaux ont manipulé les données d'un type de mesure en particulier : traceroute, ping, HTTP, etc. Dans ce qui suit, nous allons présenter brièvement quelques travaux par thème.

La liste des travaux basés sur le projet Atlas est très longue. Nous avons essayé d'énumérer quelques projets, les classer par thèmes, toutefois, ce n'est pas un classement unique, tel qu'on peut retrouver un travail dans plus d'une catégorie, ou bien les classer par un autre classement.

### I.7.1 Détection des coupures d'Internet

Les données collectées par les sondes ont permis de valider certaines coupures d'Internet. Par exemple, la coupure affectant le point d'échange AMS-IX (Amsterdam Internet Exchange). En 2015, R. Kistelevi et al. [23] ont évalué l'état des pings en provenance des sondes à destination de trois ancres qui se trouvent dans AMS-IX. En effet, peu de pings ont réussi à atteindre leurs destinations, cependant, certains pings n'ont pas réussi à le faire. Les auteurs ont conclu qu'il existe un problème du réseau, et le problème concerne les ancres plutôt que les sondes ayant lancé le ping. De même pour les mesures DNS, ils ont constaté l'absence des données DNS censées être collectées par les ancres à destination du K-root pendant la période de la coupure.

### I.7.2 Aide à la prise de décision

L'utilisation des sondes n'est pas limitée au domaine de recherche seulement, ces sondes ont permis aussi d'aider à la prise de décision pour certaines implantations, par exemple la mise en place des équipements comme les routeurs, les *data center*, les IXPs (Internet Exchange Point), etc.

Les ingénieurs du Wikimedia Foundation et du RIPE NCC ont collaboré dans un projet [10] pour étudier la latence vers les sites du Wikimedia. L'idée était d'exploiter la distribution des sondes dans le monde en vue de mesurer la latence vers les sites du Wikimedia. L'étude de la latence va permettre d'améliorer l'expérience des utilisateurs vers ces sites en réduisant la latence. Comme Wikimedia avait l'intention d'étendre son réseau de datacenters, ils ont profité des résultats de cette étude pour choisir les emplacements futurs de leurs *data center*.

Un groupe de chercheurs africains a évalué le routage inter-domaine afin d'étudier les emplacements adéquats pour la mise en place d'un IXP [32]. Après avoir analysé les données des mesures collectées par les sondes, ils ont constaté que le trafic de et à destination de l'Afrique quitte le continent vers les États-Unis ou bien l'Europe pour revenir en Afrique, d'où l'intérêt d'investir dans la mise en place des IXPs en Afrique.

### I.7.3 Le suivi des censures

Il existe plusieurs pratiques pour appliquer la censure. Ces pratiques dépendent des objectifs de cette censure ; bloquer un site web, rediriger le trafic, filtrer l'accès à travers des mots clés, etc.

En 2014, des chercheurs ont examiné les incidents de type content-blocking en Turquie et en Russie tout en prenant en considération le respect de l'aspect éthique des données. Ils ont élaboré aussi un aperçu comparatif des différents outils permettant de mesurer l'état des réseaux informatiques [13]. C. Anderson et al. ont repris deux cas d'études où une censure a été appliquée : la Turquie et la Russie. L'idée de C. Anderson et al. est de créer des méthodes pour analyser ces censures en se basant sur les données collectées par les sondes.



En mars 2014, des utilisateurs turcs ont été interdits d'accéder au réseau social *Twitter*. Ce filtrage a été fait en utilisant *DNS Tampering* et *IP Blocking*. Comme ces deux pratiques sont évaluables avec les sondes, ils ont planifié des mesures vers plusieurs destinations et depuis quelques sondes. Les détails des mesures sont repris dans le Tableau I.4.

Cible	Type	Sondes	Fréquence (s)	Crédits
Twitter	SSL	10	3,600	2,400
YouTube	SSL	10	3,600	2,400
Tor	SSL	10	3,600	2,400
Twitter	DNS (U)	10	3,600	2,400
YouTube	DNS (U)	10	3,600	2,400
Twitter	Tracert	10	3,600	7,200

TABLE I.4 – Les détails des mesures effectuées dans le travail de C. Anderson [13]

L'analyse des données obtenues a permis de détecter six changements concernant les décisions du filtrage. Plus de détails se trouvent dans [13].

Quant à la Russie, les autorités ont décidé de mettre le blog d'*Alexei Navalny* sur *LiveJournal* dans la liste noire. En même temps, certains médias indépendants ont été aussi filtrés, l'exemple du site *grani.ru*. Pour le site *Grani*, les sondes Atlas ont reçu des réponses DNS aberrantes, d'où l'impossibilité de joindre *grani.ru*. Cependant, le filtrage du site *navalny.livejournal.com* a pris une autre forme, c'était une redirection d'adresse IP. La réponse d'une requête vers ce site donne 208.93.0.190 au lieu de 208.93.0.150. Ces deux adresses sont inclut dans le préfixe 208.93.0.0/22 géré par *LiveJournal Inc*. 208.93.0.190 correspond au contenu non-blacklisted, alors que 208.93.0.150 correspond au contenu correct.

## I.7.4 Le suivi des performances d'un réseau

### Les ancres

Les ancres ont des capacités avancées que les sondes. Les ancres servent comme cibles aux mesures des sondes. De plus, elles sont capables de fournir des détails sur l'état du réseau dans lequel elles sont déployées. S. Gasmi, un hébergeur d'une ancre, a développé un outil disponible au public<sup>21</sup>. A partir des données collectées par les ancres, cet outil permet d'analyser la qualité de la connectivité d'un réseau (ou d'un AS) et permet de suivre les changements relatifs à la topologie des réseaux.

Par exemple, il a constaté que la vérification du BGP Prepending (Border Gateway Protocol) et des communautés BGP peut être faite en considérant les éléments suivants : adresse IP source, AS source, pays, le RTT du ping et les chemins du traceroute. En particulier, S. Gasmi a évalué deux corrélations. Dans un premier temps, il a visualisé la corrélation entre l'AS path et Round Trip Time (RTT). Il a regroupé des sondes par pays, ensuite, il a calculé, par ce pays, la moyenne du nombre de sauts et la moyenne du RTT des requêtes à destination de l'ancre depuis ces sondes. La Figure I.5 reprend les résultats obtenus sans donner des renseignements à propos la période des données. Pour les sondes en provenance de la France, le nombre de sauts et le RTT entre les sondes déployées en France sont faibles car l'ancre (la cible) se trouve aussi en France. Ensuite, S. Gasmi a mesuré le RTT entre des sondes dans le monde et son ancre, il a aussi visualisé le nombre de sauts parcourus entre des sondes à travers le monde et son ancre.

21. Source : <http://ripeanchor.sdv.fr/>, consultée le 08/08/2018.

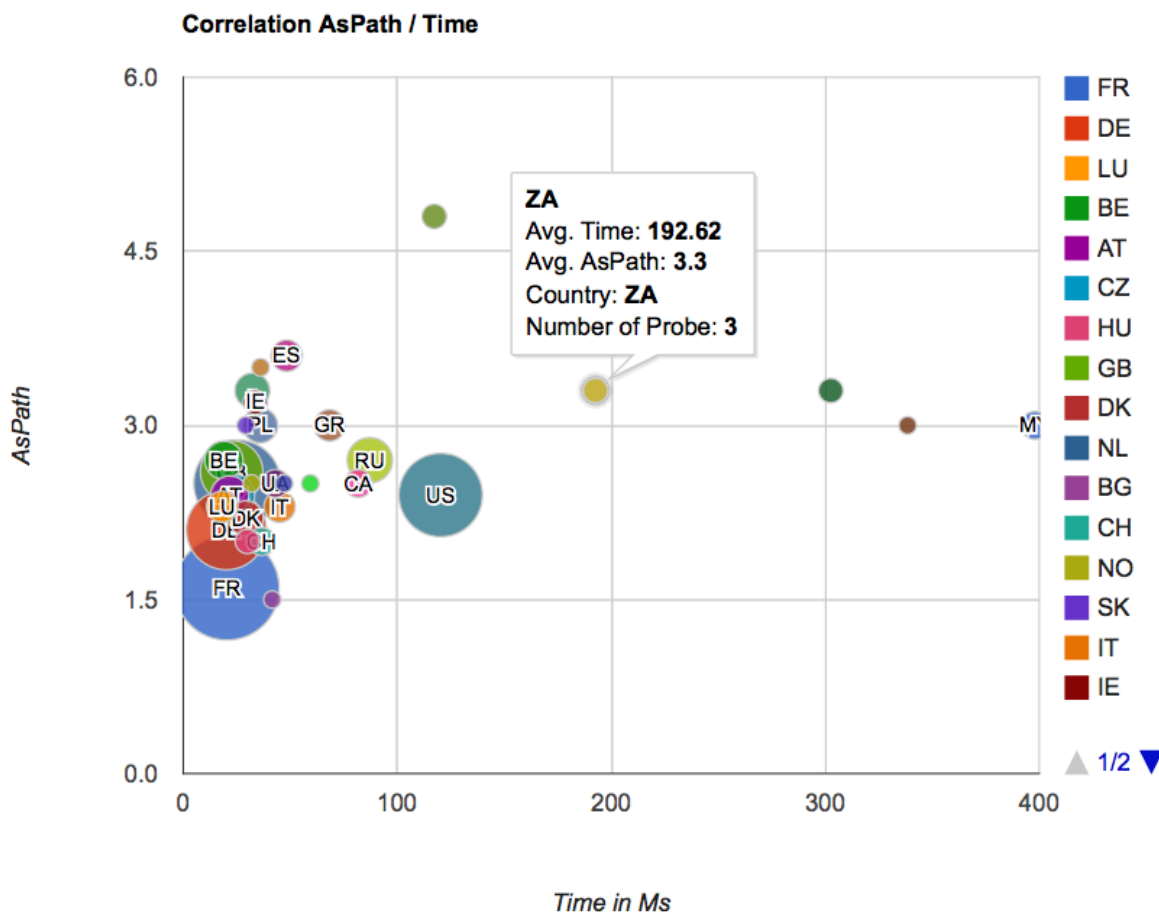


FIGURE I.5 – La corrélation entre la moyenne des AS paths et la moyenne des RTTs [18]

Ces deux visualisations permettent d’avoir une idée sur la latence entre certains pays et le pays de l’ancrage en question. Plus de détails sur l’approche sont disponibles dans [18].

### La vérification de la cohérence du Traceroute

Les chemins parcourus par traceroute pour aller d’une source  $s$  vers une destination  $d$  changent au cours du temps pour plusieurs raisons. Par exemple, suite à un changement BGP, à une répartition des charges, à des pannes des routeurs, à des pannes des liens physiques, etc.

*Traceroute Consistency Check* peut reprendre les chemins obtenus via traceroute au cours du temps. L’objectif est de suivre les nœuds apparaissant dans le chemin allant de  $s$  à  $d$  aux instants  $t$ ,  $t + 1$ ,  $t + 2$ , etc, et cela afin de voir les nœuds traversés plus fréquemment au cours du temps. Le chemin est mis à jour via Atlas streaming API.

L’outil proposé dessine les chemins traceroute comme étant un graphe dirigé, chaque nœud est coloré suivant sa cohérence. Le code source du projet est disponible sur GitHub [15]. La Figure I.6 présente un exemple de la visualisation proposée. Ce résultat concerne la mesure 1663314<sup>22</sup>. Ce sont des traceroutes à destination de l’adresse 213.171.160.1, effectués entre le 02/05/2014 13 : 00 et le 03/05/2014 15 : 00.

### BGP+traceroute

22. Source : <https://atlas.ripe.net/measurements/1663314/>, consultée le 05/08/2018.

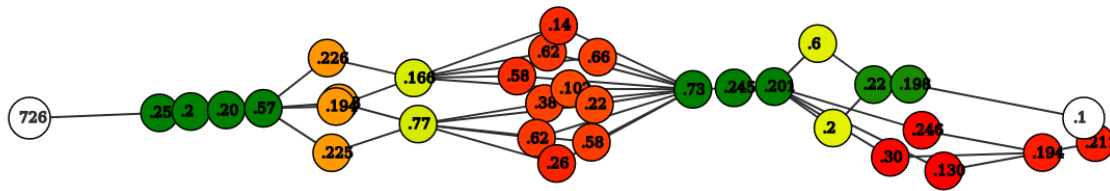


FIGURE I.6 – Visualisation des changements des chemins traceroute [15]

C'est une combinaison des données BGP (RIPE RIS) et traceroute (RIPE Atlas). L'objectif de ce projet est de partir d'un AS path pour enfin géolocaliser les ASs. L'idée est de prendre un AS path des données RIPE RIS, puis, récupérer le préfixe (bloc d'adresses IP) annoncé via cet AS path, ensuite, lancer un traceroute vers une des adresses du bloc. Enfin, géolocaliser les ASs via les données du traceroute. Le code source et la présentation de ce projet sont disponibles sur GitHub [21, 20].

### BGP Atlas Monitor (BAM)

Le projet *BAM* porte sur la visualisation, en temps réel, des informations utiles pour les opérateurs des réseaux. Par exemple, BAM montre la visibilité des préfixes obtenus par RIPE RIS. De plus, il est possible de voir le délai du *ping* obtenu via les sondes. Le code source est disponible sur GitHub [19]. En fournissant un ASN (identifiant d'un AS), *BAM* récupère les préfixes IPv4 et IPv6 et leur visibilité et il énumère aussi les sondes dans cet AS. L'outil offre les fonctionnalités suivantes :

- les préfixes annoncés par un ASN ;
- la visibilité d'un ASN ;
- la visibilité d'un préfixe ;
- la liste des sondes par AS ;
- les objets route des préfixes.

### Prédiction des routeurs provoquant la perte des paquets

Dans l'étude [17], R. Fontugne et al. ont modélisé le comportement des routeurs, ils ont développé un modèle qui permet d'estimer l'endroit de la perte des paquets. A partir des traceroutes passant par un routeur  $r$  à une destination  $d$ , ils ont construit un modèle de forwarding pour ce routeur. Ce modèle reprend les prochains sauts (routeurs) et la fréquence de passage par ces derniers. Si le routeur  $r$  change le prochain saut qui a eu "l'habitude" de traverser pour atteindre  $d$ , alors cela pour être un indicateur de l'origine de la perte des paquets.

### I.7.5 Le suivi des détours dans un trafic local

Dans leur travail [12], E. Aben et al. avaient l'objectif de voir comment les mesures d'Atlas peuvent fournir un aperçu sur le chemin du trafic local à un pays. Précisément si ce trafic traverse un autre pays en revenant au pays du départ. Ce qui pourrait aider à améliorer les performances et l'efficacité des IXPs. L'objectif de leur travail est d'analyser les chemins identifiés dans le trafic d'Internet entre les sondes dans un pays donné et essayer d'identifier si le trafic traverse des IXPs.

France-IX est un point d'échange Internet (IXP) français créé en juin 2010. Afin d'apprendre la topologie de routage, un RIS route collector (RRC21) a été installé au sein du France-IX. Actuellement, la France compte 755 sondes et 9 ancres. Une ancre sur les 9 est installée au sein de France-IX.

Une des questions posées c'est de savoir si le trafic local de la France reste localement. Les sondes ne permettent pas de mesurer le trafic entre deux points, alors qu'elles permettent de calculer le chemin entre deux points, adresses IP, ce qui permet d'inférer les sauts par lesquels il passe le trafic. Le travail [11] s'intéresse au trafic depuis et vers une sonde en France en se basant sur l'étude [12].

Les résultats obtenus de l'analyse des détours peuvent être intéressants pour les opérateurs des réseaux afin d'améliorer leurs services, aussi intéressants pour les IXPs tels qu'ils peuvent proposer des services de peering dans les endroits où il le faut.

### I.7.6 Visualisation : indicateurs et dashboard

L'objectif de certains travaux est d'exploiter les données collectées par les sondes pour concevoir des tableaux des indicateurs. Par exemple, à partir des données de connexion/déconnexion des sondes, on peut visualiser les sondes connectées, déconnectées, abandonnées. Un autre projet avait comme objectif la reconstruction d'un graphe reprenant les routeurs (nœuds) impliqués dans certains traceroutes, ainsi, identifier les nœuds les plus traversés. D'autres travaux ont repris les détails de la latence, essentiellement, sont les valeurs des RTTs dans les pings et les traceroutes qui permettent de visualiser ce type d'information.

## I.8 Conclusion

Dans ce chapitre, nous avons présenté les sondes et leur fonctionnement, ainsi que quelques travaux ayant impliqué des données collectées par ces sondes. Ces travaux ont visé différents thèmes, tels que la prise de décision, le suivi des censures, la conception des tableaux de visualisation, etc. Dans le reste de ce document, le choix a été porté sur un sujet lié aux performances des réseaux informatiques. Il s'agit d'un outil de détection des anomalies dans les délais des liens dans les réseaux informatiques, en se basant sur les mesures de traceroute. Une présentation détaillée de l'algorithme de détection est donnée dans le chapitre ??.

# Chapitre II

## Détection des anomalies dans les délais d'un lien

Dans le présent chapitre, nous présentons l'outil de détection des anomalies dans les délais d'un lien. Cet outil a été conçu dans le cadre du travail [17] de Fontugne et al.. Nous avons choisi de réutiliser ce travail qui se base sur les traceroutes collectés par les sondes, et ce en vue d'évaluer son implémentation avec quelques technologies Big Data.

### II.1 Présentation générale

Dans leur travail [17], Fontugne et al. ont exploité la grande répartition des sondes dans le monde afin d'étudier le délai des liens topologiques sur les réseaux informatiques et de tracer son évolution ; c'est le délai entre deux routeurs adjacents sur Internet.

L'idée de ce travail est de collecter les résultats des requêtes traceroute effectuées par les sondes, d'en déterminer une valeur de référence du délai du lien en question sur base de son historique, et ensuite de comparer la référence avec la valeur courante. Cette référence est mise à jour au fur et au mesure de l'analyse de nouveaux traceroutes. La comparaison de la référence avec la valeur courante ne se déclenche que lorsque cette référence est assez représentable de l'état réel <sup>1</sup> du lien.

Le travail [17] de Fontugne et al. comprend trois méthodes. Nous allons travailler sur la méthode ayant pour objectif la détection des changements des délais que subissent les liens intermédiaires dans les traceroutes.

### II.2 Pourquoi analyser les délais des liens ?

Pour certains travaux, il n'était pas possible de les reprendre suite au manque de détails et de l'accès à certaines ressources réseaux utilisées dans ces travaux <sup>2</sup>. Par exemple, afin d'étudier la censure dans un pays donné, il faut être au courant des spécificités de ce pays, les circonstances politiques et sociales, les opposants, etc. Un autre exemple concerne le sujet de la détection des détours du trafic local dans un pays. Anant Shah et al. ont travaillé sur la détection des détours, ils ont présenté dans [34] les contraintes relatives à ce sujet : la difficulté d'avoir

---

1. Avec une probabilité de 95%.

2. Par exemple, les informations du peering entre les Systèmes Autonomes, la géolocalisation des adresses IP, etc.

une géolocalisation exacte d'une adresse IP d'une part et l'absence des informations de peering entre les ASs d'autre part. Ces dernières peuvent changer complètement les conclusions finales.

Le travail sur lequel nous nous basons [17] pour l'évaluation de quelques technologies Big Data s'inscrit dans les travaux traitant les performances des réseaux informatiques. Dans la suite de ce document, ce travail est appelé travail de référence. Ce travail a été choisi comme référence pour plusieurs raisons :

**Les données utilisées** Les auteurs de ce travail ont exploité des données déjà présentes dans le dépôt d'Atlas (voir la section I.4.6 concernant les sources de données Atlas). Ainsi, il n'y a pas besoin de lancer des mesures qui nécessitent la possession d'assez de crédits (voir la section sur les crédits d'Atlas dans I.4.2). De plus, les mesures intégrées montrent plus de stabilité par rapport aux mesures personnalisées. Les destinations des mesures intégrées sont prédéfinies, généralement ce sont des instances des serveurs DNS et des serveurs gérés par RIPE NCC. Alors que, les mesures personnalisées peuvent concerner des destinations moins stables en terme de disponibilité.

**La clarté du travail** La communauté Atlas est active en nombre de travaux publiés. Certains de ces travaux reprennent seulement les résultats finaux. Pour d'autres, la méthodologie est bien détaillée. Le reste des travaux reprennent brièvement la méthodologie adoptée. En ce qui concerne le travail choisi, la méthodologie est bien détaillée.

**La disponibilité du code source** La détection des anomalies proposée a été mise en pratique à travers un outil. Le code source de cet outil est disponible sur GitHub [16]. L'accès au code source de l'outil nous a permis de bien comprendre le processus de détection.

**La possibilité de la validation des résultats** Comme il est cité dans le travail [17], les auteurs ont démontré la cohérence de l'outil de détection avec des événements réels comme les attaques DDOS.

Une attaque de type Distributed Denial of Service (*DDOS*) vise la disponibilité d'un serveur en surchargeant ce dernier avec un trafic depuis différentes sources, d'où le terme *Distributed*.

## II.3 Etude des délais des liens

### II.3.1 Les données utilisées dans l'analyse des délais

La méthode conçue pour la détection des changements des délais se base sur des fondements statistiques. Ces derniers sont capables de montrer leurs performances si la taille des échantillons<sup>3</sup> sur lesquels ils se basent est grande. Afin de surveiller un grand nombre de liens sur Internet, il faut avoir un grand nombre de sondes avec une certaine diversité. En ce qui concerne le travail de référence, les auteurs ont utilisé des traceroutes en provenance des mesures intégrées et des traceroutes à destination des ancres. Le Tableau II.1 fournit plus de détails sur les traceroutes utilisés : le nombre de traceroutes est donné suivant leur type d'adressage

3. C'est l'ensemble des RTTs différentiels caractérisant un lien.

(IPv4 et IPv6) et les sondes ayant été impliquées dans ces derniers. Ces chiffres correspondent à la période du 1 mai au 31 décembre 2015.

	Nombre de traceroutes (milliards)	Nombre de sondes
IPv4	2.8	11, 538
IPv6	1.2	4, 307

TABLE II.1 – Récapitulatif des traceroutes utilisés dans le travail de référence

### II.3.2 Le principe de détection des changements des délais

Le processus de détection des anomalies dans les délais d'un lien repose sur une métrique caractérisant un lien dans un réseau informatique : le RTT différentiel.

Définissons d'abord le RTT (Round-Trip Time) :

**RTT** est obtenu en calculant la différence entre le timestamp associé à l'envoi du paquet vers une destination et le timestamp associé à la réception de la réponse ICMP (Internet Control Message Protocol) de cette destination. C'est une métrique utilisée pour évaluer les performances d'un réseau en matière de temps de réponse.

Les mesures du RTT sont fournies par les utilitaires traceroute et ping. En ce qui concerne traceroute, il fournit les sauts (routeurs intermédiaires) impliqués dans le chemin de forwarding avec leurs RTTs, c'est le chemin parcouru par le trafic entre la source et la destination. On note qu'un RTT inclut le temps de transmission, du queuing et du traitement.

#### Définition du RTT différentiel d'un lien

La Figure II.1 (a) illustre le RTT entre la sonde P et deux routeurs B et C. Le RTT différentiel du lien entre B et C adjacents, noté  $\Delta_{PBC}$ , est la différence du RTT entre la sonde P et B (bleu) d'une part, et du RTT entre la sonde P et C (rouge) d'autre part, comme illustré dans la Figure II.1 (b).

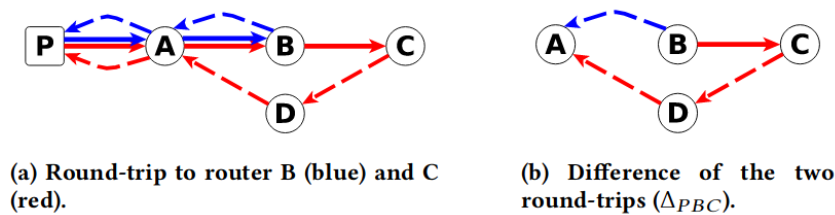


FIGURE II.1 – (a) Le RTT entre la sonde P et les routeurs B et C. (b) La différence entre les chemins de retour depuis les routeurs B et C vers la sonde P. Source : [17]

Le RTT différentiel  $\Delta_{PBC}$  de la Figure II.1 (b) est décomposé comme suit :

$$\begin{aligned}
 RTT_{PB} &= \delta_{PA} + \delta_{AB} + \delta_{BA} + \delta_{AP} \\
 RTT_{PC} &= \delta_{PA} + \delta_{AB} + \delta_{BC} + \delta_{CD} + \delta_{DA} + \delta_{AP} \\
 \Delta_{PBC} &= RTT_{PC} - RTT_{PB} \\
 &= \delta_{BC} + \delta_{CD} + \delta_{DA} - \delta_{BA} \\
 &= \delta_{BC} + \varepsilon_{PBC}
 \end{aligned} \tag{II.1}$$

où  $\delta_{BC}$  est le délai du lien  $BC$  et  $\varepsilon_{PBC}$  est la différence entre les deux chemins de retour :  $B$  vers  $P$  et  $C$  vers  $P$ . Le chemin de retour est celui présenté dans la Figure II.1 (b).

### Le principe de la détection des changements des délais

L'évolution du délai d'un lien est déduit de l'évolution de son RTT différentiel. Reprenons d'abord l'équation (II.1) du RTT différentiel du lien  $BC$ . Les deux variables  $\delta_{BC}$  et  $\varepsilon_{PBC}$  sont affectées par des facteurs différents. La valeur de  $\delta_{BC}$  dépend de l'état des deux routeurs  $B$  et  $C$  et ne dépend pas de la sonde  $P$ . Tandis que la valeur de  $\varepsilon_{PBC}$  dépend de la sonde  $P$ .

Supposons qu'on dispose d'un nombre  $n$  de sondes Atlas  $P_i$ ,  $i \in [1, n]$ , telles que toutes les sondes ont un chemin de retour  $\varepsilon_{P_iBC}$  différent depuis  $B$  et depuis  $C$ . Les  $n$  RTTs différentiels  $\Delta_{P_iBC}$  du lien  $BC$  pour chacune des sondes  $P_i$  partagent la même composante  $\delta_{BC}$  :

$$\begin{aligned}\Delta_{P_1BC} &= \delta_{BC} + \varepsilon_{P_1BC} \\ \Delta_{P_2BC} &= \delta_{BC} + \varepsilon_{P_2BC} \\ &\dots\dots\dots \\ \Delta_{P_iBC} &= \delta_{BC} + \varepsilon_{P_iBC} \\ \Delta_{P_nBC} &= \delta_{BC} + \varepsilon_{P_nBC}\end{aligned}$$

Les valeurs de  $\varepsilon_{P_iBC}$  sont indépendantes. L'indépendance de ces valeurs implique que la distribution  $\Delta_{P_iBC}$  est estimé stable au cours du temps si  $\delta_{BC}$  est constant. Toutefois, un changement significatif de la valeur de  $\delta_{BC}$  influence les valeurs des RTTs différentiels, ainsi, la distribution des RTTs différentiels  $\Delta_{P_iBC}$  change. D'où l'idée de l'évolution du délai d'un lien déduit à partir de l'évolution de son RTT différentiel.

### Caractérisation des délais d'un lien

Les auteurs du travail de référence [17] ont évalué le délai d'un lien en évaluant son RTT différentiel. Cette évaluation repose sur le théorème central limite énoncé ci-dessous <sup>4</sup>.

Soient  $X_1, \dots, X_n$   $n$  variables aléatoires *i.i.d.* de moyenne  $\mu$  et variance  $\sigma^2$  finies ( $\sigma > 0$ ). Soient  $S_n$  la somme des  $n$  variables aléatoires et

$$Z_n := \frac{S_n - n\mu}{\sqrt{n}\sigma}. \quad (\text{II.2})$$

Alors la fonction de répartition de  $Z_n$  tend vers celle d'une loi  $N(0, 1)$  lorsque  $n$  tend vers l'infini.

Remarque : on peut aussi écrire le résultat suivant :

$$S_n \approx N(n\mu, n\sigma^2) \quad (\text{II.3})$$

L'application de ce théorème, dans l'analyse des délais des liens, implique que quelque soit la distribution des RTTs différentiels, la moyenne arithmétique d'un échantillon est distribuée normalement si la taille de l'échantillon est relativement grande. En pratique, si un lien subit un

---

4. Voir le théorème 4.11.1 dans [27].



changement anormal, la distribution de la moyenne des RTTs différentiels dévie de la distribution normale, par conséquent, la moyenne des RTTs différentiels ayant produit ce changement est identifiée comme étant une anomalie.

Après avoir évalué les premiers résultats d'application de ce théorème, les auteurs ont conclu que l'utilisation de la médiane, au lieu de la moyenne, a montré plus de performance en terme de détection des anomalies. Afin de tenir compte de l'incertitude dans la médiane calculée, ayant la capacité d'identifier un changement anormal, les auteurs ont calculé l'intervalle de confiance de cette dernière. Par définition<sup>5</sup>, un intervalle de confiance à un niveau  $\gamma$ , d'un paramètre  $m$  fixé mais inconnu est l'intervalle  $(u(X_1, \dots, X_n), v(X_1, \dots, X_n))$  tel que :

$$\mathbb{P}(u(X_1, \dots, X_n) < m < v(X_1, \dots, X_n)) \geq \gamma \quad (\text{II.4})$$

## II.4 L'étude des délais des liens en pratique : l'évolution du RTT différentiel des liens

L'évolution des RTTs différentiels est une application du principe décrit dans la section II.3.2. Pour un lien donné, le suivi s'étale sur plusieurs périodes consécutives.

### II.4.1 Les étapes principales de détection

Les étapes du processus de détection peuvent être résumées dans les éléments suivants :

- (i) Vérification de la validité de tout traceroute.
- (ii) Calcul des RTTs différentiels de chaque lien identifié dans les traceroutes.
- (iii) Caractérisation des liens avec le théorème central limite (CLT).
- (iv) Comparaison de l'état de chaque lien avec sa référence et l'identification des anomalies.
- (v) Mise à jour de la référence du lien.

### II.4.2 Description des paramètres de l'analyse des délais

La détection des changements des délais nécessite l'ajustement de quelques paramètres. Ces paramètres ont des valeurs par défaut définies dans le travail de référence. Nous décrivons ci-dessous tous les paramètres : ceux ayant une valeur par défaut et ceux à ajuster :

**traceroutes** : ce sont l'ensemble des résultats de requêtes traceroute. La détection des anomalies est effectuée suivant le nombre de traceroutes analysés et la période précisée.

**start** : c'est la date de début de l'analyse. Seuls les traceroutes effectués par les sondes à partir de cette date sont analysés.

**end** : c'est la date marquant la fin de l'analyse. Comme le paramètre *start*, c'est la date maximales des traceroutes à considérer effectués par les sondes.

**timeWindow** : ce paramètre est exprimé en secondes. Il correspond à la durée de chaque période des  $n$  périodes de l'analyse. Ces périodes ont la même durée. Autrement dit, la durée entre *start* et *end* est divisée par *timeWindow*. Ceci est illustré à la Figure II.2.

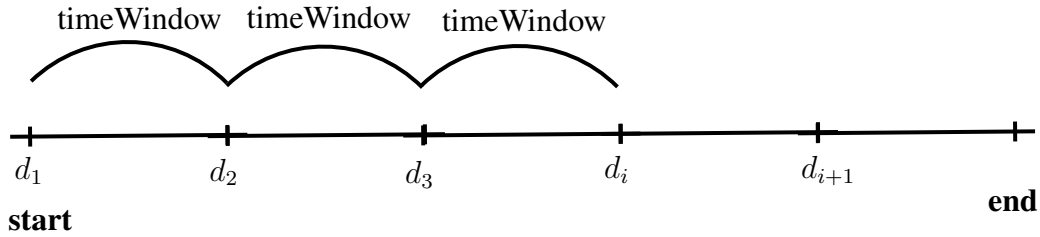


FIGURE II.2 – Illustration des périodes de l'analyse entre la date de début et la date de fin.

Avec  $d_i$  dénote le début de la  $i^{\text{ème}}$  période,  $i \in [1, n]$ . Par défaut, *timeWindow* vaut 3600 secondes.

**minSeen** : est le nombre de fenêtres à atteindre avant de commencer la comparaison du RTT différentiel d'un lien avec sa référence.

**alpha** : noté  $\alpha$ ,  $\alpha \in [0, 1]$ , c'est le paramètre de la moyenne mobile exponentielle calculée.

La moyenne mobile exponentielle est utilisée pour calculer la moyenne des RTTs différentiels de référence durant une période  $d_k$ . La valeur de la médiane des RTTs différentiels de référence  $\bar{m}_t$  pour la période  $t$  et le lien  $l$  :

$$\bar{m}_t = \alpha m_t + (1 - \alpha) \bar{m}_{t-1}$$

$m_t$  la médiane des RTTs différentiels observée pour  $l$  durant la période  $t$ .

$\bar{m}_{t-1}$  la médiane des RTTs différentiels de référence durant la période  $t - 1$ .

Pour précision,  $\{m_t\}$  et  $\{\bar{m}_t\}$  désignent deux ensembles différents. Le premier est l'ensemble des médianes des RTTs différentiels de chaque période  $d_k$ . Or, le deuxième est l'ensemble des médianes des RTTs différentiels, de référence, construite en utilisant la méthode de la moyenne mobile exponentielle. Le calcul de cette dernière prend en compte les médianes des RTTs différentiels précédentes ainsi que la médiane des RTTs différentiels courante. La participation de ces dernières dans le calcul de la référence est dirigé par le paramètre  $\alpha$ . Plus de détails sur ce calcul est donné dans la section[!].

Le paramètre  $\alpha$  contrôle l'importance des mesures précédentes par rapport aux mesures récentes. De ce fait, «plus  $\alpha$  est proche de 1 plus les observations récentes influent sur la prévision, à l'inverse un  $\alpha$  proche de 0 conduit à une prévision très stable prenant en compte un passé lointain<sup>6</sup>». Dans le travail de référence, le paramètre  $\alpha$  vaut par défaut 0.01.

**confInterval** : ce paramètre indique le niveau de confiance de l'intervalle de confiance de la médiane calculée des RTTs différentiels. Précisément, cet intervalle de confiance calculé est de niveau de confiance égal à  $1 - \text{confInterval}$ . L'intervalle de confiance d'une moyenne est défini comme représentant les valeurs probables que peut prendre cette moyenne, si on accepte une marge d'erreur définie à l'avance. Il existe plusieurs méthodes pour calculer l'intervalle de confiance d'une proportion. Parmi les critères impliqués sur le choix de la méthode du calcul, on note le nombre total d'expériences.

Au RTT différentiel courant et au celui de référence d'un lien donné sont associés des intervalles de confiance. le calcul des intervalles de confiance est approché par le score

5. Definition 2.1 dans [26].

6. Voir le Lissages Exponentiels dans [https://www.math.u-psud.fr/~goude/Materials/time\\_series/cours3\\_lissage\\_expo.pdf](https://www.math.u-psud.fr/~goude/Materials/time_series/cours3_lissage_expo.pdf), consultée le 30/09/2018.

de Wilson. Le score de wilson a été adopté car il donne des résultats même si la distribution pour laquelle l'intervalle de confiance est calculé est petite. Pour chaque lien et pour chaque période  $d_k$ , deux intervalles de confiances sont calculés, pour ensuite évaluer le chevauchement entre ces deux intervalles. L'outil de détection utilise une marge d'erreur égale à 0.05. Les détails du calcul des intervalles de confiances sont fournis dans la section[!].

### II.4.3 Processus de détection des anomalies en deux phases

La détection des anomalies dans les délais des liens peut être organisée en deux phases : la préparation de données et la détection des alarmes.

#### Phase I : préparation de données.

- **findAllBins (I.1)** : nous cherchons la liste des périodes ([bin]) entre `start` et `end`. La durée de chaque bin vaut `timeWindow`.
- **findTraceroutesByBin (I.2)** : pour toute période (bin), nous récupérons les traceroutes capturés durant cette dernière. Le nombre de traceroutes associés à chaque bin dépend des traceroutes disponibles à l'analyse ([traceroute]).
- **removeInvalidTraceroutes (I.3)** : certains traceroutes ne réussissent pas à atteindre la destination prévue, dans ce cas, ces traceroutes sont ignorés. De même, les sauts ayant des RTTs invalides sont aussi ignorés.
- **aggregateRttsByHop (I.4)** : étant donné que chaque saut (hop) est décrit par plus d'un RTT, nous calculons la médiane de ces RTTs (`rttAgg`), et ce par source de signal.
- **linkInference (I.5)** : chaque deux sauts consécutifs d'un traceroute forment un lien. Pour tout traceroute d'un bin donné, nous déduisons les liens possibles, ensuite, nous caractérisons tout lien (`ip1`, `ip2`) avec son RTT différentiel (`rttDiff`).
- **sortLinks (I.6)** : nous ordonnons les couples d'adresses IP (`ip1`, `ip2`) de chaque lien suivant l'ordre alphanumérique.
- **mergeLinks (I.7)** : nous fusionnons les liens avec leurs caractéristiques. En résultat, les liens (`ip1`, `ip2`) et (`ip2`, `ip1`) sont fusionnés et sont représentés par le lien (`ip1`, `ip2`) si  $(ip1, ip2) < (ip2, ip1)$ .  $<$  désigne l'ordre alphanumérique.
- **resumeData (I.8)** : jusqu'à cette étape, les liens sont organisés par période (bin). A travers `resumeData`, nous réorganisons chaque lien de toutes les périodes de sorte que chaque RTT différentiel soit associé à la période correspondante ; la période pendant laquelle il a été identifié. C'est pourquoi les deux listes [`rttDiff`] et [`bin`] ont la même longueur.

**Phase II : détection des alarmes.** A la fin de la phase I, nous avons tous les liens identifiés avec leurs RTTs différentiels.

- **findAllBins (II.1)** : nous générons les mêmes périodes ([bin]) que celles générées à l'étape (I.1).
- **findRTTDiffByBin (II.2)** : nous reprenons les périodes générées dans l'ordre chronologique et pour chaque période nous cherchons les RTTs différentiels correspondants. Soit `dist` l'ensemble des RTTs différentiels correspondants à la période courante. Seules les périodes ayant enregistré au moins quatre RTTs différentiels qui seront considérées.

- **wilsonScoreProcess (II.3)** : le calcul de l'intervalle de confiance à partir du score de Wilson nécessite d'avoir le nombre total des expériences, le nombre d'expérience réussies et le risque d'erreur. Nous calculons les deux valeurs du score de Wilson suivant les étapes suivantes :

1. Soit  $n$  la taille de la distribution, nous calculons le score de Wilson et cela donne le couple de valeurs ( $low$ ,  $hight$ );
2. nous multiplions  $low$  et  $hight$  par  $n$ , ainsi nous avons  $(lo, hi) = (n * low, n * hight)$ ;
3. nous récupérons la partie entière du  $lo$  et  $hi$  et nous obtenons  
 $(l, h) = (int(lo), int(hi))$ <sup>7</sup>;
4. nous trions  $dist$  et nous parcourons cette dernière pour trouver les RTTs différentiels se trouvant au rang  $l$  et  $h$  dans  $dist$ , respectivement sont  $rttDiffL$ ,  $rttDiffH$ . Ainsi :  
 $rttDiffL = dist[l]$  et  $rttDiffH = dist[h]$ , avec  $l, h \in [0, n[$ .

- **updateCurrentLinkState (II.4)** : `current` est l'objet encapsulant l'état courant du lien à une période donnée, il a les éléments suivants :

- ☐ le RTT différentiel médian<sup>8</sup> (`median`), `currentMedian = median(dist)`
- ☐ la borne inférieure de l'intervalle de confiance (`ciLow`),  
`currentCiLow = median - rttDiffL`
- ☐ la borne supérieure de l'intervalle de confiance (`ciHight`),  
`currentCiHight = rttDiffH - median`

- **updateReferenceLinkState (II.5)** : `reference` est l'objet décrivant l'état de référence du lien à une période donnée. Il reprend les mêmes éléments que l'objet `current`, à savoir, `median`, `ciLow` et `ciHight`. Ainsi, l'état référence est décrit par la médiane du RTT différentiel (`referenceMedian`), la borne inférieure de l'intervalle de confiance de référence (`referenceCiLow`) et enfin la borne supérieure de l'intervalle de confiance de référence (`referenceCiHight`).

L'état courant décrit un lien à une période donnée, alors que l'état référence décrit l'état d'un lien à une période donnée en prenant en compte aussi les périodes qui précèdent la période courante. La mise à jour de `reference` est réalisée selon trois cas :

1. Cas 1 : tant que nous n'avons pas construit une référence assez représentable de l'état du lien, nous mettons à jour la référence (`reference`) comme suit :

$$referenceMedian = median(dist) \quad (II.5)$$

$$referenceCiLow = rttDiffL \quad (II.6)$$

$$referenceCiHight = rttDiffH \quad (II.7)$$

2. Cas 2 : une fois nous atteignons le nombre nécessaire de mise à jour de la référence assurant sa représentativité, nous mettons à jour l'état référence en remplaçant, pour chaque période, la médiane par `aggr_median`, la borne inférieure par `aggr_ciLow` et la borne supérieure par `aggr_ciHigh`.

7. La fonction `int()` est utilisée pour avoir la partie entière d'un nombre réel.

8. `median(list)` calcule la valeur médiane de la liste `list`.

- soit *aggr\_median* la médiane de tous les RTTs différentiels médians de référence calculés pour toutes les périodes précédentes.
  - soit *aggr\_ciLow* la médiane de toutes les bornes inférieures calculées pendant les périodes précédentes.
  - soit *aggr\_ciHigh* la médiane de toutes les bornes inférieures calculées pendant les périodes précédentes.
3. Cas 3 : une fois la référence est assez représentable, nous mettons à jour cette dernière comme suit<sup>9</sup> :

$$referenceMedian = 0.99 * referenceMedian[-1] + 0.01 * currentMedian$$

$$referenceCiLow = 0.99 * referenceLow[-1] + 0.01 * rttDiffL$$

$$referenceCiHight = 0.99 * referenceHight[-1] + 0.01 * rttDiffH$$

- **alarmsDetection (II.6)** : la détection des anomalies est déclenchée à partir du cas 3. Si une anomalie est détectée, nous mettons à jour la liste des alarmes (*alarmsValues*) avec le RTT différentiel médian (*currentMedian*) ainsi que la liste des dates (*alarmsDates*) qui correspondent à ces anomalies. Les anomalies sont détectées en évaluant le chevauchement entre l'intervalle de confiance courant et celui de référence. La période courante présente une anomalie si la condition suivante est vraie<sup>10</sup> :

Listing II.1 – Conditions de présence d'une anomalie pour un lien

```

1 currentMedian [-1] - currentCiLow [-1] > referenceHi [-1] or
2 currentMedian [-1] + currentCiHigh [-1] < referenceCiLow [-1] ) and
3 np.abs ( currentMedian [-1] - reference [-1] ) > 1
```

**Illustrations des deux phases de détection** La Figure B.1 reprend les étapes de la phase I. Chaque étape est définie par ses entrées et ses sorties ainsi que son objectif et la Figure B.2 illustre aussi les principales étapes de la phase II.

9. Nous précisons que *var*[-1] dénote la dernière valeur que *var* avait.

10. Nous notons par [-1] l'élément précédent d'une mesure. Par exemple, *currentMedian*[-1] est la médiane des RTTs différentiels de la période qui précède la période courante.

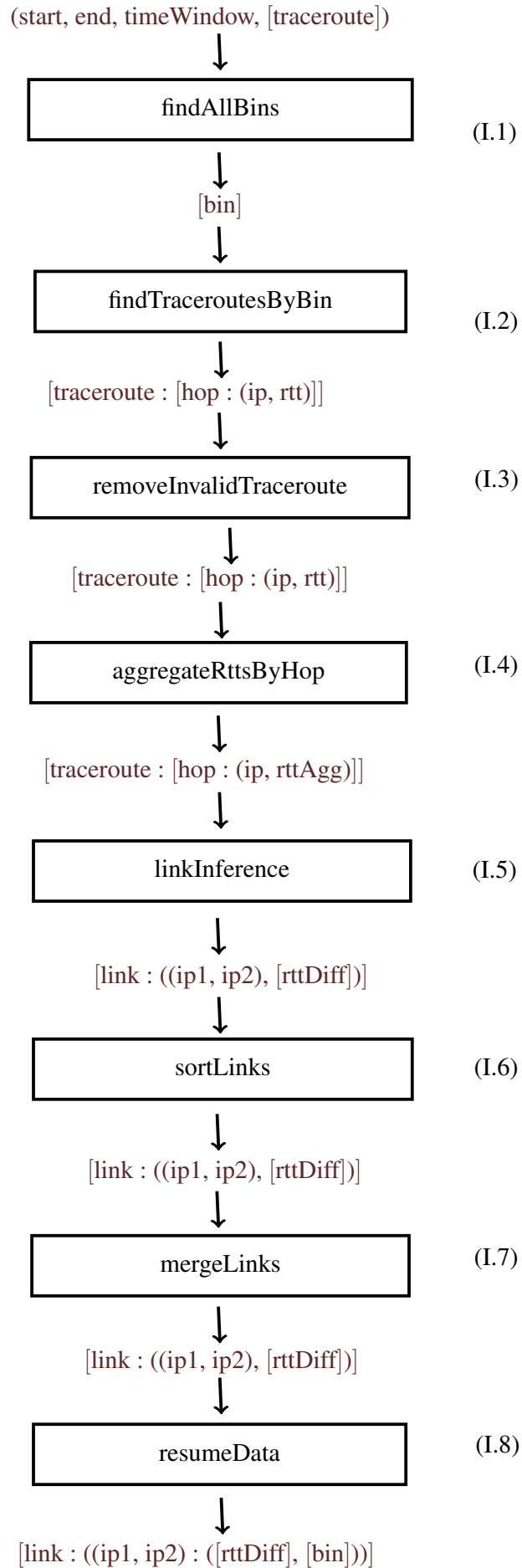


FIGURE II.3 – Phase I : préparation de données

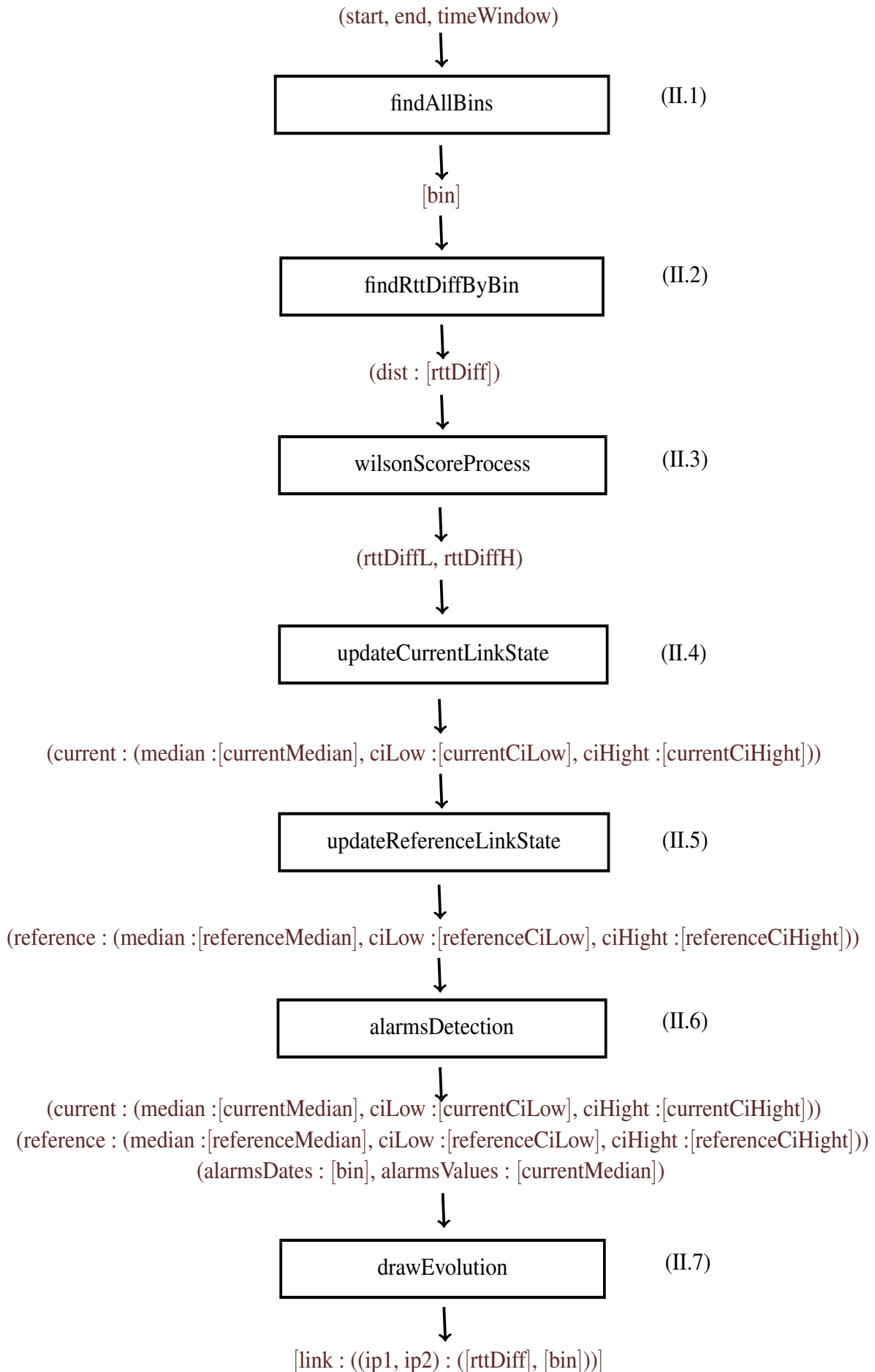


FIGURE II.4 – Phase II : détection des alarmes

## II.4.4 Exemple illustratif des étapes de la détections des alarmes

### Description de l'échantillon :

L'échantillon des traceroutes qui va nous permettre d'illustrer l'algorithme de la détection des alarmes est définie comme suit :

- le nombre total de traceroutes est de 25 ;
- le premier traceroute a été capturé à 1514769800 <sup>11</sup> ;
- le dernier traceroute a été capturé à 1514787809 <sup>12</sup> ;
- la durée d'une période est 1 heure (3600) ;
- les débuts des périodes entre 1514769800 et 1514789809 sont 1514769800, 1514773400, 1514777000, 1514780600, 1514784200, 1514787800 ;

étape	résultats		
(I.1)	1514769800, 1514773400, 1514777000, 1514780600, 1514784200, 1514787800		
(I.2)	id	période	nombre de traceroutes
	1	[1514769800, 1514769800+3600]	5
	2	[1514773400, 1514773400+3600]	4
	3	[1514777000, 1514777000+3600]	4
	4	[1514780600, 1514780600+3600]	4
	5	[1514784200, 1514784200+3600]	4
	6	[1514787800, 1514787800+3600]	4

Une fois les traceroutes sont groupés par période, nous parcourons chaque traceroute afin d'éliminer ceux invalides (étape I.3). Ensuite, nous agrégeons chaque saut par source de signal en calculant leur médiane. Nous donnons un exemple d'un traceroute qui illustre cette agrégation (voir les Listings II.2 et II.3). Le même traitement est appliqué sur les autres traceroutes.

Listing II.2 – Les sauts du traceroute T7 (sans agrégation)

```

1 hop_id : 1, hops : [(from : 89.105.200.57, rtt : 1.955, x : None), (from :
  89.105.200.57, rtt : 1.7, x : None), (from : 89.105.200.57, rtt : 1.709,
  x : None)]
2 hop_id : 2, hops : [(from : 185.147.12.31, rtt : 8.543, x : None), (from :
  185.147.12.31, rtt : 4.103, x : None), (from : 185.147.12.31, rtt : 4.41,
  x : None)]
3 hop_id : 3, hops : [(from : 185.147.12.19, rtt : 4.347, x : None), (from :
  185.147.12.19, rtt : 2.876, x : None), (from : 185.147.12.19, rtt : 3.143
  , x : None )]
```

Listing II.3 – Les sauts du traceroute T7 (après l'agrégation)

```

1 hop_id : 1, hops : [(from : 89.105.200.57, rttAgg : 1.709)]
2 hop_id : 2, hops : [(from : 185.147.12.31, rttAgg : 4.41)]
3 hop_id : 3, hops : [(from : 185.147.12.19, rttAgg : 3.143)]
```

L'inférence des liens du traceroute T7 (étape I.5) fournit les liens suivants :

11. Équivalent à GMT : Monday, January 1, 2018 1 :23 :20 suivant <https://www.epochconverter.com/>, consultée le 04/04/2019.

12. Équivalent à GMT : Monday, January 1, 2018 6 :23 :29 suivant <https://www.epochconverter.com/>, consultée le 04/04/2019.



Listing II.4 – Exemple des liens inférés du traceroute T7

```

1 lien 1 : (link : (185.147.12.31, 89.105.200.57), rttDiff : 2.701)
2 lien 2 : (link : (185.147.12.19, 185.147.12.31), rttDiff : -1.267)

```

A l'étape (I.6), nous ordonnons les adresses IP de chaque lien comme c'est illustré dans le Listing II.5 avec la fonction `sort()`.

Listing II.5 – Illustration de l'ordre des liens

```

1 sort(("185.147.12.31", "89.105.200.57")) = ("185.147.12.31", "89.105.200.57")
2 sort(("89.105.200.57", "185.147.12.31")) = ("185.147.12.31", "89.105.200.57")

```

L'ordonnement à l'étape I.6 est une préparation à la fusion. Nous fusionnons les liens ayant impliqué les mêmes routeurs. Prenons les liens inférés à partir des 5 traceroutes comme c'est montré dans le Listing II.6.

Listing II.6 – Liste des liens possibles inférés via les traceroutes T1, T2, T3, T4 et T5

```

1 T1 (link : (185.147.12.31, 89.105.200.57), rttDiff : 2.463)
2 (link : (185.147.12.19, 185.147.12.31), rttDiff : -1.029)
3
4 T2 (link : (185.147.12.31, 89.105.200.57), rttDiff : 3.463)
5 (link : (185.147.12.19, 185.147.12.31), rttDiff : -2.029)
6
7 T3 (link : (185.147.12.31, 89.105.200.57), rttDiff : 2.991)
8 (link : (185.147.12.19, 185.147.12.31), rttDiff : -1.557)
9
10 T4 (link : (185.147.12.31, 89.105.200.57), rttDiff : 4.631)
11 (link : (185.147.12.19, 185.147.12.31), rttDiff : -3.197)
12
13 T5 (link : (89.105.200.57, 185.147.12.31), rttDiff : -7.75)
14 (link : (185.147.12.19, 89.105.200.57), rttDiff : 3.803)

```

La fusion de ces liens nous donne les trois liens présentés dans le Listing II.7. Chaque lien est caractérisé par les sondes l'ayant capturé, leurs RTTs Différentiels et la période dupliquée, pendant laquelle il a été identifié, en nombre de RTTs différentiels.

Listing II.7 – Caractérisation des liens identifiés lors de la période 1514769800 avec les traceroutes T1, T2, T3, T4 et T5

```

1 [(link : (185.147.12.19, 89.105.200.57),
2 probes : [4247],
3 rttDiffs : [3.803],
4 bins : [1514769800]),
5
6 (link : (185.147.12.19, 185.147.12.31),
7 probes : [4247, 4247, 4247, 4247],
8 rttDiffs : [-1.029, -2.029, -1.557, -3.197],
9 bins : [1514769800, 1514769800, 1514769800, 1514769800]),
10
11 (link : (185.147.12.31, 89.105.200.57),
12 probes : [4247, 4247, 4247, 4247, 4247],
13 rttDiffs : [2.463, 3.463, 2.991, 4.631, -7.75],
14 bins : [1514769800, 1514769800, 1514769800, 1514769800, 1514769800]) ]

```

Après avoir analysé toutes les périodes, nous résumons les liens en fusionnant les données relatives à toutes les périodes, c'est l'objectif de l'étape I.8.

Listing II.8 – Illustration de l'ordre des liens

```

1 [(link      : (185.147.12.19,89.105.200.57),
2 probes     : (4247),
3 rttDiffs   : (3.803),
4 bins       : (1514769800)),
5
6 link       : (185.147.12.19,185.147.12.31),
7 probes     : (4247, 4247, 4247, 4247, 4247, 4247, 4247, 4247, 4247, 4247, 4247, 4247,
8               4247, 4247, 4247, 4247, 4247, 4247, 4247, 4247, 4247, 4247, 4247, 4247,
9               4247),
10 rttDiffs   : (-1.267, -2.407, -1.49, -3.029, -1060.029, -1045.029, -1098.029,
11               -1080.029, -680.029, -845.029, -998.029, -800.029, -1.029, -2.029, -1.5
12               57, -3.197, -1.277, -2.017, -1.257, -2.968, -0.96, -1.967, -0.987, -3.20
13               1),
14 bins       : (1514787800, 1514787800, 1514787800, 1514787800, 1514784200, 1514784
15               200, 1514784200, 1514784200, 1514780600, 1514780600, 1514780600, 1514780
16               600, 1514769800, 1514769800, 1514769800, 1514769800, 1514773400, 1514773
17               400, 1514773400, 1514773400, 1514777000, 1514777000, 1514777000, 1514777
18               000)),
19
20 link       : (185.147.12.31,89.105.200.57),
21 probes     : (4247, 4247, 4247, 4247, 4247, 4247, 4247, 4247, 4247, 4247, 4247, 4247,
22               4247, 4247, 4247, 4247, 4247, 4247, 4247, 4247, 4247, 4247, 4247, 4247,
23               4247, 4247),
24 rttDiffs   : (2.701, 3.841, 2.924, 4.463, 1061.463, 1046.463, 1099.463, 1081.
25               463, 681.463, 846.463, 999.463, 801.463, 2.463, 3.463, 2.991, 4.631, -7.
26               75, 2.711, 3.451, 2.691, 4.402, 2.394, 3.401, 2.421, 4.635),
27 bins       : (1514787800, 1514787800, 1514787800, 1514787800, 1514784200, 1514784
28               200, 1514784200, 1514784200, 1514780600, 1514780600, 1514780600, 1514780
29               600, 1514769800, 1514769800, 1514769800, 1514769800, 1514769800, 1514773
30               400, 1514773400, 1514773400, 1514773400, 1514777000, 1514777000, 1514777
31               000, 1514777000))]

```

A la fin de l'étape I.8, nous disposons de tous les liens possibles caractérisés avec leurs RTTs différentiels. Ainsi, nous pouvons entamer la phase II du processus de la détection.

Dans un premier temps, nous générons les mêmes périodes générées à l'étape I.1. Pour chaque période, nous récupérons les RTTs différentiels identifiés durant celle-ci, et nous obtenons `dist`, cette dernière représente l'ensemble des RTTs différentiels de la période courante, c'est l'objectif de l'étape II.2.

Dans notre exemple, et pour la clarté de ce dernier, une référence est assez représentable après avoir calculé 3 intervalles de confiances.

Les tableaux suivants reprennent les différentes itérations permettant d'analyser notre ensemble de traceroutes (25). Chaque tableau illustre les résultats de l'analyse d'une période, dans l'ordre chronologique de ces périodes.

<b>itération</b>	1
<b>bin</b>	1514769800
<b>lien</b>	(185.147.12.31, 89.105.200.57)
<b>dist</b>	[-7.75, 2.463, 2.991, 3.463, 4.631]
<b>(low, high)</b>	(0.11762077423264793, 0.7692757187239873)
<b>(lo, hi)</b>	(0.5881038711632396, 3.8463785936199364)
<b>(l, h)</b>	(0, 3)
<b>rttDiffL</b>	dist[0] = -7.75
<b>rttDiffH</b>	dist[3] = 3.463
<b>currentMedian</b>	2.991
<b>currentCiLow</b>	2.991 - (-7.75) = 10.741
<b>currentCiHi</b>	3.463 - 2.991 = 0.472
<b>cas 1 ? 2 ? 3 ?</b>	cas 1 (aucun intervalle de confiance calculé)
<b>referenceMedian</b>	2.991
<b>referenceCiLow</b>	-7.75
<b>referenceCiHi</b>	3.463
<b>détection des alarmes ?</b>	non (car cas 3 pas encore atteint)
<b>alarmesDates</b>	[]
<b>alarmesValues</b>	[]

<b>itération</b>	2
<b>bin</b>	1514773400
<b>lien</b>	(185.147.12.31, 89.105.200.57)
<b>dist</b>	[2.691, 2.711, 3.451, 4.402]
<b>(low, high)</b>	(0.15003898915214947, 0.8499610108478506)
<b>(lo, hi)</b>	(0.6001559566085979, 3.3998440433914023)
<b>(l, h)</b>	(0, 3)
<b>rttDiffL</b>	dist[0] = 2.691
<b>rttDiffH</b>	dist[3] = 4.402
<b>currentMedian</b>	3.081
<b>currentCiLow</b>	0.39
<b>currentCiHi</b>	1.321
<b>cas 1 ? 2 ? 3 ?</b>	cas 1 (1 intervalle de confiance calculé)
<b>referenceMedian</b>	3.081
<b>referenceCiLow</b>	2.691
<b>referenceCiHi</b>	4.402
<b>détection des alarmes ?</b>	non (car cas 3 pas encore atteint)
<b>alarmesDates</b>	[]
<b>alarmesValues</b>	[]

<b>itération</b>	3
<b>bin</b>	1514777000
<b>lien</b>	(185.147.12.31, 89.105.200.57)
<b>dist</b>	[2.394, 2.421, 3.401, 4.635]
<b>(low, hight)</b>	(0.15003898915214947, 0.8499610108478506)
<b>(lo, hi)</b>	(0.6001559566085979, 3.3998440433914023)
<b>(l, h)</b>	(0, 3)
<b>rttDiffL</b>	dist[0] = 2.394
<b>rttDiffH</b>	dist[3]= 4.635
<b>currentMedian</b>	2.9109999999999996
<b>currentCiLow</b>	0.5169999999999996
<b>currentCiHi</b>	1.7240000000000004
<b>cas 1 ? 2 ? 3 ?</b>	cas 1 (2 intervalle de confiance calculé)
<b>referenceMedian</b>	2.9109999999999996
<b>referenceCiLow</b>	2.394
<b>referenceCiHi</b>	4.635
<b>détection des alarmes ?</b>	(car cas 3 pas encore atteint)
<b>alarmesDates</b>	[]
<b>alarmesValues</b>	[]

<b>itération</b>	4
<b>bin</b>	1514780600
<b>lien</b>	(185.147.12.31, 89.105.200.57)
<b>dist</b>	[681.463, 801.463, 846.463, 999.463]
<b>(low, hight)</b>	(0.15003898915214947, 0.8499610108478506)
<b>(lo, hi)</b>	(0.6001559566085979, 3.3998440433914023)
<b>(l, h)</b>	(0, 3)
<b>rttDiffL</b>	dist[0] = 681.463
<b>rttDiffH</b>	dist[3]= 999.463
<b>currentMedian</b>	823.963
<b>currentCiLow</b>	142.5
<b>currentCiHi</b>	175.5
<b>cas 1 ? 2 ? 3 ?</b>	cas 2 (3 intervalle de confiance calculé)
<b>referenceMedian</b>	2.991 (car median([2.991, 3.081, 2.9109999999999996])= 2.991)
<b>referenceCiLow</b>	2.394 (car median ([-7.75, 2.691, 2.394])= 2.394)
<b>referenceCiHi</b>	4.402 (car median([3.463, 4.402, 4.635])=4.402)
<b>détection des alarmes ?</b>	non (car cas 3 pas encore atteint)
<b>alarmesDates</b>	[]
<b>alarmesValues</b>	[]

<b>itération</b>	5
<b>bin</b>	1514784200
<b>lien</b>	(185.147.12.31, 89.105.200.57)
<b>dist</b>	[1046.463, 1061.463, 1081.463, 1099.463]
<b>(low, high)</b>	(0.15003898915214947, 0.8499610108478506)
<b>(lo, hi)</b>	(0.6001559566085979, 3.3998440433914023)
<b>(l, h)</b>	(0, 3)
<b>rttDiffL</b>	dist[0] = 1046.463
<b>rttDiffH</b>	dist[3]= 1099.463
<b>currentMedian</b>	1071.463
<b>currentCiLow</b>	25.0
<b>currentCiHi</b>	28.0
<b>cas 1 ? 2 ? 3 ?</b>	cas 3 (car la référence est assez représentable )
<b>referenceMedian</b>	13.67572 (car $0.992.991 + 0.011071.463 = 13.67572$ )
<b>referenceCiLow</b>	12.83469 (car $0.992.394 + 0.011046.463 = 12.83469$ )
<b>referenceCiHi</b>	15.352609999999999 (car $0.994.402 + 0.011099.463 = 15.35261$ )
<b>détection des alarmes ?</b>	oui (car : cas 3)
<b>alarmesDates</b>	[1514784200]
<b>alarmesValues</b>	[1071.463]

<b>itération</b>	6
<b>bin</b>	1514787800
<b>lien</b>	(185.147.12.31, 89.105.200.57)
<b>dist</b>	[2.701, 2.924, 3.841, 4.463]
<b>(low, high)</b>	(0.15003898915214947, 0.8499610108478506)
<b>(lo, hi)</b>	(0.6001559566085979, 3.3998440433914023)
<b>(l, h)</b>	(0, 3)
<b>rttDiffL</b>	dist[0] = 2.701
<b>rttDiffH</b>	dist[3]= 4.463
<b>currentMedian</b>	3.3825000000000003
<b>currentCiLow</b>	0.6815000000000003
<b>currentCiHi</b>	15.243713899999998
<b>cas 1 ? 2 ? 3 ?</b>	cas 3 (car la référence est assez représentable)
<b>referenceMedian</b>	13.5727878
<b>referenceCiLow</b>	12.7333531
<b>referenceCiHi</b>	15.352609999999999
<b>détection des alarmes ?</b>	oui (car : cas 3)
<b>alarmesDates</b>	[1514784200, 1514787800]
<b>alarmesValues</b>	[1071.463, 3.3825000000000003]

# Chapitre III

## Introduction au Big Data

### III.1 Introduction

Big Data est un terme associé aux données massives, rapidement générées, ayant une grande diversité que les outils traditionnels sont incapables de gérer ces données. La complexité du Big Data est due au fait que tout type de données peut être utilisé, et ce en vue de livrer la bonne information à la bonne personne et au bon moment dans le but d'aider à prendre les bonnes décisions. Dans ce chapitre, nous présentons brièvement un des processus d'analyse de données. Ensuite, nous décrivons quelques concepts impliqués dans un processus d'analyse des données massives. Enfin, nous parcourons un ensemble de technologies utilisées dans l'analyse des données à grande échelle.

### III.2 Processus d'analyse de données massives

Les étapes d'un processus d'analyse de données, à grande échelle ou non, sont différentes selon le processus adopté. Ce dernier peut impliquer plusieurs besoins et concepts. En particulier, on note le besoin du stockage de données massives, du traitement de données massives et le besoin de la visualisation des résultats des traitements appliqués. Le choix du traitement à appliquer sur les données est dirigé par les objectifs de l'analyse menée. Par exemple, une analyse peut envisager la création d'un système de recommandations dans les sites Internet, la conception d'un outil de prédiction basé sur les données historiques, un système de suivi en temps réel, etc. Ces applications appuient sur des algorithmes basés sur les mathématiques, en particulier les mathématiques statistiques et probabilistes. Afin d'assurer l'efficacité de l'analyse de données massives, la coopération de plusieurs ressources accélère considérablement les étapes de l'analyse, c'est ce que l'informatique distribuée a apporté.

#### Exemple d'un processus d'analyse de données

Le processus d'analyse de données passe généralement par des étapes classiques. A ces étapes peuvent s'ajouter d'autres étapes intermédiaires ou supplémentaires. Pour précision, non seulement ces étapes peuvent s'appliquer dans le cas où les données sont volumineuses, mais aussi dans le cas des données manipulées par des outils traditionnels comme les bases de données relationnelles. IBM Knowledge Center<sup>1</sup> a résumé le processus de l'analyse de données

---

1. Source : [https://www.ibm.com/support/knowledgecenter/fr/SSEPGG\\_9.5.0/com.ibm.im.easy.doc/c\\_dm\\_process.html](https://www.ibm.com/support/knowledgecenter/fr/SSEPGG_9.5.0/com.ibm.im.easy.doc/c_dm_process.html), consultée le 06/08/2018.

dans les étapes suivantes : définition de problème, exploration de données, préparation de données, modélisation, évaluation et enfin l'étape de déploiement.

**Définition de problème** C'est le point d'entrée vers tout projet d'analyse de données. Les objectifs doivent être clairs. A la fin de cette étape, on connaît les objectifs de l'analyse, mais pas forcément comment cette analyse va être menée en terme de technologies utilisées, les méthodes statistiques, etc.

**Exploration de données** Ce que caractérise cette étape, c'est la découverte de la nature des données, les sources des données, la qualité de données, etc.

**Préparation de données** C'est durant cette phase que se passent les opérations d'ETL (Extract-transform-load), ce sont les opérations de chargement, nettoyage et de la transformation des données. Durant cette étape, que des tentatives sont faites pour créer le schéma de données. Dans certains cas, cela amène à chercher des sources complémentaires de données ou bien de nouvelles transformations à appliquer sur les données.

**Modélisation** A l'issue de cette phase, un modèle de qualité est créé. Ce modèle doit répondre aux objectifs précédemment établis. dans l'étape de définition du problème. Cette étape implique des principes en statistiques, probabilités, *machine learning*, etc.

**Evaluation** C'est à cette étape qu'on valide le modèle créé à l'étape de Modélisation. L'évaluation et la validation du modèle prennent en compte les objectifs prédéfinis ; si le modèle répond aux attentes précédemment exprimées.

**Déploiement** C'est l'utilisation des outils existants ou la mise place d'une solution permettant d'exploiter les résultats de l'analyse. Par exemple, la visualisation des résultats de l'analyse sous forme d'un tableau d'indicateurs sur une page Web.

## III.3 Quelques concepts associés au Big Data

Etant donné que le domaine du Big Data implique plusieurs concepts, nous présentons une liste de concepts non exhaustive.

### III.3.1 Définition du Big Data : Volume, Vitesse, Variété et Vérité

IBM définit le Big Data suivant les quatre dimensions suivants : volume, variété, vitesse et vérité.

**Volume de données** La quantité de données manipulées par les outils traditionnels de la gestion des données est de l'ordre de Gigaoctets (GB) et de Téraoctets (TB). Toutefois, le Big Data est mesuré en Pétaoctets (PB), Exaoctets (EB), voire plus. Une des premières applications du Big Data est la recherche dans Word-Wide Web (WWW). Selon une étude ([33], 2013) de l'International Data Corporation (IDC), le volume de données va atteindre 40 Zettaoctets<sup>2</sup> par entreprise en 2020.

**Vitesse de données** le Big Data est généré par des milliards d'appareils. Les données générées sont communiquées avec la vitesse de la lumière via l'Internet. L'augmentation de la vitesse de l'Internet est une des raisons ayant contribué à l'augmentation de la vitesse de la génération de données. Par exemple, Walmart (international discount retail chain) génère environ

---

2. 1 ZB = 1,000,000,000,000 GB.

2.5 Pétaoctets de données chaque heure via les transactions de ses consommateurs<sup>3</sup>.

**Variété de données** Le Big Data inclut toutes les formes de données, des fonctions diversifiées des données et des sources variées des données.

Le premier aspect de la variété des données massives est la **forme** de celles-ci. Les données manipulées incluent du texte, des graphes, des cartes, des vidéos, des photos, etc.

Le deuxième aspect de la variété des données massives concerne les **fonctions** assurées par ces données. Des données sont issues des conversations humaines, d'autres des transactions des consommateurs, ou bien des données archivées, etc.

Les **sources** du Big Data est le troisième aspect de variété. Des données sont en provenance des téléphones mobiles, des tablettes ou des ordinateurs portables, des fichiers journaux, des réseaux de capteurs, etc. Les sources du Big Data peuvent être classées en trois grandes catégories : communications *human to human* comme les conversations échangées dans les réseaux sociaux, communications *human to machine* comme l'accès des utilisateurs aux données dans le web et enfin communications *machine to machine* comme les données issues de la communication entre les capteurs dans un réseau de capteurs.

**Véracité de données** La véracité concerne la crédibilité et la qualité de données. La mauvaise qualité de données est due à de nombreuses raisons, telles que les pannes techniques comme le dysfonctionnement des appareils comme les capteurs, les erreurs humaines, etc. De plus, les données peuvent être intentionnellement erronées pour des raisons de concurrence ou des raisons stratégiques.

### III.3.2 L'architecture standard du Big Data

L'architecture standard du Big Data présentée dans ce qui suit est celle proposée dans [30]. Cette architecture est composée des couches suivantes : *Data sources*, *Data Ingest*, *Batch processing*, *Stream Processing*, *Data organizing*, *Infrastructure*, *Distributed File System* et enfin la couche *Data consumption*. La Figure III.1 reprend l'organisation des différentes couches de cette architecture.

Dans un premier temps, les données sont accueillies via la couche *ingest system* par diverses sources de données. Ensuite, les données sont traitées dans deux modes différents : *stream processing* et *batch processing*. Les résultats de ce traitement peuvent être envoyés vers les bases de données NoSQL (III.3.3) pour une utilisation ultérieure, ou bien utiliser ces résultats comme entrées pour d'autres applications. Une solution Big Data comprend typiquement ces couches logiques. Chaque couche peut être représentée par une ou plusieurs technologies disponibles. Reprenons chaque couche logique :

**Data sources layer** Le choix des sources de données pour une application donnée dépend des objectifs qui dirigent l'analyse en question. Les sources avec leurs différents aspects sont détaillées dans la section III.3.1.

**Data Ingest layer** Cette couche permet de récupérer les données depuis les différentes sources de données. Les données sont accueillies à travers des points d'entrées multiples. Ces points sont capables de recevoir ces données ayant une vitesse variable ainsi qu'une quantité aussi

---

3. Source : <https://www.bernardmarr.com/default.asp?contentID=690>, consultée le 30/06/2018.



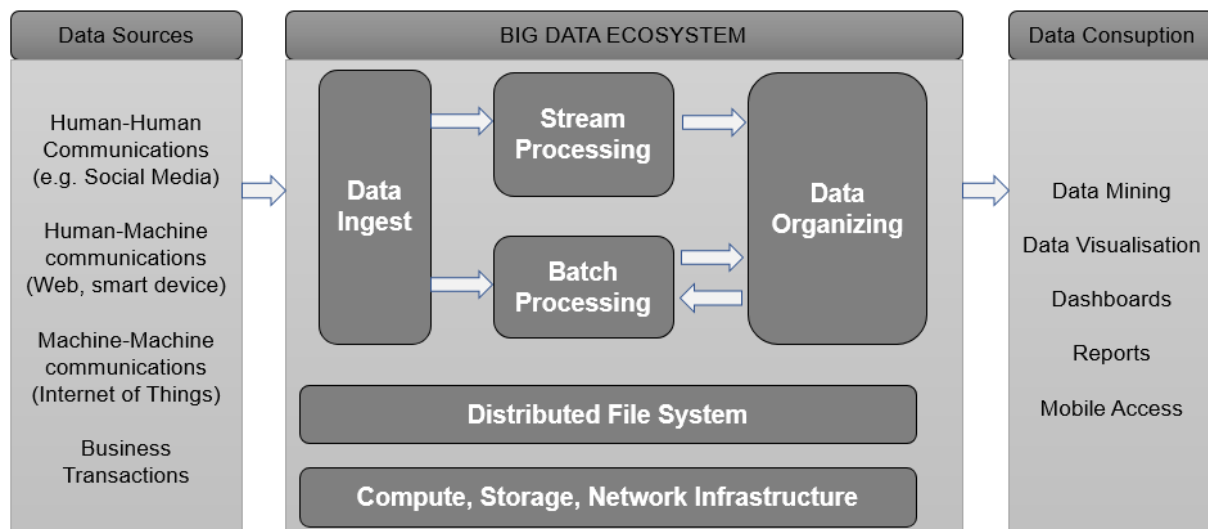


FIGURE III.1 – Architecture standard du Big Data [30]

variable. Après avoir traversé la couche *Data Ingest*, les données sont envoyées au *batch processing system*, au *stream processing system*, ou bien à un système de stockage particulier.

**Batch processing layer** Les données reçues sur cette couche sont celles en provenance du *Data Ingest* ou bien d'une des bases de données NoSQL. Ces données sont ensuite traitées, par exemple, en utilisant les techniques de la programmation parallèle en vue de fournir les résultats souhaités dans un temps raisonnable. La présente couche doit avoir connaissance des sources de données, des types de données, des algorithmes qui vont travailler sur ces données et enfin des résultats souhaités. Les résultats des traitements peuvent être utilisés par une des applications ou bien être sauvegardés dans une des bases de données adaptées.

**Stream Processing layer** Cette couche approvisionne les données directement d'une des entrées du *Data Ingest layer* ; c'est ce qui différencie cette couche de la couche Batch processing layer. En revanche, *Stream Processing* est similaire à la couche *Batch processing* en matière des techniques de la programmation parallèle utilisées ainsi que la nécessité d'avoir les détails sur les sources des données, les types de données et les résultats souhaités.

**Data organizing layer** Le rôle de cette couche est d'organiser les données en provenance de la couche *Stream Processing* et de la couche *Batch processing*. Cette couche est représentée par les bases de données NoSQL.

**Infrastructure layer** Cette composante est responsable de la gestion des ressources de stockage, des ressources du calcul et de la gestion de la communication. Les fonctionnalités de cette couche sont typiquement fournies à travers le cloud computing.

**Distributed File System layer** Cette couche permet de stocker une grande quantité de données, de sorte que ces données soient rapidement et facilement accessibles à toutes les couches qui forment un système Big Data. C'est ce qu'assure, par exemple, Hadoop Distributed File System (HDFS).

**Data consumption layer** Cette dernière couche utilise les résultats obtenus par les couches de l'analyse. Les résultats fournis peuvent être exprimés avec des rapports, des dashboards, des visualisations, un moteur de recommandation ou tout autre format.

### III.3.3 Les bases de données NoSQL (Not Only SQL)

#### Introduction

Au cours de ces dernières années, on constate une révolution dans le stockage de données non structurées ayant une taille importante. De plus, les objets à sauvegarder sont complexes ; ils sont issus de sources hétérogènes. Cette complexité a mis en question les performances des bases de données relationnelles.

Le terme NoSQL est apparu pour la première fois en 1998. Carlo Strozzi [14] a parlé des bases de données relationnelles qui n'utilisent pas le SQL comme langage d'interrogation des tables. Des années plus tard, des solutions open source basées sur ce concept ont vu le jour.

Les bases de données relationnelles sont conçues pour gérer les données structurées, optimisées pour offrir la précision et la cohérence de données. De plus, elles sont utilisées par la majorité des entreprises pour plusieurs raisons comme la facilité d'utilisation, la disponibilité de plusieurs produits et développeurs, etc. Ces dernières années, avec l'augmentation exponentielle de la quantité de données générées par certaines entreprises, ces dernières ont constaté l'insuffisance des Systèmes de Gestion de Bases de Données Relationnelles (SGBDR) pour répondre à leurs besoins.

Les bases de données NoSQL sont conçues pour gérer des volumes de données importants. Le flux ainsi que la structure de ces données sont imprévisibles. C'est pourquoi les bases de données relationnelles ne sont pas convenables. L'idée de ces bases de données NoSQL, c'est d'abord assurer la capacité de stocker des données à grande échelle dont la quantité évolue rapidement, voire exponentiellement. En deuxième lieu, les données stockées doivent être interrogées avec efficacité. Les données stockées dans les bases de données NoSQL n'obéissent pas à un modèle prédéfini comme c'est le cas pour les bases de données relationnelles. Cette flexibilité est une des caractéristiques des bases de données NoSQL.

#### Types de base de données NoSQL

Il existe quatre catégories distinctes de bases de données NoSQL. Chaque catégorie répond à des besoins particuliers. On distingue les bases de données clé-valeur, document, graphe et colonne.

**Clé-valeur** Une base de données de type clé-valeur repose sur le paradigme clé-valeur ; chaque donnée, que ce soit un nombre, du texte ou tout autre type est associé à une clé unique. Cette clé est le seul moyen d'accéder aux données stockées. Dans les bases de données NoSQL de type clé-valeur, les enregistrements n'adhèrent pas à une structure prédéfinie. Par exemple, on peut avoir le premier enregistrement de type entier et le deuxième enregistrement de type texte. Cela assure une forte évolutivité grâce à l'absence d'une structure ou de typage. La Figure III.2 reprend un exemple d'une base de données NoSQL de type clé-valeur.

**Document** Une base de données NoSQL de type document permet de stocker les données en reposant sur le paradigme clé-valeur. Toutefois, les valeurs stockées sont complexes, il s'agit de documents de type JSON, XML, etc. L'accès aux données d'un enregistrement peut se faire de manière hiérarchique. La possibilité de stocker des objets complexes et hétérogènes est un des points forts des bases de données NoSQL de type document. Un exemple est fourni dans

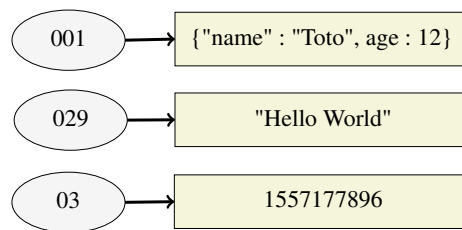


FIGURE III.2 – Illustration d’une base de données NoSQL de type clé-valeur

la Figure III.3. Une des différences majeures entre les bases de données clé-valeur et celles de type document c’est que pour les premières, l’indexation est au niveau des clés seulement, tandis que pour les deuxièmes, l’indexation est au niveau clé et valeur. En pratique, pour les bases de données clé-valeur, les données sont récupérées en se basant sur la clé et pour les autres, la récupération d’un enregistrement peut être basée sur la clé et sur la valeur étant donné que la valeur est semi-structuré (valeur de type JSON, XML, etc.)

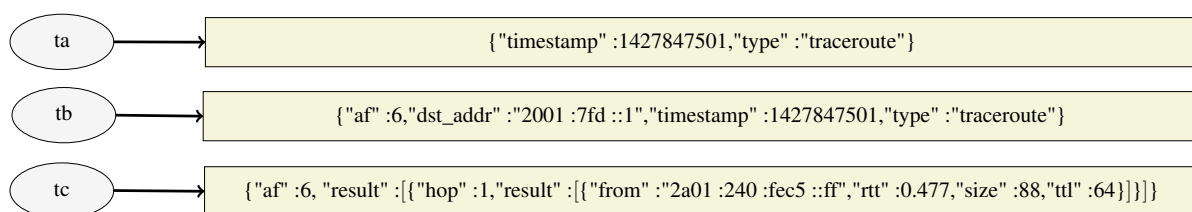


FIGURE III.3 – Illustration d’une base de données NoSQL de type document

**Colonnes** Dans les bases de données traditionnelles, les données sont stockées sur des lignes. Dans le cas d’une base NoSQL orientée colonne, les données sont stockées par colonne. L’interrogation de ce type de bases travaille sur une colonne particulière sans devoir passer par les autres colonnes comme dans les bases de données relationnelles classiques. Une base de données de type colonne est adaptée pour les requêtes analytiques comme les requêtes d’agrégation (moyennes, maximum, etc). La Figure III.4 illustre la différence entre le stockage dans une base de données relationnelle et une base de données orientée colonnes<sup>4</sup>.

**Graphe** Dans une base de données de type graphe, les données stockées sont les nœuds, les liens et les propriétés sur les nœuds et sur les liens. Un exemple de base de données NoSQL de type graphe est le réseau social ; chaque entité représente une personne et les relations entre ces personnes peuvent prendre plusieurs formes. Un autre exemple de données stockées dans une base orientée graphe est donné dans la Figure III.5. Il existe plusieurs implémentations des quatre types de bases de données NoSQL. Chaque implémentation favorise un ou plus des éléments suivants : la disponibilité des données, la cohérence des données et la tolérance au partitionnement. C’est ce qu’explique le théorème CAP.

4. Cette illustration est basée sur une figure disponible sur <https://www.illustradata.com/bases-nosql-orientees-colonnes-quest-cest/>.

### Le stockage dans une base de données relationnelles

Clé	Date début validité	Date fin validité	Nom	Prénom	Csp
A	01/01/2010	02/02/2015	Dupont	Null	Cadre
A	03/02/2015	Null	Cap	Null	Cadre
B	01/01/2010	Null	Null	Joséphine	Null
C	01/01/2010	03/03/2016	Black	George	Cadre
C	04/04/2016	Null	Black	George	Retraité

### VS dans les bases orientées colonnes

A	Nom : Dupont DATE : 01/01/2010 Nom : Cap DATE : 03/02/2015	Csp Cadre DATE : 01/01/2010
B	Prénom : Joséphine DATE : 01/01/2010	
C	Nom : Black DATE : 01/01/2010 Prénom : George DATE : 01/01/2010	Csp Cadre DATE : 01/01/2010 Csp Retraité DATE : 01/01/2010

FIGURE III.4 – Illustration d’une base de données NoSQL de type colonne

### Big Data et le théorème CAP :

Dans le but d’assurer un traitement rapide de données à grande échelle, ces dernières sont répartie sur un nombre de machines (ou nœuds). Le théorème CAP annonce que dans le cadre d’un système distribué où les données sont réparties sur plusieurs machines, une base de données ne peut pas garantir les trois attributs suivants : *Consistency*, *Availability*, et *Partition Tolerance* en même temps.

**Consistency (ou intégrité)** Chaque donnée a un seul état visible depuis l’extérieur. Par exemple, les différents serveurs hébergeant la base de données voient tous les mêmes données. C’est pourquoi une lecture faite après une écriture doit renvoyer la donnée précédemment écrite.

**Availability (ou disponibilité)** Une base de données doit toujours fournir une réponse à une requête d’un client.

**Partition tolerance (ou la tolérance au partitionnement)** Une coupure du réseau entre deux nœuds ou l’indisponibilité d’un de ces nœuds ne devrait pas affecter le bon fonctionnement du système. Tout de même, ce dernier doit répondre à la demande d’un client.

Les trois attributs du théorème CAP s’opposent entre eux. On distingue les trois scénarios possibles :

- Le couple **CA** : les SGBDR adoptent les deux attributs C et A, qui sont une forte cohérence et disponibilité. Cependant, l’attribut partitionnement réseau n’est pas toujours pris en compte.
- Le couple **CP** : les implémentations du C et du P assurent la tolérance aux pannes en distribuant les données sur plusieurs serveurs. Malgré cette réplication, ces implémentations assurent la cohérence des données même en présence de mises à jour concurrentielles.

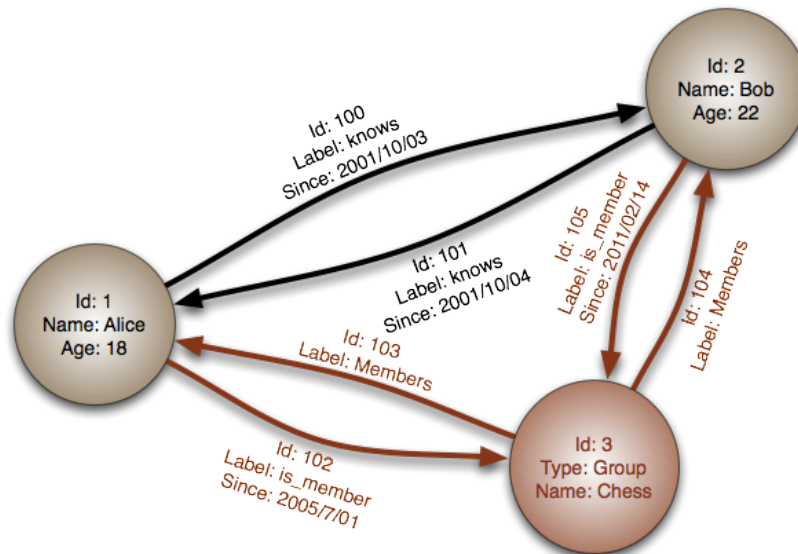


FIGURE III.5 – Illustration d’une base de données NoSQL de type graphe

- Le couple **AP** : les implémentations du A et du P assurent un temps de réponse rapide et une réplication des données. Cependant, les mises à jour étant asynchrones, la garantie que la version d’une donnée soit bonne, ne peut pas être assurée.

La Figure III.6 présente des implémentations des différents types de bases de données NoSQL pour chaque couple CA, CP et AP.

Le choix d’une base de données relationnelle ou NoSQL dépend des besoins des entreprises. En terme de tendances, la Figure III.7 reprend un classement des SGBDs au 1 août 2018. La suite de la liste ainsi que la méthode qui dirige ce classement sont disponibles sur le site web *DB-Engines Ranking*<sup>5</sup>. Parmi les critères du classement, on trouve le nombre de références du SGBD sur les sites Internet.

### III.3.4 Extraction, Transformation, Loading (ETL)

Dans une même organisation, il est possible d’avoir plusieurs sources de données : des bases de données relationnelles, des bases de données NoSQL, des fichiers de données de type Excel, etc. Dans le cas où une analyse de données devrait impliquer des données en provenance de sources de données hétérogènes, il est nécessaire de faire appel aux opérations ETL :

**Extraction** : la diversité des sources de données implique des formats de données différents. Généralement, ce sont des données en provenance des bases de données relationnelles, des fichiers plats, des bases de données non relationnelles, etc. En général, le but de la phase d’extraction est de convertir les données en un seul format approprié à l’étape de la transformation. Cette phase vérifie si les données respectent un modèle ou une structure attendu. Si ce n’est pas le cas, les données peuvent être totalement ou partiellement rejetées.

**Transformation** : cette étape s’applique sur les données extraites. Elle comprend une série de règles ou de fonctions. Ces derniers sont appliqués sur les données avant de les envoyer vers la cible. Certaines sources de données nécessiteront peu de transformations, voire aucune. Dans d’autres cas, un ou plusieurs types de transformation sont nécessaires. Quelques exemples de transformations dans la liste ci-dessous :

5. Source : <https://db-engines.com/>, consultée le 01/08/2018.

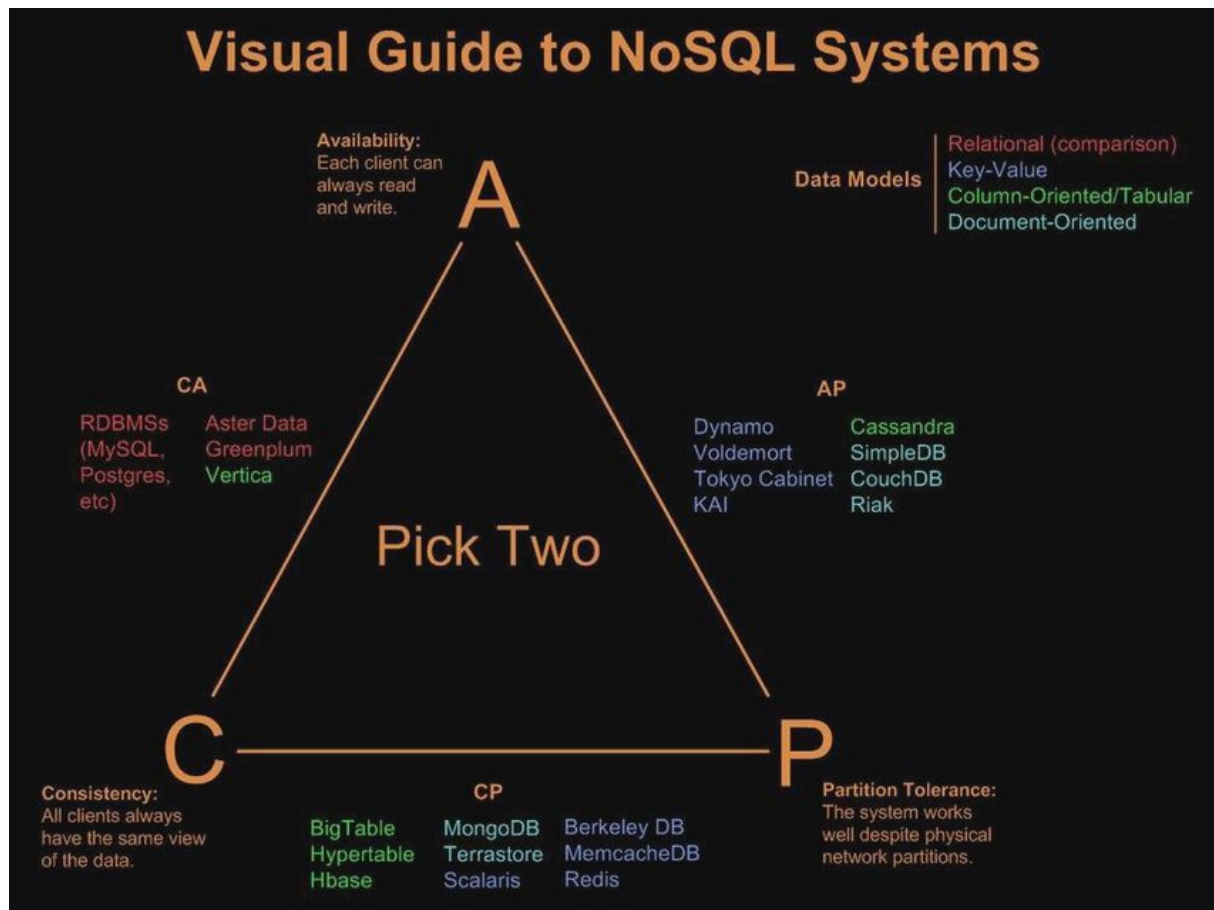


FIGURE III.6 – Bases de données NoSQL suivant le théorème de CAP

Source : [https://www.researchgate.net/figure/CAP-theorem-concept-5-II-WHY-YOU-NEED-NOSQL-The-first-reason-to-use-NoSQL-is-because\\_fig2\\_323309389](https://www.researchgate.net/figure/CAP-theorem-concept-5-II-WHY-YOU-NEED-NOSQL-The-first-reason-to-use-NoSQL-is-because_fig2_323309389), consultée le 05/08/2018.

- sélection de seulement un nombre de colonnes à charger ;
- adaptation des codes, c'est le cas où le système de stockage source utilise 1 pour dénoter un homme et l'entrepôt de données utilise "H" ;
- conversion des devises, c'est le cas par exemple où les salaires sont exprimés en une devise différente de l'euro ;
- élimination des doublons ;
- etc.

**Loading** : l'objectif de cette étape est de charger les données transformées vers l'entrepôt de données. Le chargement des données dépend des besoins de l'organisation. Dans certains cas, on prévoit le remplacement des informations existantes par des informations cumulative plus récentes. Dans d'autre cas, les nouvelles informations sont ajoutées dans la suite de celles existantes.

En ce qui concerne la fréquence des opérations ETL, elles peuvent être planifiées de manière horaire, quotidienne, hebdomadaire, mensuelle ou autre.



343 systems in ranking, August 2018

Rank			DBMS	Database Model	Score		
Aug 2018	Jul 2018	Aug 2017			Aug 2018	Jul 2018	Aug 2017
1.	1.	1.	Oracle +	Relational DBMS	1312.02	+34.24	-55.85
2.	2.	2.	MySQL +	Relational DBMS	1206.81	+10.74	-133.49
3.	3.	3.	Microsoft SQL Server +	Relational DBMS	1072.65	+19.24	-152.82
4.	4.	4.	PostgreSQL +	Relational DBMS	417.50	+11.69	+47.74
5.	5.	5.	MongoDB +	Document store	350.98	+0.65	+20.48
6.	6.	6.	DB2 +	Relational DBMS	181.84	-4.36	-15.62
7.	7.	↑ 9.	Redis +	Key-value store	138.58	-1.34	+16.68
8.	8.	↑ 10.	Elasticsearch +	Search engine	138.12	+1.90	+20.47
9.	9.	↓ 7.	Microsoft Access	Relational DBMS	129.10	-3.48	+2.07
10.	10.	↓ 8.	Cassandra +	Wide column store	119.58	-1.48	-7.14
11.	11.	11.	SQLite +	Relational DBMS	113.73	-1.55	+2.88
12.	12.	12.	Teradata +	Relational DBMS	77.41	-0.82	-1.83
13.	13.	↑ 16.	Splunk	Search engine	70.49	+1.26	+9.03
14.	14.	↑ 18.	MariaDB +	Relational DBMS	68.29	+0.78	+13.60
15.	↑ 16.	↓ 13.	Solr	Search engine	61.90	+0.38	-5.06
16.	↓ 15.	↓ 14.	SAP Adaptive Server +	Relational DBMS	60.44	-1.68	-6.48
17.	17.	↓ 15.	HBase +	Wide column store	58.80	-1.97	-4.72
18.	18.	↑ 20.	Hive +	Relational DBMS	57.94	+0.32	+10.64
19.	19.	↓ 17.	FileMaker	Relational DBMS	56.05	-0.33	-3.60
20.	20.	↓ 19.	SAP HANA +	Relational DBMS	51.93	+0.33	+3.96

FIGURE III.7 – Un classement des SGBDs sur *DB-Engines Ranking* du 1 août 2018Source : <https://db-engines.com/en/ranking>, consultée le 01/08/2018.

### III.3.5 Schema on Write VS Schema on Read

Lors du chargement des données depuis leurs sources de stockage, on distingue deux approches : *Schema on Write* et *Schema on Read*.

Dans la première, il faut définir les colonnes, le format de données, les types, etc. La lecture des données est rapide et moins coûteuse étant donné l'effort entrepris pour définir la structure. C'est le cas des bases de données relationnelles.

Dans la deuxième, les données sont chargées telles qu'elles sont, sans transformations ou changements. L'interprétation de ces données se fait lors de la lecture, et cela dépend des besoins pour lesquels les données sont analysées. Ainsi, les mêmes données peuvent être lues de différentes manières. Par exemple, l'action de lire les données d'une colonne, qu'elles soient de type entier ou bien chaîne de caractère d'un fichier CSV est la même, c'est le type de la donnée qui diffère. C'est l'approche utilisée par Amazon Athena (voir III.4.3).

Afin de montrer la différence entre des deux approches, nous comparons Hadoop avec les bases de données traditionnelles.

1. Créer le schéma :

```
CREATE TABLE Traceroutes (
    id INT,
    dst_name VARCHAR(30),
    ...
)
```

2. Ajouter les données depuis le fichier *traceroute-2019-04-08T0000.json*, nous savons que chaque ligne du fichier a la même structure que celle créée à l'étape 1 :

```
BULK INSERT Traceroutes
FROM 'c:\traceroutes\traceroute-2019-04-08T0000.json'
WITH ROWTERMINATOR = '\n'
```

3. Interroger les données : à l'issue de l'étape 2, les données sont chargées et prêtes à l'interrogation :

```
SELECT dst_name FROM Traceroutes
```

En SQL, on ne peut pas ajouter des données avant de créer le schéma dirigeant ces données. De plus, la création du schéma nécessite la compréhension exhaustive des données. Car en cas d'un changement du contenu du fichier, en terme de structure, la table créée doit être supprimée, mise à jour, ensuite recharger à nouveau les données. L'implication de la mise à jour du schéma peut être coûteuse, en terme de temps si par exemple le volume de donnée est important, de l'ordre de plusieurs centaines de téraoctets, la mise à jour des relations avec d'autres tables, etc.

En ce qui concerne l'approche *Schema on Read*, nous prenons un exemple de la technologie AWS Athena présentée en détail dans la section III.4.3. En utilisant cette technologie, on crée un schéma suivant les besoins de l'analyse de ces données et non pas pour que la structure des données est exactement celle à créer, sauf si cela fait partie des besoins. Par exemple, on peut créer une table dont les colonnes correspondent seulement à la moitié des colonnes possibles. Un autre exemple concerne l'utilisation des conditions ; on crée une table et durant le chargement des données, on ne charge que celles vérifiant une condition donnée. Pour AWS Athena et les autres technologies qui fonctionnent de cette approche, les données sont chargées au moment de l'utilisation et avec plus de flexibilité.

La meilleure approche dépend des besoins de l'analyse. La première approche est meilleure en performances, en revanche, la deuxième est tolérante aux erreurs humaines.

### III.3.6 L'informatique distribuée et l'analyse de données massives

Il existe deux stratégies pour appliquer des traitements sur un grand ensemble de données :

- Par distribution des traitements (*scaling* des traitements) : les traitements sont distribués sur un nombre de nœuds important. De ce fait, les données sont amenées jusqu'à ces nœuds ;
- Par distribution des données (*scaling* des données) : les données sont distribuées sur un nombre important de nœuds. Par ailleurs cela permet de stocker un maximum de données. Il s'agit d'amener les traitements aux machines sur lesquelles les données sont stockées. Du fait que le stockage de données est réparti sur plusieurs machines, il est possible de traiter des données très volumineuses en un temps optimal. La première mise en œuvre de cette approche est le schéma Map-Reduce.

*MapReduce est un patron d'architecture de développement informatique, inventé par Google, dans lequel sont effectués des calculs parallèles, et souvent distribués, de données potentiellement très volumineuses, typiquement supérieures en taille à 1 téraoctet<sup>a</sup>.*

a. Source : <https://fr.wikipedia.org/wiki/MapReduce>, consultée le 20/12/2018.



## III.4 Parcours de quelques technologies du Big Data

La liste des technologies dans le domaine du Big Data est en expansion continue pour répondre au mieux aux besoins de l'analyse de données massives. C'est pourquoi nous allons parcourir une liste non exhaustive des technologies liées au Big Data. En particulier, ce sont les technologies expérimentées pour analyser les traceroutes en provenance du RIPE Atlas.

### III.4.1 MongoDB

MongoDB<sup>6</sup> est une base de données NoSQL de type Document<sup>7</sup>. MongoDB est classé parmi les SGBDs adoptant le couple CP (Consistency et Partition Tolerance) dans le théorème CAP<sup>8</sup>. Une base de données créée dans MongoDB est un ensemble de collections. Une collection dans MongoDB est équivalente à une table dans un SGBDR.

En 2016, MongoDB devient disponible en mode cloud sous le nom MongoDB Atlas<sup>9</sup>. Il est distribué à travers les trois fournisseurs du cloud : Amazon Web Services (AWS), Google Cloud Platform et Microsoft Azure. En terme de tarifs, plusieurs formules sont proposées<sup>10</sup>, y inclut l'offre gratuite pour expérimenter MongoDB Atlas. Les frais d'utilisation du service MongoDB Atlas dépendent du stockage, de la RAM allouée et des options choisies. Les documents sont stockés dans MongoDB sous format BSON.

BSON (ou Binary JSON) est un format utilisé pour stocker et transférer les données dans la base de données MongoDB. BSON facilite la représentation des structures de données simples et des tableaux associatifs<sup>a</sup>.

a. Source : <https://fr.wikipedia.org/wiki/BSON>, consultée le 02/08/2018.

### III.4.2 Amazon DynamoDB

Amazon DynamoDB<sup>11</sup> est une base de données NoSQL de type clé-valeur distribuée, gérée par les services d'Amazon. Elle est capable de stocker un volume important de données limité par la capacité de l'infrastructure d'AWS. Amazon DynamoDB est un service simple et facile à utiliser, il ne nécessite aucune configuration préalable.

Amazon DynamoDB est une base de données évolutive permettant à l'utilisateur final de passer à l'échelle facilement et rapidement. Cette technologie offre des performances constantes à une échelle essentiellement infinie, limitée uniquement par la taille physique du cloud AWS. Elle est flexible. Aucun schéma n'est requis pour stocker les données. Les frais d'utilisation de ce service dépendent de trois éléments<sup>12</sup> :

- la quantité de données stockées : DynamoDB est facturé par Go d'espace disque utilisé (0, 250 USD par Go par mois) ;
- la capacité en lecture par seconde (0, 470 USD par unité de capacité d'écriture par mois) ;

6. Source : <https://www.mongodb.com/>, consultée le 02/08/2018.

7. Une base de données NoSQL de type document est décrite dans la section III.3.3.

8. Le théorème CAP est décrit dans la section III.3.3

9. Source : <https://www.mongodb.com/cloud/atlas>, consultée le 02/08/2018.

10. Source : <https://www.mongodb.com/cloud/atlas/pricing>, consultée le 02/08/2018.

11. Source : <https://aws.amazon.com/fr/dynamodb/>, consultée le 02/05/2018.

12. Source : <https://aws.amazon.com/fr/dynamodb/pricing/>, consultée 02/05/2018.

- la capacité en écriture par seconde (0,090 USD par unité de capacité de lecture par mois) ;

### III.4.3 Amazon S3, Amazon Glue et Amazon Athena

La combinaison d'Amazon S3, Amazon Glue et Amazon Athena permet de créer un environnement Big Data capable d'assurer respectivement le stockage de données, le chargement de données et l'interrogation de données.

**Amazon S3**<sup>13</sup> est un service de stockage d'objets dans le cloud. Il est conçu pour stocker et récupérer toute quantité de données. Il peut assurer 99,99999999 % de durabilité. La sécurité et l'accès aux données sont assurés. Il existe plusieurs classes de stockage qui répondent aux différents besoins.

Dans Amazon S3, le fichier à stocker est considéré comme objet. Un objet est référencé par une clé qui reprend d'abord le chemin vers un pseudo répertoire suivi par le nom de l'objet. Le terme pseudo répertoire est utilisé car, en réalité, Amazon S3 ne stocke pas les objets dans des dossiers comme le cas d'un système d'exploitation. Chaque objet appartient à un compartiment, un compartiment appartient aussi à une des régions d'Amazon et le nom d'un compartiment est unique. Prenons l'exemple d'un compartiment *foo*, contenant deux objets ayant respectivement les clés *A/b/c/i.txt* et *A/b/d/k.txt*, dans ce cas, ces deux objets ne partagent que le même compartiment. Le Tableau III.1 décrit les tarifs de la formule standard relative à un mois.

Région	UE (Irlande)
<b>Première tranche de 50 To</b>	0,023 USD par Go
<b>450 To suivants</b>	0,022 USD par Go
<b>Plus de 500 To</b>	0,021 USD par Go

TABLE III.1 – Les tarifs du AWS S3 (formule Stockage standard S3)

Source : <https://aws.amazon.com/fr/s3/pricing/>, consultée le 05/08/2018.

**Amazon Glue**<sup>14</sup> est un service d'extraction, de transformation et de chargement. L'objectif de ce service est de découvrir les données, les transformer et les rendre accessibles à la recherche et à l'interrogation. Amazon Glue est utile pour la construction des entrepôts de données ; il découvre les métadonnées relatives aux magasins de données et les rend accessibles dans un catalogue central. En prenant en entrée les données présentes dans un compartiment dans Amazon S3, Amazon Glue découvre le schéma de ces données. Il dispose de plusieurs classificateurs intégrés pour la découverte des données. Par exemple un classificateur pour trouver le schéma de données en format JSON, XML, etc. Si les classificateurs intégrés ne répondent pas aux besoins particuliers, il est possible de créer des classificateurs personnalisables.

Les frais de ce service dépendent du temps écoulé lors de l'analyse des données par les robots d'analyse durant la découverte du schéma. A ces frais s'ajoutent les frais du catalogue de données qui va être peuplé par les résultats fournis par les robots de l'analyse. Par exemple, on paye 0,44 USD par heure par DPU<sup>15</sup>, il est facturé à la seconde avec un minimum de 10 minutes par robot d'analyse exécuté. Plus de détails sont disponibles sur le site Web d'Amazon Glue<sup>16</sup>.

13. Source : <https://aws.amazon.com/fr/s3/>, consultée le 06/07/2018.

14. Source : <https://aws.amazon.com/fr/glue/>, consultée le 06/07/2018.

15. DPU : unité de traitement des données.

16. Source : <https://aws.amazon.com/fr/glue/pricing/>, consultée le 05/08/2018.

**Amazon Athena**<sup>17</sup> est un service de requêtes interactif. Il permet d'interroger les données présentes dans Amazon S3 avec des requêtes SQL plus avancées. Le service Amazon Athena est considéré comme *serverless*. Amazon Athena utilise l'approche ***schema-on-read*** (voir la section III.3.5) afin de projeter le schéma donné en entrée sur les données au moment de l'exécution de la requête SQL demandée. Le schéma sur lequel les données peuvent être projetées peut être créé manuellement ou bien utiliser le catalogue créé dans Amazon Glue. Le service Amazon Athena est facturé suivant la quantité de données analysée. Précisément, 5 USD par To de données analysées.

Une ***requête est interactive*** si on peut obtenir immédiatement une réponse à la requête. Dans le cas échéant, les résultats sont obtenus dans le cadre d'un code source pour un des langages de programmation, souvent à travers une API.

***Serverless*** peut être décomposé en *server* et *less*. Un outil est *serverless* quand l'utilisateur final de cet outil peut l'utiliser sans se soucier de toute configuration ou gestion des serveurs derrière ce service. D'après Amazon<sup>a</sup>, *Serverless* est l'architecture native du cloud.

a. <https://aws.amazon.com/serverless/>, consultée le 05/08/2018.

L'exécution des requêtes SQL est effectuée par le moteur de requêtes SQL Presto. Pour les instruction DDL (Data Definition Language), elles sont effectuées par *Hive Data Definition Language*<sup>18</sup>. Les requêtes DDL incluent la création, la suppression et la mise à jour de la structure de la table dans le cas d'une base de données relationnelles, d'une collection, d'une vue, etc.

**Hive Data definition language (DDL)** est un sous-ensemble de déclarations qui décrivent la structure de données dans Apache Hive. Principalement, ce sont les instruction de création, suppression et de mise à jour de la structure des objets comme les bases de données, les tables, les vues et autres.

**Presto**<sup>a</sup> est un moteur de requêtes SQL open source destiné au Big Data. Il permet d'exécuter des requêtes analytiques interactives sur des données de taille importante; jusqu'à des Pétaoctets de données.

Presto interroge les données où elles sont hébergées. Ceci inclut les bases de données relationnelles, Amazon S3 et autres dépôts propriétaires. De plus, une même requête SQL peut combiner plusieurs sources de données. C'est intéressant pour les organisations ayant plusieurs sources de données. Il fournit les résultats en quelques secondes, voire quelques minutes. Il supporte les types de données complexes comme les objets JSON, les tableaux d'éléments, etc.

a. Source : <http://prestodb.io/>, consultée le 01/08/2018.

17. Source : <https://aws.amazon.com/fr/athena/>, consultée le 06/07/2018.

18. Source : <https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL>, consultée le 05/08/2018.

### III.4.4 Hadoop ecosystem

Hadoop est un écosystème destiné au Big Data et il regroupe plusieurs projets.

**Apache Hadoop HDFS** (Hadoop Distributed File System) est un système de fichier distribué permettant de stocker les fichiers volumineux tout en offrant une haute disponibilité, fiabilité et tolérance aux pannes.

**Hadoop Yarn (Yet Another Resource Negotiator)** est la composante de Hadoop permettant de gérer les ressources dans un cluster Hadoop. Ceci inclut l'allocation des ressources aux différentes applications. Yarn garantit les différents traitements d'être exécutés sur les données stockées dans HDFS.

**Hadoop MapReduce** représente la couche destinée aux traitements en parallèle dans l'écosystème Hadoop. En particulier, ce sont les traitements qui concernent des volumes de données importants. MapReduce divise le travail en petites parties, chacune pouvant être effectuée en parallèle sur le cluster de machines. Autrement dit, le problème est divisé en un grand nombre de problèmes plus petits, chaque petit problème est traité pour donner des résultats individuelles. Ces résultats sont ensuite traités pour donner un résultat final. Hadoop MapReduce est évolutif et peut également être utilisé sur de nombreux ordinateurs. Hadoop MapReduce est considéré comme étant moins rapide en le comparant au framework Apache Spark présenté dans la section III.4.5.

### III.4.5 Apache Spark

Apache Spark<sup>19</sup> est un framework de calcul distribué. C'est un ensemble de composantes conçues pour assurer la rapidité, la facilité d'utilisation ainsi que la flexibilité dans l'analyse des données à grande échelle. Plusieurs APIs sont disponibles pour interagir avec Spark et appliquer les transformations sur les données à analyser.

#### Core Concepts et architecture de Spark

**Spark Clusters et Resource Management System** Spark est un système distribué conçu pour traiter les données massives rapidement et avec efficacité. Ce système est déployé sur un ensemble de machines, qu'on appelle *Spark cluster*. La taille du cluster en nombre de machines est variable. Il est possible d'avoir un cluster avec peu de machines mais aussi un cluster avec des milliers de machines. En vue de gérer efficacement les machines d'un cluster, les entreprises recourent à un système de gestion de ressources tel que Apache YARN<sup>20</sup> ou Apache Mesos<sup>21</sup>. Les deux composantes les plus importantes dans un système de gestion de ressources sont : le *cluster manager* et le *worker*.

Le *cluster manager* a une vue globale de l'emplacement des *workers* ; la mémoire qu'ils ont et le nombre de cœurs CPU dont chaque worker dispose. Le rôle du *cluster manager* est d'orchestrer le travail en le désignant à chaque worker. Tandis que le rôle d'un *worker* est de fournir les informations utiles pour le *cluster manager* ainsi que la réalisation du travail y assigné. La Figure III.8 montre l'interaction entre une application spark, le cluster manager et les *workers*.

---

19. Source : <https://spark.apache.org/>, consultée le 14/12/2018

20. Description dans <https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html>, consulté le 09/12/2018.

21. Site officiel : <https://mesos.apache.org/>, consulté le 09/12/2018.

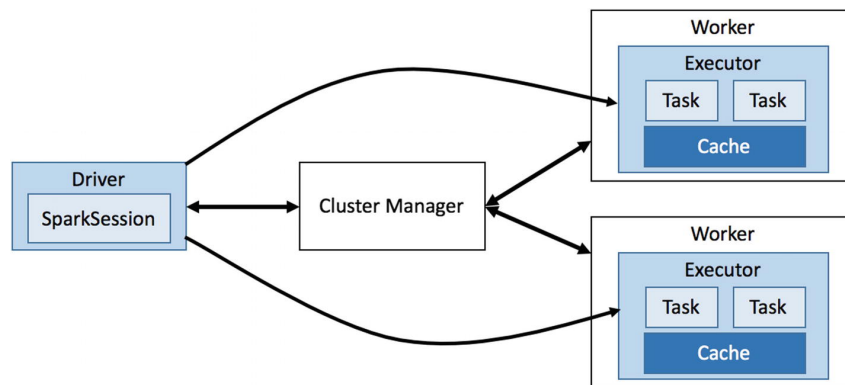


FIGURE III.8 – Interaction entre une application Spark et le cluster manager. Source : [28]

**Application Spark** Une application Spark consiste en deux parties. La première partie concerne la logique décrivant les traitements à appliquer sur les données. Cette logique est décrite en utilisant les APIs<sup>22</sup> disponibles. La deuxième partie est appelée le *driver*, c'est le coordinateur principal d'une application Spark. Le driver interagit avec le cluster manager afin de trouver les machines sur lesquelles le traitement de données doit être réalisé. Ainsi, pour chacune de ces machines, le driver Spark lance le processus *executor* en passant par le cluster manager. Un autre rôle du driver Spark est de gérer et de distribuer les tâches Spark en provenance de l'application Spark sur chaque executor. Pour précision, dans la Figure III.8, la classe *SparkSession* est le point d'entrée vers une application Spark.

**Spark driver et executor** Chaque Spark *executor* est alloué exclusivement à une application Spark spécifique et la durée de vie d'un *executor* est celle de l'application Spark.

Spark utilise l'architecture master-slave. Spark *driver* est le master et Spark *executor* est le slave. De ce fait, une application Spark n'a qu'un seul Spark driver et plusieurs Spark *executors*. Chaque Spark *executor* s'occupe d'un traitement sur une partie des données à analyser. De cette manière, Spark est capable de traiter les données de façon parallèle.

**Spark Unified Stack** Spark offre ce qu'on appelle Spark *Stack*. C'est un ensemble de composantes construites au dessus de la composante Spark Core. Ces composantes sont conçues pour répondre à des besoins spécifiques :

- Spark SQL est conçu pour le traitement interactif ;
- Spark Streaming est utilisé pour les traitements en temps réel ;
- GraphX est destiné au traitement de graphes ;
- MLlib est conçu pour le machine learning ;
- SparkR est consacré au traitement lié au machine learning en utilisant R.

**Spark Core** est la base du moteur Spark pour le traitement distribué de données. On distingue deux parties formant Spark Core. Premièrement, la partie concernant l'infrastructure distribuée du calcul. Cette dernière est responsable de la distribution, de la coordination et de la planification des tâches sur les différentes machines formant le cluster. De plus, cette partie gère l'échec d'un traitement donné et le transfert de données entre les machines. Le deuxième

22. API en Java, Scala, Python ou R.

élément formant Spark Core est appelé RDD (Resilient Distributed Dataset). Un RDD est une collection partitionnée d'objets, tolérante aux pannes et en lecture seule. La Figure III.9 présente les différentes entités du Spark Unified Stack avec Spark Core.

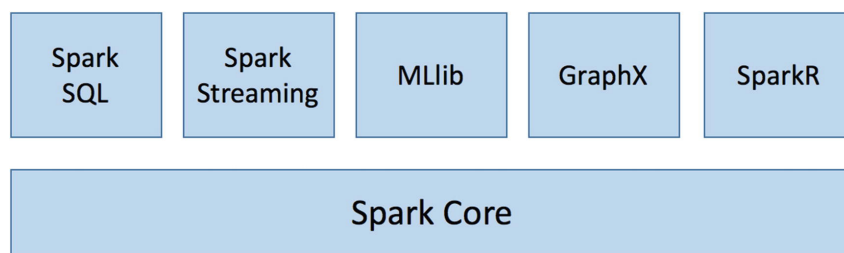


FIGURE III.9 – Spark Unified Stack. Source : [29]

### Resilient Distributed Datasets

Spark dispose d'une abstraction notée Resilient Distributed Datasets (ou RDDs). Un RDD est une collection d'objets immuable. Ces objets sont répartis sur les nœuds du cluster afin d'être traités en parallèle. Un RDD peut être conservé pour une éventuelle réutilisation. On distingue deux manières de persistance. La conservation du RDD dans la mémoire (in-memory) ce que garantit l'amélioration des performances. En outre, un RDD peut être aussi conservé dans un disque.

Les RDDs supportent deux types d'opérations sur les objets stockés : les transformations et les actions. Une transformation appliquée sur un RDD crée un nouveau RDD, par exemple la transformation *filter* retourne un RDD ayant vérifié la condition donnée en entrée. Pour les actions, une action appliquée sur un RDD retourne une seule valeur, par exemple l'action *count* calcule le nombre d'objets d'un RDD.

Input data sont les données à analyser en utilisant Spark. Ces données sont récupérées depuis des sources extérieures vers Spark. Ce dernier crée un RDD basé sur ces données. Un RDD est représenté par le rectangle en orange, les morceaux en orange dans le rectangle représentent les partitions d'un RDD. Il existe plusieurs transformations à appliquer sur un RDD, avec la possibilité d'enchaîner plusieurs transformations. Comme une transformation est à la base *lazy*, les partitions ne sont partagées sur les nœuds du cluster qu'à la suite de l'appel d'une action. Une fois qu'une partition est localisée sur un nœud donné, les transformations ainsi que les actions peuvent s'enchaîner.

En cas de perte de partition pour une raison ou une autre, Spark est capable de reproduire automatiquement la partition en question. Cette fonctionnalité est assurée via le DAG (Direct Acyclic Graph). Dans ce graphe, Spark enregistre toutes les opérations appliquées sur un RDD.

**A propos lazy evaluation** Les transformations en Spark sont *lazy*. Cela implique que lorsqu'on exécute des fonctions de transformation en Spark, celles-ci ne sont pas exécutées de suite. Par contre, elles sont enregistrés dans le graphe (DAG). Les transformations sont exécutées une fois le *driver* invoque un appel à une fonction de type action. *lazy* ou l'évaluation paresseuse est un mécanisme permettant d'éviter le chargement de données depuis la source tant que ceci n'est pas nécessaire. Par conséquent, cela peut améliorer considérablement les performances.

La Figure III.10 illustre un flux de données avec l'utilisation de Spark. Dans cette figure, *input data* représente les données qu'on souhaite analyser. Ces dernières sont en provenance de sources de stockage externes. Le framework Spark crée le RDD représenté par le premier rectangle orange à gauche. Ce dernier comprend des petits rectangles chacun représente une



partition du RDD. Les transformations peuvent être enchaînées sur le RDD créé sans être exécutées. Les partitions seront envoyées à travers les nœuds dès que le *driver* appelle une action sur ce RDD. Un nœud est une machine avec des ressources de stockage (*disk*), de calcul (*CPU*), etc. Enfin, le reste des opérations peuvent s'enchaîner sur chaque nœud où se trouvent les données.

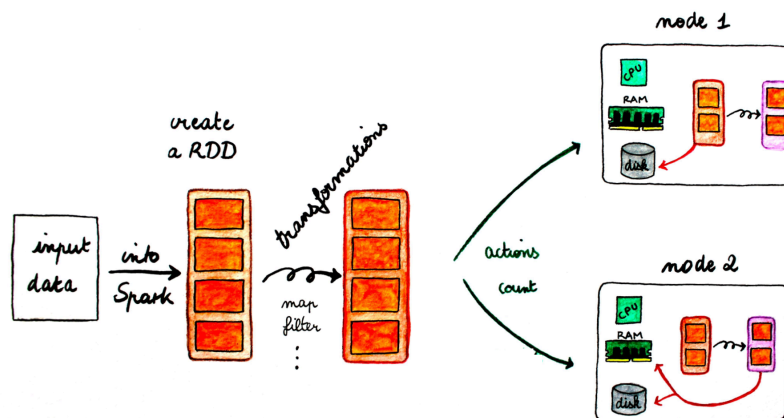


FIGURE III.10 – Exemple d'un flux de données avec Spark

Source : <https://www.duchess-france.org/starting-with-spark-in-practice/>, consultée le 15/12/2018.

**Les APIs de Spark** Le framework Spark a été écrit en Scala. Il existe plusieurs APIs pour utiliser les fonctionnalités fournies par ce framework : Scala, Python, Java et R. Il n'est pas évident de choisir entre ces quatre langages, car ce choix dépend du contexte de l'analyse.

**Présentation de Scala** La première version du langage Scala date de l'année 2003. Scala est un terme venant du *Scalable language*. C'est un langage basé sur le langage Java, ainsi, sur sa machine virtuelle. La particularité de ce langage est le fait de proposer à la fois les concepts suivants : scalability, programmation multi-paradigme et compatibilité. Scala est un langage à la fois orienté objet, un paradigme qui a prouvé ses performances depuis des années. De plus, Scala supporte la programmation fonctionnelle. En terme de compatibilité, Scala est interopérable avec le code Java, car ce dernier est compilé en bytecode Java. Ce qui permet d'utiliser du code Java avec le code Scala et garantit l'indépendance des systèmes d'exploitation du moment qu'ils disposent d'une machine virtuelle Java.

L'implémentation proposée implique plusieurs éléments relatifs au langage Scala comme les case class, les fonctions et le RDD relatif au Spark.

### Lancement d'une application avec de Spark

On distingue deux modes d'exécution d'une application écrite en Spark : cluster et local. En mode local, l'application peut être exécutée au sein d'une machine locale. Par exemple, avec un seul worker thread (aucun parallélisme), en précisant  $K$  worker sur  $K$  threads, idéalement  $K$  est le nombre des cœurs de la machine sur laquelle Spark est installé, etc. Pour le mode cluster, on distingue plusieurs types de clusters. Ces derniers varient suivant le gestionnaire de ressources qui assure la communication entre les différentes entités du cluster selon le principe

master-slave. Par exemple Hadoop YARN<sup>23</sup>, Apache Mesos<sup>24</sup>. La liste détaillée des alternatives pour chaque mode est donnée dans le site Web de Spark<sup>25</sup>.

Il est possible d'installer Spark dans un cluster Amazon EMR (Elastic MapReduce)<sup>26</sup>. Par défaut, ce cluster utilise YARN comme gestionnaire de ressources.

## III.5 Conclusion

Dans ce chapitre, nous avons décrit brièvement quelques technologies du Big Data, car la liste de toutes les technologies est très longue. Afin de découvrir ces technologies en pratique, nous allons aborder dans le chapitre V l'utilisation de ces technologies dans le cas de l'analyse des délais d'un lien décrite dans la chapitre ??.

---

23. Source : <https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html>, consultée le 14/05/2019

24. Source : <https://mesos.apache.org/>, consultée le 14/05/2019.

25. <https://spark.apache.org/docs/latest/submitting-applications.html#master-urls>, consultée le 14/05/2019.

26. Source : <https://aws.amazon.com/fr/emr/>, consultée le 08/05/2019



# Chapitre IV

## Implémentation en Spark/Scala

### IV.1 Introduction

Ce chapitre décrit l'implémentation de l'outil de détection en utilisant le framework Spark, décrit dans la section [III.4.5](#), en utilisant l'API Scala (voir la présentation de Scala dans [III.4.5](#)).

### IV.2 Implémentation

#### IV.2.1 Description de l'environnement

La création d'une application Spark/Scala passe par les étapes suivantes :

- gérer les dépendances nécessaires au fonctionnement de l'application avec le fichier de modèle objet du projet (POM) ;
- écrire l'application dans Scala ;
- générer le fichier JAR de l'application ;
- exécuter l'application en soumettant cette dernière au cluster.

Dans l'implémentation proposée, nous avons utilisé le mode local de Spark, intitulé *Standalone*. Le code source, qui traduit l'ensemble des traitements, est organisé dans une archive de type JAR. En ce qui concerne l'automatisation et la gestion du fichier JAR, nous avons utilisé l'outil *Maven*<sup>1</sup>.

#### IV.2.2 Brève présentation de l'implémentation

Un programme Spark implique un ensemble d'éléments. La première chose à faire est de configurer le programme Spark. Ensuite, nous utilisons la composante Spark SQL du Spark Unified Stack (voir la section [III.4.5](#)) pour lire les traceroutes présents dans des fichiers d'entrée. Cela permet de créer un Dataset d'objet Traceroute. Nous convertissons ensuite un Dataset en un RDD afin d'appliquer différentes transformations aboutissant à l'identification des anomalies dans les délais des liens.

---

1. <https://maven.apache.org/>, consultée le 09/04/2019.

## IV.2.3 Création d'une application Spark/Scala

### Gestion des dépendances

Le fichier POM permettant de gérer les dépendances est disponible sur GitHub<sup>2</sup>. Ces dépendances concernent les composantes de Spark comme spark-core, spark-mllib, spark-sql. De plus, il y a les dépendances permettant de gérer les fichiers de type JSON.

### Notation relatives au langage Scala

Nous présentons des notations propres au langage Scala, elles sont utilisées dans quelques Listings. c'est une liste non exhaustive, elle est utilisée pour comprendre les morceaux de code introduits tout au long de ce chapitre.

**case class** : créer une case classe.

**Seq** : créer une séquence, équivalent à une liste. Par exemple Seq[String] représente le type d'une liste dont ses éléments sont de type String.

**Dataset** : est une collection distribuée de données.

**map** : permet de transformer le contenu d'une liste en appelant une fonction sur chaque élément de la liste, elle renvoi une liste transformée.

**Option** : représente une valeur optionnelle.

**groupBy** : groupBy est une fonction utile pour regrouper les éléments d'une liste ayant une clé en commun.

**List** : crée une liste d'un type donné. La liste l\_1 de deux chaînes de caractères est créée comme suit :

```
1 l_1 : List[Int] = List("aa", "bb")
```

**filter** : cette fonction permet de filtrer les éléments d'une collection pour créer une nouvelle collection contenant uniquement les éléments de la collection vérifiant une condition. Prenons un exemple :

```
1 // créer une liste d'entier entre 1 et 10, l_2 vaut List(1, 2, 3, 4, 5, 6,
  7, 8, 9)
2 val l_2 = List.range(1, 10)
3 // filter les éléments de l_2 afin de ne garder que ceux paires, evens vaut
  List(2, 4, 6, 8)
4 val evens = l_2.filter(_ % 2 == 0)
```

**Paramètres de l'analyse** Afin de tracer l'évolution du délai des liens, nous avons besoin des fichiers stockant les traceroutes dans des objets JSON<sup>3</sup>, la date du début de l'analyse (p. ex. 1517961600), la date de fin (p. ex. 1518134400) et enfin la durée d'une période (p. ex. 3600s). En ce qui concerne les fichiers de données, ils sont stockés localement et le chemin vers ces derniers est configuré dans un fichier de configuration.

### Configuration d'une application Spark

Une application Spark nécessite l'ajustement de quelques paramètres, qu'il s'agisse d'une application qui tourne en mode local ou bien en mode cluster. On peut ajuster ces paramètres

2. <https://github.com/hayatbellafkih/SparkSalacaTraceroutesAnalysis/blob/master/rttDelaysSparkScala/pom.xml>, consultée le 23/04/2019.

3. Voir la liste des traceroutes utilisés dans l'exemple illustratif disponible sur GitHub [https://github.com/hayatbellafkih/SparkSalacaTraceroutesAnalysis/blob/master/rttDelaysSparkScala/src/main/resources/test/result\\_modified.json](https://github.com/hayatbellafkih/SparkSalacaTraceroutesAnalysis/blob/master/rttDelaysSparkScala/src/main/resources/test/result_modified.json), consultée le 23/04/2019.

selon trois possibilités. La première possibilité est à travers l'objet *SparkConf* comme illustré dans l'exemple du Listing IV.1, où nous donnons un nom à l'application Spark (ligne 5) et nous précisons l'URL du cluster (ligne 6).

Listing IV.1 – Exemple de configuration avec SparkConf

```

1 // imports
2 import org.apache.spark.SparkConf
3
4 // Spark configuration : create configuration
5 val conf = new SparkConf().setAppName("Link delay analysis")
6                               .setMaster("local"),

```

Nous pouvons passer certains paramètres au moment de la soumission de l'application au cluster. Un exemple de ces paramètres est illustré dans le Listing IV.32. Enfin, quelques paramètres peuvent être lus depuis le fichier de configuration *conf/spark-defaults.conf*<sup>4</sup>.

### Point d'entrée vers les fonctionnalités de Spark

Le point d'entrée vers les fonctionnalités de Spark se fait par la création du *SparkContext*. Néanmoins, il existe d'autres points d'entrée qui sont plus spécifiques aux composantes du *Spark Unified Stack*. Par exemple, *SparkSession* est le point d'entrée vers Spark SQL, *StreamingContext* est le point d'entrée vers Spark Streaming, etc. Dans notre cas, nous avons utilisé *SparkSession* pour lire les traceroutes en tant que liste d'objets.

### Lecture de données

L'outil de détection proposé par R. Fontugne et al. n'exploite qu'une partie des données d'une réponse traceroute<sup>5</sup>. En particulier, on peut utiliser Spark SQL pour ne lire que les données qui nous intéressent, c'est un des avantages du principe du *Schema-On-Read* décrit dans la section III.3.5.

Chaque réponse traceroute est structurée dans un objet JSON dans une seule ligne. Afin de lire chaque ligne, nous avons créé la classe *Traceroute*, cette dernière a pour objectif de faire l'association entre l'objet JSON et un objet *Traceroute* de sorte à encapsuler les données d'un objet JSON. La classe *Traceroute* reprend le nom de la destination de la requête traceroute (*dst\_name*), l'adresse IP de la sonde effectuant la requête traceroute (*from*), l'identifiant de cette sonde (*prb\_id*), le temps de la requête (*timestamp*) et enfin la liste des sauts (*Seq[Hop]*). La classe *Traceroute* est définie en Scala comme montre le Listing IV.2.

Listing IV.2 – Définition de la classe Traceroute

```

1 case class Traceroute(
2     dst_name : String ,
3     from :    String ,
4     prb_id :  BigInt ,
5     msm_id :  BigInt ,
6     timestamp : BigInt ,
7     result :  Seq[Hop])

```

Un saut représente un des routeurs parcourus avant d'atteindre la destination finale. Nous modélisons un saut par la classe *Hop* (voir le Listing IV.3). Un saut est défini par son rang (*hop*), ce dernier indique l'ordre du saut en question. Etant donné que la sonde reçoit au moins trois signaux pour chaque saut. Un saut est donc défini par une liste de signaux (*Seq[Signal]*).

4. Plus de détails relatifs à la configuration sont disponibles sur <https://spark.apache.org/docs/latest/configuration.html>, consultée le 14/04/2019.

5. Un exemple d'une réponse traceroute est donné dans l'annexe C.

Listing IV.3 – Définition de la classe Hop

```

1 case class Hop(
2     var result : Seq[ Signal ],
3     hop :      Int )

```

Un signal (*Signal*) est émis par un routeur dont l'adresse IP est *from*. Le temps nécessaire à la réception du signal par la source est *rtt*. Enfin, *x* est un indicateur de la validité du signal car il se peut que la sonde ne reçoive pas une réponse d'un ou de plusieurs routeurs. Si le signal est valide, *x* est une chaîne vide. Dans le cas d'un signal invalide, *x* vaut "\*". Nous précisons qu'un signal est invalide quand la sonde ne reçoit pas une réponse après le temps *timeout*. Afin d'adapter un saut aussi dans le cas de l'absence des détails du signal, les attributs *rtt*, *x* et *from* sont définis comme étant optionnels. Un signal est modélisé par la classe *Signal* (voir le Listing IV.4).

Listing IV.4 – Définition de la classe Signal

```

1 case class Signal(
2     rtt : Option[ Double ],
3     x :   Option[ String ],
4     from : Option[ String ])

```

Nous avons défini la classe *Traceroute* qui nous permet de lire les données. Pour ce faire, nous utilisons l'objet *spark* de type *SparkSession* créé précédemment. En particulier, nous faisons appel à la fonction *read()* via ce dernier. Nous spécifions à *read()* le schéma de lecture à travers la classe *Traceroute*, le chemin vers les fichiers de données (*dataPath*) et comment les données sont structurées (*json*).

Listing IV.5 – La lecture des données traceroutes

```

1 val rawTraceroutes = spark.read
2   .schema( Encoders . product[ Traceroute ] . schema )
3   .json( dataPath )
4   .as[ Traceroute ]
5 import spark.implicits._

```

Nous obtenons la liste des traceroutes dans la variable *rawTraceroutes*. Ce dernier est un Dataset d'objets *Traceroute*. La correspondance entre un enregistrement traceroute JSON et une instance de *Traceroute* se base sur les noms des attributs dans JSON. Nous notons que la réussite de la correspondance ne nécessite pas l'association de tous les attributs de l'objet JSON. C'est un des avantages du principe *schema on read*. Si par exemple, nous définissons des attributs dans la classe *Traceroute* et que ces attributs ne font pas partie des attributs de l'objet JSON, aucune erreur ne sera survenue. La ligne 5 du Listing IV.5 est nécessaire pour pouvoir utiliser l'API du Spark, précisément, les fonctionnalités relatives DataSets et DataFrames. L'appel à la ligne 5 du Listing IV.5 ne peut pas être fait sans avoir une instance du *SparkContext*. Dans notre cas, nous avons défini une instance de *SparkSession*, et une instance de *SparkContext* est fournie, par défaut, avec cette dernière.

### Trouver les périodes de l'analyse

Dès à présent, la liste des traceroutes est prête à toute transformation de la phase I (voir la phase I dans la section II.4.3). Tout d'abord, nous devons trouver les périodes entre la date de début et la date de fin (étape *FindBins* (I.1)). Cette étape est illustrée par la fonction *generateDateSample* dans le Listing IV.6; nous construisons les tuples de périodes afin de faciliter le test d'appartenance d'un traceroute, un tuple est formé par le début de la période (*start*), *start+timewindow*, nous prenons *timewindow* comme étant équivalent à une heure.

Listing IV.6 – Etape FindBins (I.1)

```

1 // Generate the start of all bins : between start date and end date
  espaced by the timewindow
2 val rangeDates = generateDateSample(start, end, timewindow)
3
4 // Find the start and the end of each bin
5 val rangeDatesTimewindows = rangeDates.map(f => (f, f + timewindow))

```

A la fin de cette étape, toutes les périodes sont déterminées. Nous passons à l'étape du groupement des traceroutes, disponibles à l'analyse, par période (étape I.2).

### Groupement des traceroutes

L'objectif de cette étape est de grouper les traceroutes capturés par période. Dans l'implémentation du travail de référence[17], les données sont organisées dans des collections MongoDB (voir la section III.4.1), le groupement des traceroutes par période se base sur la structuration des noms des collections<sup>6</sup>. Pour une période donnée, seules les collections concernées seront interrogées.

**MongoDB** Nous résumons dans la Figure IV.1 le groupement tel qu'il est présenté dans le travail de référence. Selon la période en question, nous interrogeons la collection adéquate.

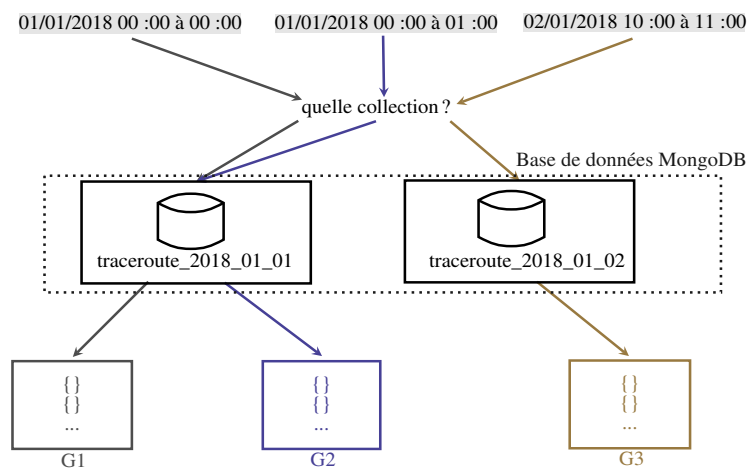


FIGURE IV.1 – Groupement des traceroutes avec MongoDB

Nous illustrons ce groupement avec un pseudo-code décrit par l'algorithme 1.

---

#### Algorithm 1 Groupement des traceroutes dans le cas de MongoDB

---

```

for  $period \in rangeDatesTimewindows$  do
   $collection \leftarrow$  Trouver la collection incluant  $period$ 
   $rawTraceroutes \leftarrow$  les traceroutes stockés dans  $collection$  et
  enregistrés durant  $period$ 
  ...
  ▷ Traitements appliqués sur l'ensemble de traceroutes
end for

```

---

Cette manière de grouper les traceroutes ne prend pas en considération le cas où il faut chercher les traceroutes dans plus d'une collection.

---

6. Voir la section V.4.

**Spark/Scala** En ce qui concerne l'implémentation en Spark/Scala, nous avons groupé les traceroutes autrement. Les traceroutes sont organisés dans des fichiers reprenant des traceroutes d'une heure, d'une journée ou d'une toute autre période. Peu importe le nombre de ces fichiers, Spark lit tout ce qui est disponible dans le chemin *dataPath*. Notons que le parcours de tous les fichiers à chaque période est coûteux en terme de performance. C'est pourquoi nous avons attribué les traceroutes aux périodes. Dans ce cas, les fichiers de données sont lus une seule fois. C'est ce que nous résumons dans l'algorithme 2. Les lignes entre 1 et 7 de l'algorithme 2 permettent d'associer chaque traceroute à une période faisant partie de la période de l'analyse. La ligne 8 permet de grouper les traceroutes par période; il s'agit de grouper les traceroutes ayant la même période. L'objectif des lignes entre 9 et 11 de l'algorithme 2 est d'appliquer les traitements sur les groupes de traceroutes, cependant, elles ne sont pas détaillées dans cet algorithme.

---

**Algorithm 2** Groupement des traceroutes en Spark
 

---

```

1: traceroutePerPeriod ← []
2: for traceroute ∈ rawTraceroutes do
3:   for period ∈ rangeDatesTimewindows do
4:     Vérifier si traceroute appartient à period
5:   end for
6:   traceroutePerPeriod.append( (traceroute, period) )
7: end for
8: traceroutesPerPeriods ← traceroutePerPeriods.groupBy(period)
9: for element ∈ traceroutesPerPeriods do
10:   ...                                ▷ Traitements appliqués sur l'ensemble de traceroutes
11: end for

```

---

En Spark/Scala, nous avons chargé les traceroutes disponibles à l'analyse sur un RDD. Ce dernier crée des partitions de données, sensées être manipulées sur différentes machines si Spark est lancé sur un cluster de machine. Afin de créer les groupes de traceroutes, nous vérifions l'appartenance de chaque traceroute à une des périodes considérées.

```

1 // Group each traceroute by the bin that they belong in
2 // If one traceroute does not belongs in any bin, then by default it
  belongs to the bin 0
3 val tracerouteAndPeriodRdd = rawTraceroutes.rdd.map(traceroute =>
  TracerouteWithTimewindow(traceroute, findTimeWindowOfTraceroute(
  traceroute, rangeDatesTimewindows)))

```

Avec la ligne 3 dans le Listing IV.2.3 :

- nous transformons *rawTraceroutes* en un RDD;
- pour chaque objet *Traceroute*, nous appliquons le traitement du groupement en utilisant la méthode *findTimeWindowOfTraceroute*. Cette dernière prend en paramètre le traceroute et les périodes possibles dont ce dernier peut y appartenir et elle renvoie la période adéquate;
- nous passons d'un RDD de type *Traceroute* à un RDD de type *TracerouteWithPeriod*.

La classe *TracerouteWithPeriod*, définie dans le Listing IV.7, permet de représenter un traceroute avec sa période dans une seule entité.

Listing IV.7 – Définition de la classe `TracerouteWithPeriod`

```

1 case class TracerouteWithPeriod (
2     traceroute : Traceroute ,
3     period :      Int )

```

### Élimination des traceroutes qui n'appartiennent pas à l'analyse

Il se peut qu'un traceroute ne fait pas partie de la période de l'analyse. Ce sont les objets de type *TracerouteWithPeriod* ayant *period* de valeur 0 et période = 0 si *traceroute.timestamp* n'appartient à aucune des période. C'est pourquoi nous filtrons ces traceroutes, comme illustre le Listing IV.8.

Listing IV.8 – Élimination des traceroutes non concernés par l'analyse

```

1 val onlyConcernedTraceroutes = tracerouteAndPeriodRdd.filter(_ . period != 0)

```

Après l'élimination des traceroutes non concernés, nous agrégeons ces derniers par période pour construire une liste de type *TraceroutesPerPeriod* dont sa définition est donnée dans le Listing IV.10. L'objectif de cette agrégation est de créer des groupes de traceroutes et y appliquer les traitements relatifs à la détection. Le Listing IV.9 illustre l'étape de l'agrégation.

Listing IV.9 – Le groupement des traceroutes

```

1 val groupedTraceroutesByPeriod = onlyConcernedTraceroutes.groupBy(_ . period)
2 val traceroutesPerPeriod = groupedTraceroutesByPeriod.map(f =>
    TraceroutesPerPeriod(f._2.map(f => f.traceroute).toSeq, f._1))

```

Listing IV.10 – Définition de la classe *TraceroutesPerPeriod*

```

1 case class TraceroutesPerPeriod (
2     traceroutes : Seq[ Traceroute ],
3     period :      Int )

```

### Déduction des liens

L'étape qui suit le groupement des traceroutes est la génération des liens. Ainsi, nous générons les différents liens possibles dans chacune des périodes avec le code du Listing IV.11 :

Listing IV.11 – La classe *TracerouteWithPeriod*

```

1 val allLinksRttDiffsPeriods = traceroutesPerPeriod.map(f => linksInference(
    spark , f))

```

La fonction *linksInference* est une abstraction de plusieurs traitements appliqués sur chaque groupe de traceroutes. Cette fonction renvoie la liste des liens caractérisés par leurs périodes et RTTs différentiels. Le Listing IV.12 illustre les étapes de la déduction des liens par groupe de traceroutes, ces étapes sont les suivantes :

1. élimination des traceroutes échoués ;
2. élimination des sauts non valides ;
3. calcul de la médiane de chaque saut ;
4. lister les liens possibles par traceroute en encapsulant ces derniers dans des objets *DetailedLink* ;
5. construction d'une liste reprenant les listes de l'étape 4.
6. tri alphanumérique des adresses IP de chaque lien ;



7. groupement des liens ayant les mêmes adresses IP ;
8. création d'un récapitulatif de chaque lien ; chaque lien est associé à une liste des RTTs différentiels ainsi que la liste des périodes. La liste des périodes contient la période courante dupliquée en nombre de RTTs différentiels de ce lien durant cette période.

Listing IV.12 – Deduction des liens par groupe de traceroutes

```

1  def linksInference(spark : SparkSession , rawtraceroutes :
    TraceroutesPerPeriod) : Seq[ResumedLink] = {
2
3      //Filter failed traceroutes ...
4      val notFailedTraceroutes = rawtraceroutes.traceroutes.filter(t => t
        .result(0).result != null)
5
6      //Remove invalid data in hops
7      val cleanedTraceroutes = notFailedTraceroutes.map(t =>
        removeInvalidSignals(t))
8
9      //Compute median by hop
10     val tracerouteMedianByHop = cleanedTraceroutes.map(t =>
        computeMedianRTTByHop(t))
11
12     //Find links in a traceroute
13     import org.apache.spark.mllib.rdd.RDDFunctions._
14     val tracerouteLinks = tracerouteMedianByHop.map(t =>
        findLinksAndRttDiffByTraceroute(spark , t))
15
16     //Create a set of DetailedLink objects for every traceroute
17     val detailedLinks = tracerouteLinks.map(summarizeLinksTraceroute)
18
19     //Flatten the list of lists to have one liste of DetailedLink
        objects
20     val allDetailedLinks = detailedLinks.flatten
21
22     //Sort the links
23     val sortAllDetailedLinks = allDetailedLinks.map(l => sortLinks(l))
24
25     //Merge the links from all traceroutes in the current bin
26     val mergedLinks = sortAllDetailedLinks.groupBy(_.link)
27
28     //Summarize the link
29     val summarizeLink = mergedLinks.map(f => ResumedLink(f._1, f._2.map
        (_.probe), f._2.map(_.rttDiff), generateDatesSample(f._2.size ,
        rawtraceroutes.timeWindow)))
30
31     summarizeLink.toSeq
32 }

```

Nous reprenons dans les sections suivantes la définition de fonctions utilisées dans la fonction *linksInference*.

**Elimination des traceroutes échoués** (Ligne 4 dans le Listing IV.12) Nous parcourons chaque traceroute de la liste des traceroutes *rawtraceroutes.traceroutes* afin d'éliminer tout traceroute échoué. Dans l'implémentation de l'outil de détection, ils ont constaté que la liste des sauts d'un traceroute échoué contient un seul saut (*t.result(0)*) et la liste des signaux



n'existe pas (*t.result(0).result != null*). Avec *t* dénote le traceroute courant dans la liste *rawtracroutes.traceroutes*.

**Elimination des sauts non valides** (Ligne 7 dans le Listing IV.12) Nous parcourons les traceroutes valides (*notFailedTraceroutes*) et nous appliquons *removeInvalidSignals* sur chaque traceroute. Cette fonction renvoie un traceroute après avoir appliqué la fonction *checkSignal* sur chaque signal de chaque saut.

Listing IV.13 – Définition de la méthode *removeInvalidSignals*

```

1  def removeInvalidSignals(traceroute : Traceroute) : Traceroute = {
2      val hops = traceroute.result
3      for (hop <- hops) {
4          val signals = hop.result
5          val tmpSignals = signals.filter(checkSignal(_))
6          hop.result = tmpSignals
7      }
8      traceroute
9  }
```

Le rôle de la fonction *checkSignal*, définie dans le Listing IV.14, est de vérifier chaque signal : si le signal est valide (Ligne 3), si le RTT existe (Ligne 6), si le RTT est positif et enfin si le signal ne provient pas d'une adresse IP privée (Ligne 10).

Listing IV.14 – Définition de la méthode *checkSignal*

```

1  def checkSignal(signal : Signal) : Boolean = {
2      // Check if a signal is not failed
3      if (signal.x == "*")
4          return false
5      // Check if the RTT exist
6      else if (signal.rtt == None)
7          return false
8      else if (signal.rtt.get <= 0) {
9          return false
10     } else if (javatools.Tools.isPrivateIp(signal.from.get))
11         return false
12     else {
13         return true
14     }
15 }
```

**Calcul de la médiane de chaque saut** (Ligne 10 dans le Listing IV.12) Après avoir vérifié les sauts des traceroutes, nous appelons la fonction *computeMedianRTTByhop* présentée dans le Listing IV.15 pour calculer la médiane des RTTs par saut. Dans un premier temps, la fonction *computeMedianRTTByhop* récupère les sauts du traceroute donné en entrée, elle calcule la médiane des RTTs et renvoie un traceroute avec des RTTs agrégés.

Listing IV.15 – Définition de la fonction *computeMedianRTTByhop*

```

1  def computeMedianRTTByhop(traceroute : Traceroute) : MedianByHopTraceroute
    = {
2      val hops = traceroute.result
3      val procHops = hops.map(f => findMedianFromSignals(f))
4      s
```

```

5      MedianByHopTraceroute(traceroute.dst_name, traceroute.from,
6      traceroute.prb_id, traceroute.msm_id, traceroute.timestamp,
        procHops)

```

Le traceroute renvoyé est de type *MedianByHopTraceroute*. Ce dernier est défini dans le Listing IV.16. Une instance de la classe *Traceroute* est se diffère d’une instance de la classe *MedianByHopTraceroute* au niveau de l’attribut *result* qui désigne la liste des sauts.

Listing IV.16 – Définition de la classe MedianByHopTraceroute

```

1 case class MedianByHopTraceroute(
2     dst_name: String,
3     from: String,
4     prb_id: BigInt,
5     msm_id: BigInt,
6     timestamp: BigInt = 0,
7     result: Seq[PreparedHop])

```

La nouvelle classe qui représente un saut est définie dans le Listing IV.17.

Listing IV.17 – Définition de la classe PreparedHop

```

1 case class PreparedHop(
2     var result: Seq[PreparedSignal],
3     hop: Int)

```

Dans la nouvelle définition du saut, ce dernier est toujours défini par une liste de signaux, car la sonde peut recevoir les signaux, pour un même saut, depuis différents routeurs. La nouvelle classe représentant un signal est définie dans le Listing IV.18, où *medianRtt* est la médiane des RTTs en provenance du routeur *from*.

Listing IV.18 – Définition de la classe PreparedSignal

```

1 case class PreparedSignal(
2     medianRtt: Double,
3     from: String)

```

**Enumération des liens possibles par traceroute et leur RTT différentiel** (Ligne 14 du Listing IV.12) La fonction *findLinksAndRttDiffByTraceroute* prend en paramètre une instance de *MedianByHopTraceroute* et renvoie une instance de *LinksTraceroute*. Les liens sont déduits suivant l’approche décrite dans la Figure ?? . Le rôle de cette méthode est de parcourir les sauts afin de construire les liens possibles. Précisément, c’est la fonction *findAllLinks* (Ligne 8 du Listing IV.19) qui crée les liens possibles pour tous deux sauts consécutifs et calcule leur RTT différentiel.

Listing IV.19 – Définition de la fonction findLinksAndRttDiffByTraceroute

```

1 def findLinksAndRttDiffByTraceroute(spark: SparkSession, traceroute:
2   MedianByHopTraceroute): LinksTraceroute = {
3     val hops = traceroute.result
4     val size = hops.size
5     val s = hops.zipWithIndex
6     val z = s.map {
7       case (element, index) =>
8         if (index + 1 < size) {
9           findAllLinks(hops(index + 1), element)
10        } else {

```

```

10         null
11     }
12 }
13 return new LinksTraceroute(traceroute.dst_name, traceroute.from,
14     traceroute.prb_id, traceroute.msm_id, traceroute.timestamp, z.
        filter(p => p != null).flatten)

```

La fonction *findAllLinks*, définie dans le Listing IV.20, prend deux paramètres de type *PreparedHop* : *nextHop* et *currentHop*. Cette fonction renvoie la liste des liens (*links*) avec leur RTT différentiel.

Listing IV.20 – Définition de la fonction *findAllLinks*

```

1 def findAllLinks(nextHop : PreparedHop, currentHop : PreparedHop) : Seq[Link] = {
2     var links = Seq[Link]()
3
4     for (nextRouter <- nextHop.result) {
5         for (currentRouter <- currentHop.result) {
6             val rttDiff = BigDecimal(nextRouter.medianRtt) -
                BigDecimal(currentRouter.medianRtt)
7             links = links :+ Link(nextRouter.from,
                currentRouter.from, rttDiff.toDouble)
8         }
9     }
10    return links
11 }

```

Après avoir parcouru les sauts consécutifs du traceroute, nous créons une instance de la classe *LinksTraceroute*. Cette classe reprend la liste des liens au lieu de la liste des sauts avec les autres informations du traceroute. La définition de cette classe est donnée dans le Listing :

Listing IV.21 – Définition de la fonction

```

1 case class LinksTraceroute(
2     dst_name : String,
3     from : String,
4     prb_id : BigInt,
5     msm_id : BigInt,
6     timestamp : BigInt,
7     links : Seq[Link])

```

La classe *Link* modélise un lien et son RTT différentiel (voir le Listing IV.22).

Listing IV.22 – Définition de la classe *Link*

```

1 case class Link(
2     ip1 : String,
3     ip2 : String,
4     rttDiff : Double)

```

**Construction de la liste des liens par période** (Ligne 17 du Listing IV.12) A cette étape, nous souhaitons réorganiser les liens précédemment énumérés par traceroute. L'objectif de cette réorganisation est de passer des liens identifiés dans le cadre d'un traceroute (une instance de *LinksTraceroute*) à une liste de liens (plusieurs instances de *DetailedLink*) comme illustre la définition de la fonction *summarizeLinksTraceroute* dans le Listing IV.23.

Listing IV.23 – Définition de la fonction summarizeLinksTraceroute

```

1  def summarizeLinksTraceroute(traceroute : LinksTraceroute) : Seq[
    DetailedLink] = {
2      val links = traceroute.links
3      val summarizedLinks = links.map(f => DetailedLink(f.rttDiff,
    LinkIPs(f.ip1, f.ip2), traceroute.prb_id))
4      summarizedLinks
5  }

```

La classe *DetailedLink*, définie dans le Listing IV.24, permet modéliser un seul lien.

Listing IV.24 – Définition de la classe DetailedLink

```

1  case class DetailedLink(
2      rttDiff : Double,
3      var link : LinkIPs,
4      probe : BigInt)

```

La fonction *linksInference* est appliqué sur chaque groupe de traceroutes. Dans ce cas, pour chaque groupe de traceroutes, on associe une liste des liens encapsulés chacun dans une instance de *DetailedLink*. Maintenant que chaque lien reprend l'information de la période pendant laquelle il a été identifié, nous pouvons créer une liste d'instance de *DetailedLink* concernant toutes les périodes. C'est ce que fait la ligne 20 dans la fonction dans le Listing IV.12.

### Tri alphanumérique des adresses IP de chaque lien (Ligne 23 dans le Listing IV.12).

La méthode *sortLinks*, définie dans le Listing IV.25, permet de trier les deux adresses IP du lien donné en paramètre et renvoi un nouveau lien.

Listing IV.25 – Définition de la fonction sortLinks

```

1  def sortLinks(linkToSort : DetailedLink) : DetailedLink = {
2      val link = Seq(linkToSort.link.ip1, linkToSort.link.ip2)
3      val sortedLink = link.sorted
4      linkToSort.link = LinkIPs(sortedLink(0), sortedLink(1))
5      linkToSort
6  }

```

### Groupeement des liens ayant les mêmes adresses IP (Ligne 26 dans le Listing IV.12)

Nous regroupons les liens par leurs adresses IP en utilisant la fonction *GroupBy* de Scala.

**Création d'un récapitulatif de chaque lien** (Ligne 29 dans le Listing IV.12) C'est la dernière étape de la phase de la préparation de données. L'objectif de cette étape est de regrouper les même lien dans une seule entité, représentée par une instance de la classe *ResumedLink* définie dans le Listing IV.26. Cette classe reprend les deux adresses IP du lien (*link*), la liste des sondes ayant identifié ce lien (*probes*), la liste des RTTs différentiels de ce lien (*rttDiffs*) et enfin les *periods* qui représentent les périodes pendant lesquelles les *rttDiffs* ont été identifiés.

Listing IV.26 – Définition de la classe ResumedLink

```

1  case class SummarizedLink(
2      link : LinkIPs,
3      probes : Seq[BigInt],
4      rttDiffs : Seq[Double],
5      var periods : Seq[Int])

```

### Caractérisation des liens de toutes les périodes de l'analyse

Après avoir traité tous les groupes de traceroutes, nous obtenons un RDD de liste de liens (*RDD[Seq[classes.ResumedLink]]*). Nous devons collecter les résultats des traitements de chacune des partitions de ce RDD afin de passer à la phase II de l'analyse des délais. Dans le cas d'un cluster de machines, il s'agit de la collecte de ces résultats de la part de chaque machine. Le code illustrant la collecte des résultats est illustrée par le Listing IV.27. Autrement dit, le *driver* reçoit les résultats des traitements appliqués sur les données et effectués par les *workers*<sup>7</sup>.

Listing IV.27 – Collecte des résultats intermédiaires

```
1 val finalLinksDetailsList = allLinksRttDiffsPeriods.collect().toSeq.flatten
```

Après avoir collecté les résultats intermédiaires, nous fusionnons les données des liens en provenance de toutes les périodes. C'est ce que illustre le Listing IV.28. La différence entre cette collecte et la collecte précédente (Ligne 20 dans le Listing IV.12), c'est que la première a été faite au niveau d'un *worker*, or, la collecte précédente est fait au niveau du *driver*.

Listing IV.28 – Fusion des liens de toute la période de l'analyse

```
1
2 // Merge all links from all periods
3 val finalResult = collectedRTTDiff.groupBy(_.link)
4 val finalRawRttDiff = finalResult.map(f => ResumedLink(
5     f._1,
6     (f._2.map(_.probes)).flatten,
7     (f._2.map(_.rttDiffs)).flatten,
8     (f._2.map(_.periods)).flatten)
9 )
```

Commentaires du Listing IV.28 :

*finalResult* est de type *Map* : la clé est de type *LinkIPs* et la valeur est de type *Seq[classes.SummarizedLink]* (*Map[classes.LinkIPs, Seq[classes.SummarizedLink]]*).

En parcourons *finalResult*, l'élément courant parcouru *f* contient deux partie, vu qu'il appartient à une *Map*, la première, *f.\_1*, représente la clé (de type *LinkIPs*) et la deuxième, *f.\_2*, représente la valeur (de type *Seq[classes.SummarizedLink]*). Ainsi, le rôle de *f.\_2.map(\_.periods).flatten* est parcourir (via la fonction *map*) la liste des liens, et de récupérer en particulier la liste des période stockée dans *periods* enfin regrouper toute les période en une seule liste (via *flatten*).

A cette étape, nous avons une liste de type *ResumedLink* qui concerne la période entière de l'analyse.

### Détection des anomalies

Nous présentons dans ce qui suit la phase II de l'analyse des délais. A travers la méthode *listAlarms()* nous analysons un lien et nous identifions les anomalies de ce dernier. Dans le Listing IV.29, d'abord nous convertissons la liste des liens en un RDD afin de distribuer le traitement de ces liens. Ensuite, nous appliquons la méthode *listAlarms* sur tout lien.

A la fin de la phase I, nous obtenons une liste des liens de type *Iterable* (*Iterable[classes.SummarizedLink]*). Comme chaque lien peut être traité indépendamment des autres liens, nous créons un RDD à partir de cette liste (Ligne 2 du Listing IV.29), ensuite, nous appliquons la méthode sur chaque lien (Ligne 5 du Listing IV.29).

Listing IV.29 – Détection des alarmes des liens

```
1 // Create a RDD having the ResumedLink elements from stage I
```

7. Voir le principe de la distribution des traitements en Spark dans la section III.4.5.

```

2  val finalRawRttDiffRdd = spark.sparkContext.parallelize(finalRawRttDiff.
    toSeq)
3
4  // Alarms detection by link
5  val linkAnalysisResult = finalRawRttDiffRdd.map(p => listAlarms(spark, p,
    timewindow, rangeDates))

```

La méthode *listAlarms* est détaillée dans le Listing IV.30. Dans cette dernière, nous assurons :

1. initialisation des variables : *reference* est l'état référence du lien, *current* est l'état courant du lien, *alarmsValues* est la liste des alarmes qui sont des RTTs différentiels médians, *alarmsDates* est la liste dates correspondantes aux alarmes, *dates* est la liste des dates concernées ; ce sont les périodes ayant une distribution des RTTs différentiels de taille plus grande d'un nombre donné.
2. génération des périodes ; nous générons les périodes correspondantes à au moins une journée ;
3. en partant des périodes générées, dans leurs ordre chronologique, nous appliquons la méthode *findAlarms()* sur les RTTs différentiels de chaque période ;
4. enfin, nous construisons un objet JSON reprenant les détails du lien. A savoir, leurs périodes, leurs anomalies et leurs dates d'anomalies.

Listing IV.30 – Définition de la méthode *listAlarms*

```

1  def listAlarms(spark : SparkSession, rawDataLinkFiltred : ResumedLink,
    timewindow : Int, rangeDates : Seq[Int]) : String = {
2      // Save the reference state of a link
3      var reference = LinkState(Seq(), Seq(), Seq(), Seq())
4
5      // Save the current state of a link
6      var current = LinkState(Seq(), Seq(), Seq(), Seq())
7
8      // Save the RTT differentials anomalies
9      var alarmsValues = AlarmsValues()
10
11     // Save the dates having delay anomalies
12     var alarmsDates = AlarmsDates()
13
14     // Save all the dates to draw the evolution
15     var dates = AllDates()
16
17     val rawDataLink = rawDataLinkFiltred
18
19     /* Regardless of the period specified in the inputs, the evolution
        is created for one or more days
20     * Eg : if the period is only 2 hours, the evolution is created for
        24 hours,
21     * and the begin date is the begin date given in inputs
22     * */
23     val start = rawDataLink.bins.min
24     val max = rawDataLink.bins.max
25     val differenceDays = (max - start) / 60 / 60 / 24
26     val end = start + ((differenceDays + 1) * 86400)
27
28     // Find all the bins in the selected days

```

```

29     val datesEvolution = start.to(end - timewindow).by(timewindow)
30
31     // For each bin, find the data (RTTs differentials) and find alarms
32     datesEvolution.foreach(f => findAlarms(spark, f, reference,
33         rawDataLink, current, alarmsDates, alarmsValues, dates))
34
35     // create a JSON string to save the results
36     implicit val formats = DefaultFormats
37     val linkEvolution = LinkEvolution(rawDataLink.link, reference,
38         current, alarmsDates.dates, alarmsValues.medians, dates.dates)
39     val linkEvolutionJsonStr = write(linkEvolution)
40     linkEvolutionJsonStr
41 }

```

La définition de la fonction *findAlarms* est donnée dans le Listing IV.31. Cette méthode s'applique sur les données d'un lien, son objectif est de comparer l'état courant du lien en question avec la référence suivant les trois cas détaillés dans l'étape II.5 du processus décrit dans la section II.4.3.

Listing IV.31 – Définition de la méthode findAlarms

```

1  def findAlarms(spark: SparkSession, date: Int, reference: LinkState,
2     dataPeriod: ResumedLink, current: LinkState, alarmsDates: AlarmsDates,
3     alarmsValues: AlarmsValues, dates: AllDates): Unit = {
4     println("Find indices ...")
5     val indices = dataPeriod.bins.zipWithIndex.filter(_._1 == date).map
6     (_._2)
7     val dist = indices.map(f => dataPeriod.rttDiffs(f))
8
9     println("Find RTTs for the current timewindow ...")
10    val distSize = dist.size
11
12    if (distSize > 3) {
13        val tmpDates = dates.dates :+ date
14        dates.dates = tmpDates
15
16        // Compute the Wilson Score
17        val wilsonCi = scoreWilsonScoreCalculator(spark, dist.size)
18        .map(f => f * dist.size)
19
20        //update the current link state
21        updateLinkCurrentState(spark, dist, current, wilsonCi)
22
23        //Sort the distribution
24        val newDist = dist.sorted
25
26        //Get the reference
27        val tmpReference = reference
28
29        // Case : 1
30        if (tmpReference.valueMedian.size < 3) {
31            val newReferenceValueMedian = tmpReference.
32                valueMedian :+ current.valueMedian.last
33            val newReferenceValueHi = tmpReference.valueHi :+
34                newDist(javatools.JavaTools.getIntegerPart(
35                    wilsonCi(1)))
36            val newReferenceValueLow = tmpReference.valueLow :+
37                newDist(javatools.JavaTools.getIntegerPart(
38                    wilsonCi(0)))

```

```

30         reference.valueHi = newReferenceValueHi
31         reference.valueLow = newReferenceValueLow
32         reference.valueMedian = newReferenceValueMedian
33
34     } // Case : 2
35     else if (reference.valueMedian.size == 3) {
36
37         val newReferenceValueMedian1 = tmpReference.
38             valueMedian :+ medianCalculator(tmpReference.
39                 valueMedian)
40         val newReferenceValueHi1 = tmpReference.valueHi :+
41             medianCalculator(tmpReference.valueHi)
42         val newReferenceValueLow1 = tmpReference.valueLow
43             :+ medianCalculator(tmpReference.valueLow)
44
45         reference.valueHi = newReferenceValueHi1
46         reference.valueLow = newReferenceValueLow1
47         reference.valueMedian = newReferenceValueMedian1
48
49         val newReferenceValueMedian = reference.valueMedian
50             .map(f => reference.valueMedian.last)
51         reference.valueMedian = newReferenceValueMedian
52         val newReferenceValueHi = reference.valueHi.map(f
53             => reference.valueHi.last)
54         reference.valueHi = newReferenceValueHi
55         val newReferenceValueLow = reference.valueLow.map(f
56             => reference.valueLow.last)
57         reference.valueLow = newReferenceValueLow
58     } // Case : 3
59     else {
60
61         val newReferenceValueMedian2 = tmpReference.
62             valueMedian :+ (0.99 * tmpReference.valueMedian.
63                 last + 0.01 * current.valueMedian.last)
64         val newReferenceValueHi2 = tmpReference.valueHi :+
65             (0.99 * tmpReference.valueHi.last + 0.01 *
66                 newDist(javatools.JavaTools.getIntegerPart(
67                     wilsonCi(1))))
68         val newReferenceValueLow2 = tmpReference.valueLow
69             :+ (0.99 * tmpReference.valueLow.last + 0.01 *
70                 newDist(javatools.JavaTools.getIntegerPart(
71                     wilsonCi(0))))
72         reference.valueHi = newReferenceValueHi2
73         reference.valueLow = newReferenceValueLow2
74         reference.valueMedian = newReferenceValueMedian2
75
76         // Anomalies dection : compare the current with the
77         // reference
78         if ((BigDecimal(current.valueMedian.last) -
79             BigDecimal(current.valueLow.last) > reference.
80                 valueHi.last - current.valueMedian.last + current.
81                 valueHi.last < reference.valueLow.last) &&
82             scala.math.abs(current.valueMedian.last -
83                 reference.valueMedian.last) > 1) {
84
85             val updateAlarmsDates = alarmsDates.dates
86                 :+ date

```



```

66         alarmsDates . dates = updateAlarmsDates
67
68         val updateAlarmsValues = alarmsValues .
           medians :+ current . valueMedian . last
69         alarmsValues . medians = updateAlarmsValues
70     }
71 }
72 }
73 }
```

Nous résumons le fonctionnement de la méthode *findAlarms* dans les étapes suivantes. Notons que *spark* est l'objet *SparkSession* créé au début du programme Spark, *date* est la période courante, *reference* est l'état référence du lien, *dataPeriod* est la liste des RTTs différentiels de la période *date*, *current* est l'état courant du lien, *alarmsDates* sont les dates d'alarmes, *alarmsValues* sont les RTTs différentiels médians et *dates* sont les dates où on constate une distribution assez représentable des RTTs différentiels.

1. trouver les indices, dans *dataPeriod.bins*, correspondants à la période *date* (*indices*);
2. trouver les RTTs différentiels, dans *dataPeriod.rttDiffs*, identifiés durant *date* (*dist*);
3. la représentativité d'une distribution dépend d'une valeur donnée (ici 3), Si la taille de *dist* est supérieure à cette valeur, nous ajoutons *date* à *dates* et nous continuons l'analyse de cette période;
4. calculer le score de Wilson, qui fournit deux valeurs, et multiplier ce score par la taille de la distribution;
5. calculer l'état courant du lien (*current*), cela inclut la borne inférieure de l'intervalle de confiance, le RTT différentiel médian et la borne supérieure de l'intervalle de confiance;
6. récupérer l'état référence du lien;
7. mettre à jour l'état référence du lien *reference* suivant les trois cas;
8. comparer les deux intervalles de confiance s'il s'agit du cas 3 et mettre à jour *alarmsValues* et *alarmsDates* si une anomalie est identifiée.

## IV.2.4 Exécution d'une application Spark

Afin de pouvoir exécuter une application Spark, il faut qu'elle soit packagée dans un fichier de type JAR. Ce dernier doit reprendre une classe contenant une méthode *main* et doit reprendre toutes les dépendances nécessaires à l'exécution de l'application. Enfin, l'application Spark est soumise avec la commande *bin/spark-submit*. Un exemple d'une soumission est donné dans [IV.32](#).

Listing IV.32 – Exemple de la soumissions d'un traitement sur Spark

```

1 ~$ bin/spark-submit --class ripeatlasanalysis.AnalyseTraceroute
   --master local --driver-memory 30G --conf "spark.network.timeout
   =10000000" DelayAnalysis-0.0.5-SNAPSHOT-jar-with-dependencies.jar
   1517961600 1518134400 3600
```

La commande *bin/spark-submit* prend plusieurs paramètres, nous présentons quelques paramètres utilisés :

- *class* est un objet Scala contenant la fonction *main*;

- *master* est l'URL du cluster (voir les différents modes dans la section [III.4.5](#));
- *driver-memory* est la mémoire dont le processus du driver peut utiliser;
- *–conf "key = value"* est une manière de configurer l'application Spark. Dans l'exemple, "spark.network.timeout=10000000", 10000000 est le temps durant lequel le driver doit recevoir des mises à jour de la part des différents workers; après ce temps, le worker n'est plus considéré comme actif.

### Résultats finaux

Nous sauvegardons le résultat de l'analyse dans un fichier JSON en vue de toute réutilisation. Un exemple de résultat est illustré dans le Listing [IV.33](#).

Listing IV.33 – Exemple des résultats de l'analyse d'un lien

```

1 {
2     "link": {
3         "ip1": "185.147.12.31",
4         "ip2": "89.105.200.57"
5     },
6     "reference": {
7         "valueMedian": [
8             2.991,
9             2.991,
10            2.991,
11            2.991,
12            13.67572,
13            13.5727878
14        ],
15        "valueHi": [
16            4.402,
17            4.402,
18            4.402,
19            4.402,
20            15.352609999999999,
21            15.243713899999998
22        ],
23        "valueLow": [
24            2.394,
25            2.394,
26            2.394,
27            2.394,
28            12.83469,
29            12.7333531
30        ],
31        "valueMean": []
32    },
33    "current": {
34        "valueMedian": [
35            2.991,
36            3.081,

```

```
37         2.9109999999999996,  
38         823.963,  
39         1071.463,  
40         3.3825000000000003  
41     ],  
42     "valueHi": [  
43         0.472,  
44         1.321,  
45         1.7240000000000004,  
46         175.5,  
47         28,  
48         1.0804999999999998  
49     ],  
50     "valueLow": [  
51         10.741,  
52         0.39,  
53         0.5169999999999996,  
54         142.5,  
55         25,  
56         0.6815000000000003  
57     ],  
58     "valueMean": []  
59 },  
60     "alarmsDates": [  
61         1514784200,  
62         1514787800  
63     ],  
64     "alarmsValues": [  
65         1071.463,  
66         3.3825000000000003  
67     ],  
68     "dates": [  
69         1514769800,  
70         1514773400,  
71         1514777000,  
72         1514780600,  
73         1514784200,  
74         1514787800  
75     ]  
76 }
```

# Chapitre V

## Application de quelques technologies Big Data sur l'analyse des traceroutes

### V.1 Introduction

Ce chapitre reprend un ensemble de technologies destinées à la manipulation des données massives. Ce sont les technologies que nous avons expérimenté pour analyser des traceroutes disponibles dans le dépôt de RIPE Atlas. Précisément, ce sont les traceroutes permettant de tracer l'évolution du délai d'un lien comme c'est détaillé dans le chapitre III. Les technologies que nous présentons couvrent les besoins d'une ou de plusieurs étapes d'un processus d'analyse de données.

### V.2 Critères d'évaluation des technologies Big Data

Les critères d'évaluation d'une technologie Big Data varient suivant son objectif : stockage, calcul, etc. En générale, la liste des critères que l'on peut considérer dans la comparaison des technologies Big Data est très longue. Les critères sur lesquels nous évaluons les différentes technologies Big Data expérimentées sont les suivants :

- facilité de la mise en route et de la configuration de l'environnement de la technologie ;
- flexibilité liée à la définition du schéma de données présentes dans les fichiers ;
- temps d'exécution nécessaire pour fournir les résultats finaux d'une analyse de traceroutes ;
- évolutivité de l'environnement Big Data mis en place pour des nouvelles données et de nouveaux besoins.

Dans la présente évaluation de quelques technologies Big Data, nous n'avons pas pris en compte d'autres critères. Car nous ne pouvons pas les évaluer. Par exemple, l'utilisation du Big Data engendre des coûts liés aux ressources nécessaires au stockage de données massives ainsi qu'au traitement de ces dernières. Nous avons donné des indications théoriques concernant les frais d'utilisation de deux technologies dédiées au stockage de données massives : Amazon S3 (voir le Tableau III.1) et MongoDB Atlas dont les frais d'utilisation dépendent de plusieurs paramètres<sup>1</sup>.

---

1. Une estimation est possible suivant le fournisseur de cloud, elle est disponible sur <https://www.mongodb.com/cloud/atlas/pricing>, consulté le 25/12/2018.

## V.3 Caractéristiques de l’environnement de test

**La machine de test** L’évaluation des technologies Big Data choisies sur un échantillon de traceroutes a été réalisé sur un conteneur de type OpenVZ ayant les caractéristiques suivantes : système Debian GNU/Linux 7.11 (wheezy), 32768 MB de RAM, CPU MHz 2294.331.

Il existe différentes catégories de virtualisation. **OpenVZ** s’inscrit dans la catégorie Iso-lateur. Un isolateur est un logiciel permettant d’isoler l’exécution des applications dans des contextes ou zones d’exécution. Un conteneur OpenVZ adopte un partitionnement logique au niveau des ressources systèmes : processus, réseau et système de fichier<sup>a</sup>.

<sup>a</sup>. Source : [http://cesar.resinfo.org/IMG/pdf/jtsiars-openvz\\_1\\_.pdf](http://cesar.resinfo.org/IMG/pdf/jtsiars-openvz_1_.pdf), consultée le 29/12/2018.

Les différents tests effectués, présentés dans le présent chapitre, ont été effectués au sein de cette machine. Nous notons qu’un seul test est lancé à un moment donné dans la machine.

**Paramètres de l’analyse** Pour les paramètres de détection, nous avons utilisé les valeurs suivantes : *timeWindow* est de 3600 secondes, l’intervalle de confiance est de 0,05, *alpha* est de 0,01 et *minSeen* est de 3. Les dates de début et de fin varient suivant les traceroutes analysés.

## V.4 Application 1 : MongoDB

### Evaluation des critères de sélection

MongoDB est la technologie Big Data utilisée par Fontugne et al. dans l’implémentation de l’outil de détection [16]. Dans MongoDB, les traceroutes sont organisés dans des collections. Chaque collection stocke les traceroutes effectués lors de la journée *YYYY\_MM\_DD* et en adressage *V*. Par convention, *V* vaut 6 s’il s’agit de l’adressage IPv6 et est vide. La nomenclature des collections permet de ne récupérer que les traceroutes concernés par l’analyse lancée. Le nom d’une collection est structuré comme suit : *tracerouteV\_YYYY\_MM\_DD*.

MongoDB est une technologie conçue pour assurer le stockage de données dans un processus d’analyse de données. Nous avons utilisé la version locale de MongoDB, la quantité de données que nous pouvons stocker ainsi que le traitement appliqué sur les données récupérées dépendent principalement des ressources de la machine dans laquelle MongoDB est installé.

MongoDB est flexible en terme de définition du schéma de données ; aucun schéma n’est requis. Par exemple, dans certains cas, les traceroutes planifiés ne réussissent pas à atteindre une destination, dans ce cas, la structure de ces traceroutes est différente de la structure des traceroutes réussis. Les deux types de traceroutes sont stockés sans contrainte.

Les données stockées dans une collection MongoDB peuvent être manipulées en mode lecture et en mode écriture. Dans le premier, on cherche à lire des données en provenance de différentes sources. C’est le plus répandu dans les projets Big Data. Pour le deuxième mode, on peut mettre à jour un enregistrement dans une collection MongoDB. Ceci est moins fréquent dans les projets Big Data.

MongoDB est évolutif ; en cas de mise à jour de la structure de nouveaux objets traceroutes par Atlas, cela n’affecte pas les données précédemment stockées dans MongoDB.

### Performances de la base de données MongoDB dans l’analyse des délais

Nous mesurons le temps écoulé durant l'analyse des traceroutes, stockés dans une base de données MongoDB, en vue de détecter les anomalies dans le délai des liens. C'est le temps nécessaire à l'accomplissement des étapes de la phase I, de la phase II et de l'écriture des résultats dans un fichier. Chaque ligne de ce dernier décrit un lien comme l'exemple donné dans le Listing IV.33.

Dans le Tableau V.1, nous varions l'ensemble de traceroutes. Pour Chaque période, nous mesurons le temps nécessaire pour analyser les traceroutes capturés durant cette période pour plusieurs reprises. L'analyse de chaque période est fait 5 fois, ce qu'on appelle ici des essais : Essai 1, Essai 2, etc. Les traceroutes analysés sont ceux à destination des instances du f.root-servers.net<sup>2</sup>.

Période	Taille (bytes)	Essai 1 (s)	Essai 2 (s)	Essai 3 (s)	Essai 4 (s)	Essai 5 (s)	Médiane (s)
07/02/18 – 07/02/18	1, 028, 343, 572						
07/02/2018 – 08/02/2018							
07/02/2018 – 09/02/2018							
07/02/2018 – 10/02/2018							
07/02/2018 – 11/02/2018							

TABLE V.1 – Les temps d'exécution d'analyse de traceroutes en fonction de la taille de données avec MongoDB

Nous reprenons les informations du Tableau V.1 dans la Figure V.1. L'axe des abscisses représente la taille des fichiers contenant les traceroutes analysés, appelée  $q$ . L'axe des ordonnées représente le temps nécessaire à l'analyse d'une quantité de traceroutes. Nous agrégeons les temps des différents essais et nous calculons leur valeur minimale, maximale et la médiane. Pour précision, le temps calculé est la différence entre l'instant qui précède le lancement de l'analyse et l'instant qui suit la fin de l'analyse.

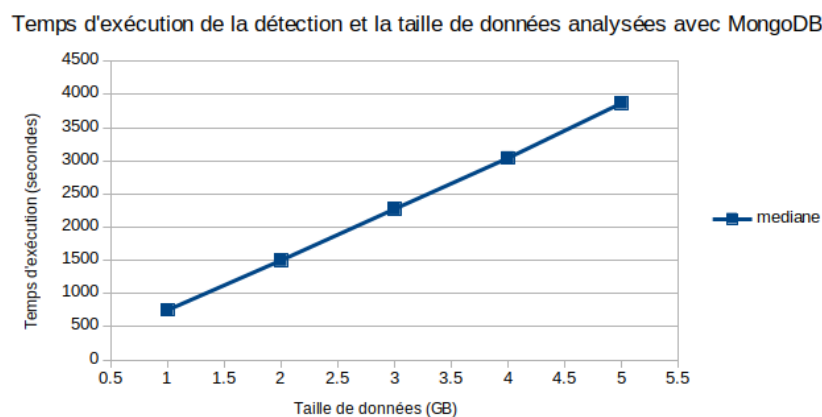


FIGURE V.1

## V.5 Application 2 : Amazon DynamoDB

L'élasticité est une des caractéristiques attirantes des services web d'Amazon. En particulier, c'est le cas d'Amazon DynamoDB. Ainsi, une implémentation basée sur Amazon Dy-

2. Voir les détails de la mesure 5004 sur <https://atlas.ripe.net/measurements/5004/>, consultée le 12/12/2018

namoDB n'a pas à se soucier de la capacité de stockage de données si la quantité de données évolue rapidement.

Amazon DynamoDB n'assure que le stockage de données dans un processus d'analyse de données. La récupération et le traitement des données stockées nécessitent l'ajustement des ressources de la machine qui reçoivent ces données, pareillement à MongoDB. La différence se situe à l'évolutivité implicite du stockage de données, qui ne se limite que par la capacité de stockage physique d'AWS. Tandis qu'une installation locale de MongoDB est liée aux ressources de la machine hébergeant ce dernier. Nous n'avons pas expérimenté Amazon DynamoDB pour analyser les traceroutes, étant donné que notre évaluation des temps d'exécution est effectuée sur une machine locale, nous aurons les mêmes remarques que dans le cas de MongoDB en ce qui concerne l'ajustement des ressources de la machine qui reçoive les données.

A titre indicatif, une heure de tous les traceroutes effectués par toutes les sondes Atlas, concernant tous les identifiants de mesure, fait une taille moyenne de 620 MB en format compressé, ce que représente une quantité d'environ 9 GB en format texte.

## V.6 Application 3 : Amazon S3, Amazon Glue et Amazon Athena

### Vue générale

Nous avons combiné les trois services d'Amazon (Amazon S3, Amazon Glue et Amazon Athena) afin de créer un environnement d'analyse de données massives. Un des scénarios possibles mettant en pratique ensemble ces trois services est illustré dans la Figure V.2<sup>3</sup>. Nous détaillons chaque services dans les sections suivantes.

Afin d'utiliser Amazon Athena pour l'interrogations des traceroutes stockés dans des fichiers, nous avons besoin d'abord de stocker les fichiers dans Amazon S3. De plus, nous avons besoin de créer un schéma de données. Il s'agit de créer une table comme les tables dans un SGBDR. Chaque enregistrement dans cette table correspond à une ligne dans les fichiers de données censés être lus par cette table. Il existe deux manières pour créer une table dans Athena : en utilisant Amazon Glue ou création manuelle. *traceroutes\_api* désigne le nom de la table reprenant tous les traceroutes.

### Création de la table *traceroutes* avec Amazon Glue

Nous avons lancé la détection automatique du schéma, avec Amazon Glue, d'un ensemble de traceroutes enregistrés dans un fichier faisant une taille de 500 MB. Toutefois, la détection a échoué. Autrement dit, Amazon Glue n'a pas pu inférer le schéma d'une seule table capable de lire tout traceroute dans ce fichier. L'échec de l'inférence est dû au fait que le fichier contient des traceroutes différents en terme de structure, car la structure dépend de la version du firmware de la sonde ayant effectué le traceroute. Les différentes versions du firmware pour chaque type de mesure sont détaillées dans le site Web d'Atlas<sup>4</sup>.

### Création manuelle de la table *traceroutes\_api*

Nous avons créé la structure de la table *traceroutes\_api* manuellement en se basant sur la structure détaillée d'une réponse traceroute pour chaque version du firmware. Les différentes

---

3. Amazon Redshift est un entrepôt de données et Amazon Quicksight est un service cloud d'informatique décisionnelle.

4. [https://atlas.ripe.net/docs/data\\_struct/](https://atlas.ripe.net/docs/data_struct/), consultée le 16/01/2018.

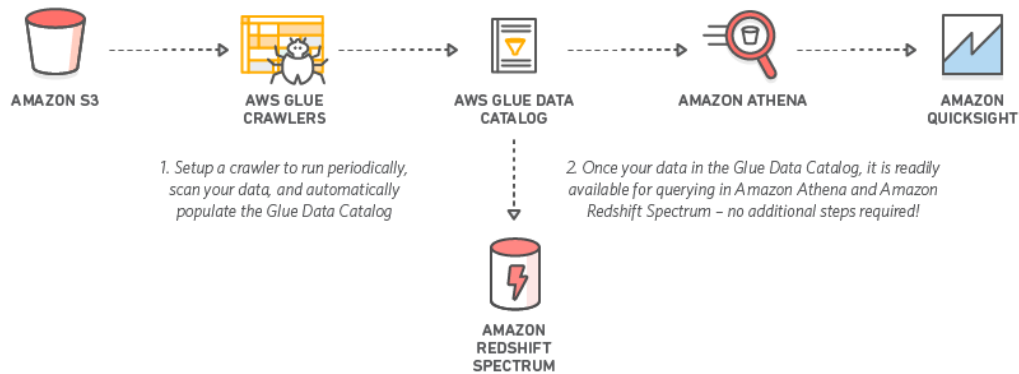


FIGURE V.2 – Une combinaison des services web d'Amazon : Amazon S3, Amazon Glue, Amazon Athena, Amazon Quicksight et Amazon Redshift

Source : [https://docs.aws.amazon.com/fr\\_fr/athena/latest/ug/glue-best-practices.html](https://docs.aws.amazon.com/fr_fr/athena/latest/ug/glue-best-practices.html), consultée le 16/05/2018.

structures de réponses d'une requête traceroute n'a posé aucun problème dans la création manuelle de la table. Dans notre cas, la réussite de la création manuelle est due au fait que les attributs dont l'outil de détection a besoin sont présents dans toutes les versions du firmware d'une part. D'autre part, Amazon Athena est flexible en ce qui concerne l'association entre un objet JSON et un enregistrement dans une table. Autrement dit, si un attribut existe dans l'objet JSON, la colonne correspondante prend sa valeur et vide dans le cas échéant.

### Partitionnement des données stockés dans Amazon S3

Nous avons pris en compte le partitionnement de données dans un compartiment S3 dans la création de la table *traceroutes\_api*. L'utilisation du partitionnement est optionnel. Le partitionnement de données présentes dans un compartiment Amazon S3 permet de limiter la quantité de données à analyser par une requête Amazon Athena. Le partitionnement améliore les performances d'Amazon Athena. D'une part, la requête s'exécute rapidement. D'autre part, le partitionnement réduit les coûts engendrés suite à l'utilisation d'Amazon Athena, car ce dernier est facturé selon la quantité de données analysées. En pratique, une partition créée joue un rôle similaire à celui d'une colonne durant l'interrogation d'une table dans Athena.

Prenons un exemple illustrant l'apport du partitionnement. Nous avons des traceroutes effectué en adressage IP la version 4 et 6.

*af\_* désigne le type d'adressage : *af\_* vaut 4 en cas d'adressage IPv4 et 6 en cas d'adressage IPv6. Sans l'utilisation du partitionnement et si on ne souhaite récupérer que les traceroutes ayant comme adressage IPv4, tous les traceroutes présents dans le compartiment S3 (appelé *s3://ripeatlasdata*), dédié au stockage des traceroutes récupérés depuis le dépôt d'Atlas, sont évalués<sup>5</sup>.

Toutefois, en partitionnant les données suivant par exemple le type d'adressage, seuls les fichiers dans la partition<sup>6</sup> *af\_* = 4 qui sont analysés. Par conséquent, le partitionnement permet de réduire les coûts d'utilisation du service Amazon Athena, surtout si la quantité de données est très importante.

5. L'évaluation du type d'adressage est effectué selon la valeur de l'attribut *af* d'un traceroute, il vaut 4 ou 6.

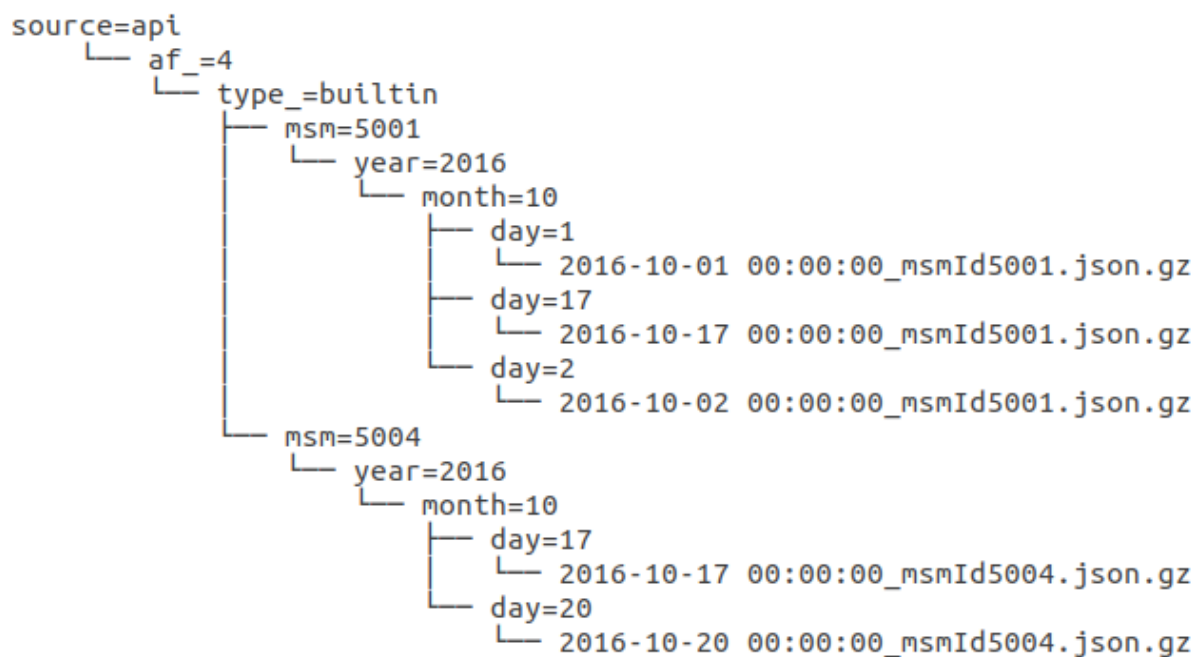
6. Partition dans le sens d'Amazon Athena.



partition	Valeurs	Commentaires
source	api	Les traceroutes récupérés depuis le dépôt d'Atlas via l'API
	typeanddate	Les traceroutes récupérés depuis la page Web
af_	4	Les traceroutes en adressage IPv4
	6	Les traceroutes en adressage IPv6
type	builtin	Les traceroutes en provenances des mesures intégrées
	anchor	Les traceroutes à destinations des ancres
msm	5001	Les traceroutes ayant msm_id = 5001
	5004	Les traceroutes ayant msm_id = 5001
year	2016	Les traceroutes effectués en 2016
month	10	Les traceroutes effectués en octobre
day	1	Les traceroutes effectué le premier du mois

TABLE V.2 – Exemple des partitions créées dans un compartiment Amazon S3

Les partitions créées sont illustrées dans la Figure V.3. Nous détaillons les différentes partitions dans le Tableau V.2. Nous donnons le nom de la partition dans la première colonne, quelques valeurs de chaque partition dans la deuxième colonne et la troisième colonne de ce tableau reprend une description de la partition.

FIGURE V.3 – L'organisation des traceroutes dans le compartiment Amazon S3  
s3://ripeatlasdata

Les partitions *af\_* et *type\_* sont nommée de cette manière, au lieu de *af* et *type* car la table *traceroutes\_api* contient des colonnes avec ces noms et comme les partitions agissent comme des colonnes lors de l'évaluation d'une requête avec Amazon Athena, les noms de ces partitions ont été adaptés.

Par exemple, les traceroutes qui se trouvent dans le fichier

*2016-10-01 00:00:00\_msmId5001.json.gz* sont analysés par toute requête Athena impliquant les partitions d'une des manières suivantes : (source = api) ou (source = api et af\_ = 4) ou

(source = api et af\_ = 4 et type = builtin) ou (source = api et af\_ = 4 et type = builtin et msm = 5001) ou (source = api et af\_ = 4 et type = builtin et msm = 5001 et year = 2016) ou (source = api et af\_ = 4 et type = builtin et msm = 5001 et year = 2016 et month = 10) ou (source = api et af\_ = 4 et type = builtin et msm = 5001 et year = 2016 et month = 10 et day = 1).

### Interrogation des données Avec Amazon Athena

Une fois les fichiers de données synchronisés vers le compartiment AWS S3 et le schéma de données créé, on passe à l'interrogation de données en utilisant les requêtes SQL basées sur Presto. Nous donnons un exemple d'une requête Athena dans la section A.2.3 de l'annexe A.

### Intégration d'Amazon Athena dans l'outil de détection

Pour intégrer Amazon Athena dans l'outil de détection [16], on distingue deux possibilités. La première possibilité n'utilise Athena que pour récupérer les traceroutes stockés dans Amazon S3 vers la machine locale. Ensuite, cette dernière poursuit les traitements décrits dans la phase I et II. Dans ce cas, nous ne profitons pas des performances d'Amazon Athena vu que les traitements complexes sont effectués dans la machine locale.

Tandis que la deuxième possibilité vise la maximisation des traitements des deux phases I et II au sein de l'infrastructure d'Athena. De ce fait, la machine locale n'a qu'à recevoir les derniers résultats de la détection, voire les résultats finaux. Pour cette deuxième possibilité, les données doivent être manipulées de sorte à maximiser, au niveau d'Amazon Athena, les traitements relatifs à toutes les étapes des deux phases I et II.

Pour la deuxième possibilité, le défi est de trouver la requête ou bien l'ensemble de requêtes SQL à exécuter sur Athena en vue d'avoir l'évolution du RTT différentiel des liens. Vu la complexité des étapes des phases I et II, on ne peut pas trouver une seule requête SQL assurant toutes ces étapes à la fois. Supposons qu'il existe une requête SQL capable de trouver les liens possibles avec leurs RTTs différentiels : à l'étape 4 dans ??, on construit la distribution des RTTs différentiels pour tout lien  $l$  identifié dans les traceroutes de la période  $d_k$ . Cette distribution est mise à jour à chaque fois  $l$  est identifié dans un des traceroutes de la période  $d_k$ .

Soient  $T_k = \{t_{k,j}\}$  l'ensemble de traceroutes effectués durant  $d_k$ , avec  $j \in [1, R_k]$  et  $R_k$  est le nombre de traceroutes effectués durant  $d_k$ . Nous décrivons le parcours des traceroutes d'une période  $d_k$  brièvement dans le pseudo-code 3. Nous n'avons pas donné tous les détails, car l'objectif est d'évaluer la convenance d'Athena au traitement souhaité.

---

#### Algorithm 3 Une partie de l'étape 4 du processus de la détection des anomalies des délais

---

```

1 : for all  $t_{k,j} \in T_k$  do
2 :    $links \leftarrow \text{getLinksFromTraceroute}(t_{k,j})$ 
3 :   for all  $l \in links$  do
4 :      $\text{updateLinkRttDistribution}(l)$ 
5 :   end for
6 : end for

```

---

Avec :

- $\text{getLinksFromTraceroute}(t_{k,j})$  énumère tous les liens possibles dans le traceroute  $t_{k,j}$ .
- $\text{updateLinkRttDistribution}(l)$  ajoute le RTT différentiel calculé du lien  $l$  à la distribution des RTTs différentiels courante de ce lien pour la période  $d_k$ .

Le service Athena est conçu pour la lecture de données, toute mise à jour de données n'est pas possible avec ce service. C'est pourquoi la distribution des RTTs différentiels de chaque lien identifié doit être sauvegardée dans un endroit accessible en lecture et en écriture, par exemple dans un compartiment AWS S3. Que ce soit un fichier reprenant la distribution des RTTs différentiels par un seul lien ou bien un fichier pour tous les liens, à la ligne 4 du pseudo-code 3, un fichier doit être lu et mise à jour avec de nouvelle valeur. Pour une période  $d_k$  d'une heure, le nombre de traceroutes est de l'ordre de milliers. Chaque traceroute  $t_{k,j}$  peut inclure  $L_{k,j}$  liens. Dans ce cas, le nombre total, d'une période  $d_k$ , de mise à jour de la distribution des RTTs différentiels est  $\sum_{m=1}^{R_k} L_{k,m} \cdot L_{k,j}$  dépend du nombre de saut du traceroute  $t_{k,j}$ .

En plus du nombre de lectures et d'écritures, relatives à la phase I, que nous venons de décrire, à la phase II, la détection des anomalies s'effectue en comparant les intervalles de confiances : un intervalle de confiance courant du lien avec celui de référence. Cette comparaison révèle deux contraintes. La première contrainte concerne la fonction permettant de calculer les deux bornes de l'intervalle de confiance de Wilson ne fait pas partie des fonctions disponibles sur Amazon Athena. D'autre part, Amazon Athena ne permet pas la création des fonctions personnalisées pour répondre à des besoins non couverts par Amazon Athena. La deuxième contrainte concerne la mise à jour de l'intervalle de confiance de référence qui doit être faite à chaque nouvelle période.

### Evaluation des critères pour Amazon S3 et Amazon Athena

Afin d'utiliser le service Amazon Athena à moindre coût, il est conseillé d'utiliser le partitionnement, car moins de frais sont appliqués. Si un partitionnement particulier est adopté, la création du schéma de données est basé sur ce partitionnement ainsi que les requêtes SQL destinés à l'interrogation de la table de données.

En ce qui concerne l'évolutivité d'une application basée sur ces deux services d'Amazon, on note que toute mise à jour de la structure de données des objets traceroutes peut affecter l'entièreté de la configuration initiale. A savoir, l'organisation des fichiers de données via le partitionnement, le schéma de données et les requêtes SQL.

Quant à la flexibilité du schéma de données, le service Amazon Athena est tolérant au données manquantes. Etant donné que la structure d'un objet traceroute dépend de la version du firmware de la sonde, nous avons créé trois schémas de tables. La première table modélise tout objet traceroute de version 5, la deuxième modélise tout objet traceroute de version 6 et enfin la troisième table modélise ceux ayant la version 7. En expérimentant différentes requêtes, nous avons conclu que Amazon Athena a pu récupérer les données de la version récente (7) via le schéma de la version 5 malgré que la version 7 a plus d'attributs par rapport à la version 5.

### Performances des services Amazon S3 et Athena dans l'analyse des délais

Nous avons utilisé Amazon Athena et Amazon S3 pour analyser les traceroutes et détecter les anomalies des délais. Nous précisons que nous avons évalué la première possibilité décrite dans la section V.6. Nous avons bénéficié de la possibilité de lancer des requêtes destinées à Amazon Athena à travers l'API REST, précisément en Python. Et comme l'implémentation proposée par les auteurs du travail de référence est écrite en Python, nous avons adapté cette implémentation de sorte de récupérer les traceroutes depuis Amazon S3 via Amazon Athena au lieu de le faire depuis la base de données locale MongoDB.

Etant donné que nous avons utilisé le partitionnement de données, une analyse des délais nécessite d'autres paramètres à ajuster en plus de ceux relatifs à la détection. Ce sont les paramètres permettant de sélectionner les traceroutes présents sur Amazon S3. Du fait que le partitionnement de données (voir une partie de l'arborescence dans la Figure A.1) est réalisé

sur base du type de traceroute (*builtin* ou *anchor*) et de l'identifiant de la mesure (5004, 6001, etc) qui a enregistré un traceroute, nous devons personnaliser la requête visant la récupération des traceroutes depuis Amazon S3 pour qu'elle prennent en compte aussi les partitions.

Le Tableau V.3 contient les temps d'exécution suivant la taille de l'ensemble de données donné en entrée de la détection.

Période	Taille(GB)	Temps (secondes)
07/02/18 – 07/02/18	1	1898.31
07/02/2018 – 08/02/2018	2	3533.6562171
07/02/2018 – 09/02/2018	3	5284.91494989
07/02/2018 – 10/02/2018	4	7228.88
07/02/2018 – 11/02/2018	5	8984.873281

TABLE V.3 – Les temps d'exécution par taille de l'ensemble de données (Amazon Athena et Amazon S3)

Nous distinguons trois phases dans cette approche (approche 1). Premièrement, les données sont récupérées depuis Amazon S3. Plusieurs facteurs affectent cette étapes, par exemple, les conditions du réseau, les ressources allouées par Amazon pour répondre à chaque requête Athena à destination des données disponibles sur Amazon S3, l'optimalité de la requête SQL, etc. En deuxième lieu, les résultats de la requête doivent être désérialisés pour pouvoir les utiliser localement. Enfin, sur base des données récupérées, la détection des anomalies peut être déclenchée. La Figure V.4 présente un seul essai pour chacune des tailles utilisées auparavant avec MongoDB. Le temps de chaque essai comprend l'étape de la récupération des traceroutes depuis Amazon S3, le temps de préparation des traceroute et enfin le temps de la détection des anomalies. Autrement dit, le temps nécessaire à la réalisation des phases I, II et celui nécessaire pour sauvegarder les résultats par lien.

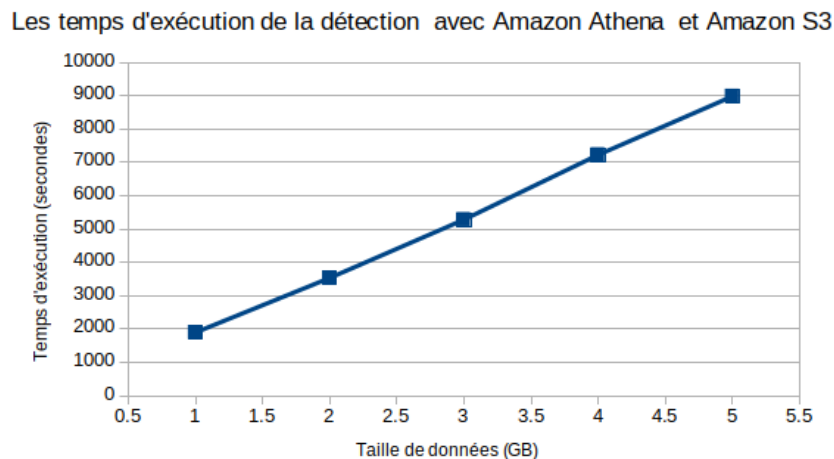


FIGURE V.4 – Les temps d'exécution de la détection des anomalies en fonction de la taille de données (Amazon S3 et Amazon Athena)

## V.7 Application 4 : Spark Apache avec Scala

Nous avons implémenté l'outil de détection avec le framework Spark et l'API Scala. Les détails de l'implémentation sont donnés dans le chapitre IV. Nous avons évalué le temps d'exé-

cution de l'outil de détection en analysant différents échantillons de traceroutes en mode local et en mode cluster. Pour le mode local, nous avons lancé l'application Spark sur la machine ayant les caractéristiques reprises dans la section V.3. Pour le mode cluster, nous avons utilisé le cluster EMR.

Les fonctionnalités de Spark sont accessibles avec les APIs en Scala, Java et Python. Nous avons choisi l'utilisation de l'API en Scala parce que Scala est le langage natif de Spark. De plus, Scala est interopérable avec Java.

### V.7.1 Mode Standalone

A la base Spark est conçu pour être utilisé dans un cluster de machines, sur lequel l'analyse est distribuée. Toutefois, Spark peut être utilisé en mode local. Dans ce mode, on trouve le *driver* et un seul *executor*. Ce dernier est "lié" au processus initié par le *driver*. Dans une application Spark, la taille de la mémoire allouée pour le *driver* et les *executors* est définie par défaut. D'après la documentation officielle de Spark <sup>7</sup>, Spark réserve 1 GB pour le *driver* et 1 GB pour chaque *executor*.

#### Performances d'Apache Spark dans l'analyse des délais

Nous avons évalué le temps d'exécution de l'implémentation de l'outil de détection en utilisant Spark en variant le nombre de traceroutes à analyser. Nous avons aussi varié certains paramètres relatifs à la soumission de l'application au Spark. Nous avons varié la taille de la mémoire allouée au driver afin de choisir celle la plus adaptée. Ensuite nous avons mesuré le temps d'exécution dans le cas de local, local[K] et local[\*].

---

7. Source : <https://spark.apache.org/docs/latest/configuration.html>, consultée le 29/12/2018.

### Variant la mémoire allouée au driver

En mode local (`--master local`), le *driver* et le *worker* sont liés au même processus. Nous avons mesuré le temps d'exécution de l'application Spark en variant la taille mémoire allouée au *driver* via le paramètre *driver-memory*. Par défaut, la mémoire allouée au driver est de 1 Go. Avec cette valeur, il n'est pas possible d'analyser un ensemble de traceroutes qui fait 1,028,343,572 octets. Afin de voir l'effet de la mémoire allouée au *driver*, en mode d'exécution local, nous avons utilisé deux échantillons de traceroutes. La première comprend les traceroutes capturés pendant le 07/02/2018 dans le cadre de la mesure ayant l'identifiant 5004, ce qui fait 1,028,343,572 octets (*data\_1go*). Pour le deuxième échantillon, il reprend les traceroutes effectués entre le 07/02/2018 et le 08/02/2018 effectués aussi dans le cadre de la mesure ayant l'identifiant 5004, ce qui fait 2,055,167,238 (*data\_2go*) octets.

La Figure V.5 présente les résultats obtenus. L'axe des abscisses indique la quantité de mémoire allouée au *driver* et l'axe des ordonnées représente le temps d'exécution de détection. Pour les mêmes quantités de mémoire allouée au *driver*, nous mesurons le temps d'exécution une fois pour *data\_1g* et une autre fois pour *data\_2g*. Pour les valeurs nulles relatives au temps d'exécution, l'exécution de la l'application a échoué. La raison de l'échec revient au manque de mémoire (message d'erreur est `OutOfMemoryError : Java heap space`).

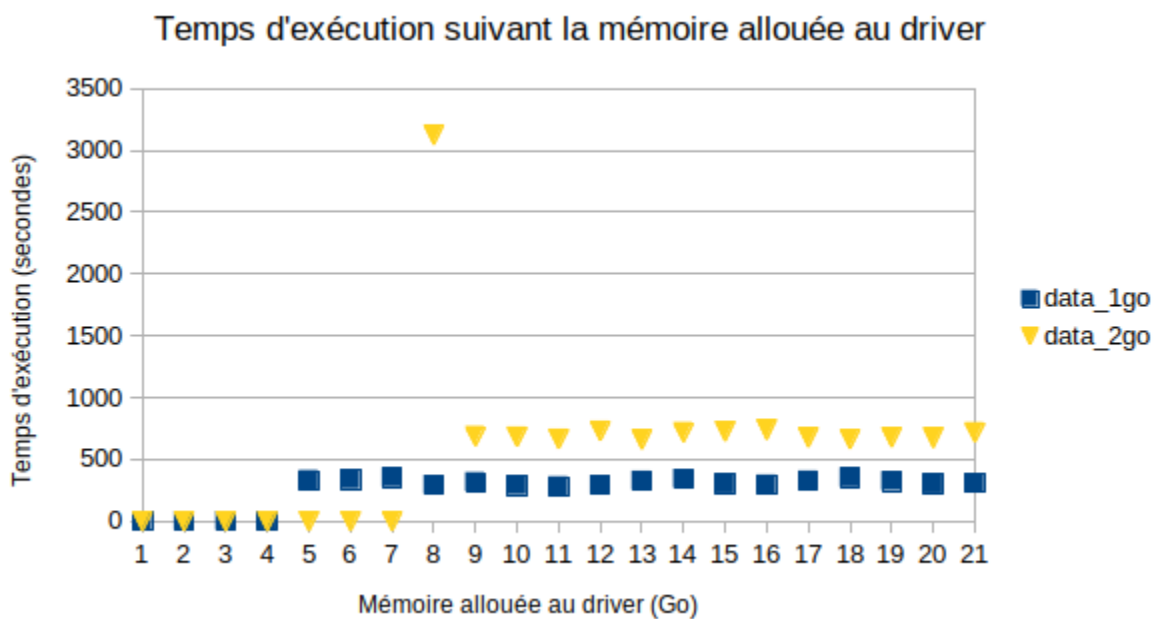


FIGURE V.5 – Mesure des temps d'exécution de l'application Spark selon différentes tailles de mémoire allouées au *driver* et pour deux ensembles de données différentes

D'après la Figure V.5, nous remarquons qu'à partir d'une taille mémoire allouée au driver, le temps écoulé durant l'exécution de l'application Spark est relativement stable. De plus, nous constatons qu'il faut prévoir une taille mémoire minimale pour assurer l'exécution de l'application Spark. Les tailles mémoire supérieures à cette valeur minimale affectent faiblement le temps d'exécution. Cette valeur minimale dépend fortement de la quantité de données à analyser. Enfin, malgré que la machine sur laquelle nous lançons l'application Spark ne dispose que de 32 Go de RAM, le fait d'allouer au driver 35 Go, 40 Go, 45 Go n'a pas généré une erreur lors de l'exécution.

**Variante la taille de données** Le Tableau V.4 contient les résultats de 5 analyses. Les trois premiers tests correspondent à l'analyse de 1 GB de traceroutes. Dans le quatrième test, nous avons analysé les données des trois premiers tests. Durant le cinquième test, l'analyse a échoué.

Test	Période	Durée	Taille	Temps (secondes)
1	07/02/2018	1 jour	1 GB	2498
2	08/02/2018	1 jour	1 GB	2942
3	09/02/2018	1jour	1 GB	2991
4	07/02/2018 – 09/02/2018	3 jour	3 GB	20955
5	07/02/2018 – 10/02/2018	4 jour	4 GB	échec

TABLE V.4 – Les temps d'exécution des analyses lancées sur des traceroutes (approche 1)

Le temps total nécessaire pour analyser les trois échantillons correspondants aux 07, 08 et 09 février 2018, chacun à part, est de 8431. Tandis que le temps nécessaire à l'analyse des traceroutes correspondants au 3 échantillons en une seule fois est de 20955, ce que correspond à 2.4 fois du temps de plus. En ce qui concerne l'analyse des traceroutes correspondants aux 4 jours, l'analyse a échoué.

En revenant à l'implémentation, pour chaque période  $d_k$ , il faut consulter tous les traceroutes et ne récupérer que ceux effectués durant  $d_k$ . Par exemple, 3 GB de données doivent être consultées  $3 * 24$  fois.

Nous avons modifié l'implémentation afin de réduire le temps de l'analyse. Au lieu de chercher les traceroutes à chaque période  $d_k$ , on attribue chaque traceroute à une des périodes de l'analyse si le timestamp correspond ; le timestamp du traceroute  $t_j$  appartient à la période  $d_k$ , appelée approche 2.

Soient  $d_k$  une période dans l'ensemble de périodes  $D$  entre  $start$  et  $end$  et  $t_j$  est un traceroute dans l'ensemble de traceroutes  $T$  donnés en entrée,  $j \in [1, n]$  où  $n$  est le nombre total de traceroutes à analyser. Nous illustrons la différence entre l'approche 1 et 2 avec les deux pseudo-code 4 et 5. Avec :

- *findAllPeriods(start, end, timeWindow)* : crée la liste des périodes entre la date *start* et la date *end*, chaque période a la durée *timeWindow* ;
- *loadTraceroutesFromInputFiles(dataPath)* : lit les traceroutes qui se trouvent dans le répertoire *dataPath* ;
- *checkTracerouteInPeriod(d\_k, t\_j)* : vérifie si le traceroute  $t_j$  a été effectué durant la période  $d_k$ .

---

**Algorithm 4** Regroupement des traceroutes par période  $d_k$  (étape 1) : approche 1

---

**Inputs :** *dataPath* chemin vers les fichiers de données, *start*, *end*, *timeWindow*

**Outputs :** regroupement des traceroutes par périodes

```

1:  $D \leftarrow \text{findAllPeriods}(start, end, timeWindow)$ 
2: for all  $d_k \in D$  do
3:    $T \leftarrow \text{loadTraceroutesFromInputFiles}(dataPath)$ 
4:   for all  $t_j \in T$  do
5:      $\text{checkTracerouteInPeriod}(d_k, t_j)$ 
6:   end for
7: end for
```

---



**Algorithm 5** Regroupement des traceroutes par période  $d_k$  (étape 1) : approche 2**Inputs :** *dataPath* chemin vers les fichiers de données, *start*, *end*, *timeWindow***Outputs :** regroupement des traceroutes par périodes

```

1:  $D \leftarrow findAllPeriods(start, end, timeWindow)$ 
2:  $T \leftarrow loadTraceroutesFromInputFiles(dataPath)$ 
3: for all  $t_j \in T$  do
4:   for all  $d_k \in D$  do
5:      $checkTracerouteInPeriod(d_k, t_j)$ 
6:   end for
7: end for

```

Le Tableau V.5 reprend des indications sur les résultats obtenus, avec *driver-memory* configuré sur 30 GB. Ce sont les temps d'exécution obtenus lors de deux tests pour les tailles de données : 1 GB, 2 GB, 3 GB, 4 GB et 5 GB, ce tableau reprend aussi les périodes correspondantes à ces données et le nombre de traceroutes présents dans chaque échantillon.

Période	Taille (GB)	Nb de traceroutes	Essai 1 (secondes)	Essai 2 (secondes)
07/02/18	1	494158	838	1137
07/02/2018 à 08/02/2018	2	987995	936	1034
07/02/2018 à 09/02/2018	3	1481942	1449	1930
07/02/2018 à 10/02/2018	4	1976010	2139	1714
07/02/2018 à 11/02/2018	5	2470027	3116	3030

TABLE V.5 – Les temps d'exécution lors de la détection des anomalies en utilisant Spark (approche 2)

La Figure V.6 permet de présenter les temps d'exécution pendant plusieurs essais et pour plusieurs ensembles de données.

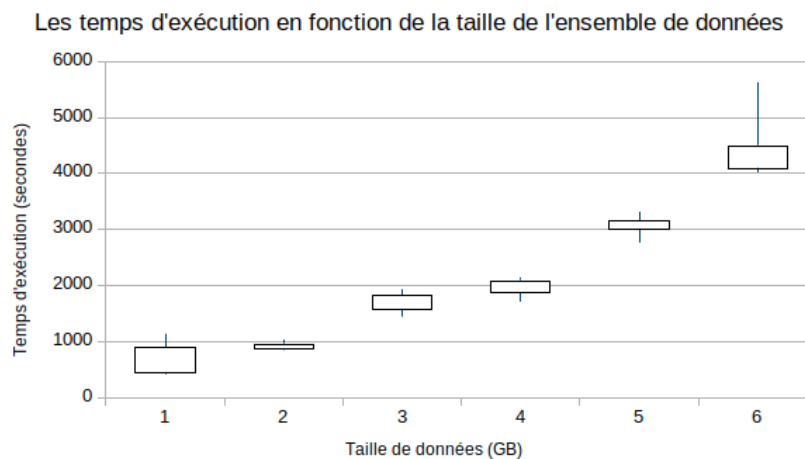


FIGURE V.6

## V.7.2 EMR Cluster

Nous avons choisi un cluster EMR pour lancer la détection des anomalies. En créant un cluster, nous pouvons choisir le nombre de *workers*. De plus, nous pouvons passer des arguments à notre application comme dans le mode local. Nous avons adapté la version destinée au



cluster EMR pour qu'elle prennent en compte les fichiers contenant les traceroutes stockés cette fois dans Amazon S3 au lieu de chercher ces fichiers dans l'arborescence de la machine locale.

## V.8 Récapitulatif

Nous avons discuté l'évaluation des technologies Big Data en terme du temps d'exécution. L'objectif de l'analyse des traceroutes est de détecter les changements anormaux dans les délais des liens à travers les changements de leurs RTTs différentiels. Nous avons évalué les temps d'exécution en fonction de la quantité de données, alors qu'on peut aussi évaluer les temps d'exécution en fonction du nombre de traceroutes. Du fait que l'outil de détection se base sur la liste des sauts par traceroute, et sachant que le nombre de saut varie d'un traceroute à un autre, une évaluation de la détection en fonction du nombre de traceroute peut être moins représentable. On ne peut pas traiter un ensemble de  $n$  traceroutes de la même manière étant donné que la détection des anomalies se base sur le nombre de sauts, les traceroutes n'ont pas tous un nombre de sauts équitable. C'est que illustre la Figure V.7<sup>8</sup>. Ainsi, le temps nécessaire à évaluer un traceroute ayant, par exemple, 10 saut est différent de celui nécessaire à l'évaluation d'un traceroute ayant 2 sauts.

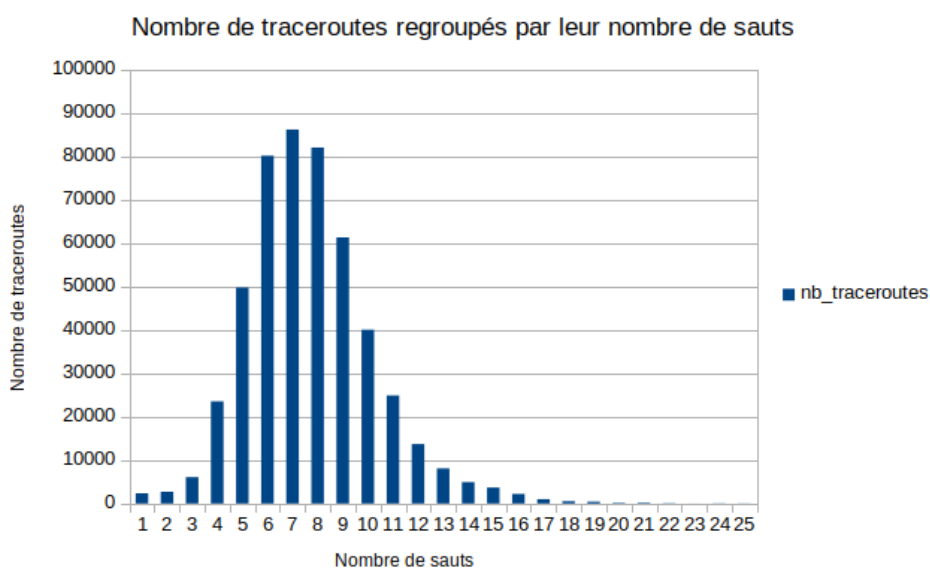


FIGURE V.7 – Regroupement des traceroutes d'une journée par leurs nombre de sauts

**Quelques chiffres sur les liens et les anomalies identifiés** Notre évaluation des technologies Big Data a concerné, en premier lieu, les temps d'exécution. Nous donnons à travers la Figure V.8 une idée sur le nombre de liens identifiés ainsi que les liens, parmi ceux identifiés, ceux ayant enregistré un moins une anomalie. La Figure V.8 présente ces liens par taille de données. Nous précisons que les traceroutes présents dans l'échantillon de 3 GB sont aussi repris dans l'échantillon de 4 GB.

8. Les résultats concernent les traceroutes de la journée 07/02/2018.

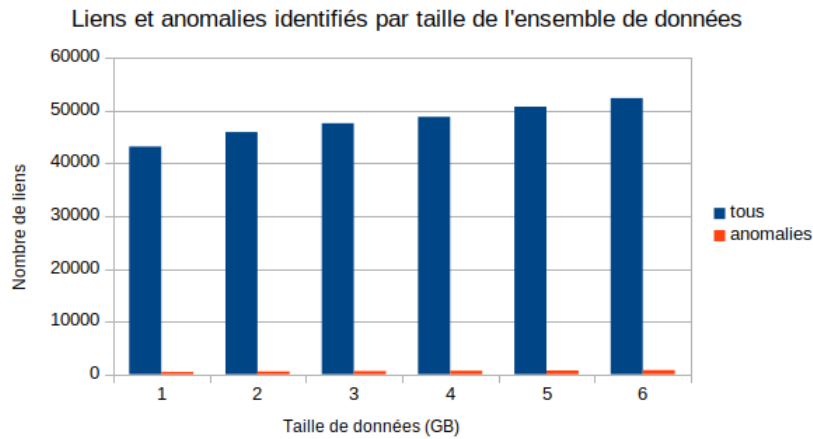


FIGURE V.8 – Les liens et les anomalies identifiés par ensemble de traceroutes

### MongoDB vs Amazon Athena et Amazon S3

L'évolution du temps d'exécution écoulé en utilisant Amazon Athena est différents de celui avec de MongoDB. Cela peut être dû aux facteurs discutés dans la section V.6. Nous précisons que l'implémentation de l'outil de détection nécessite de chercher les traceroutes à chaque période  $d_k$ . Nous avons considéré les résultats d'un seul identifiant (5004), le nombre de traceroutes récupérés à chaque  $d_k$  est plus de 20.000 (par requête avec un timeWindow égale à 3600 secondes).

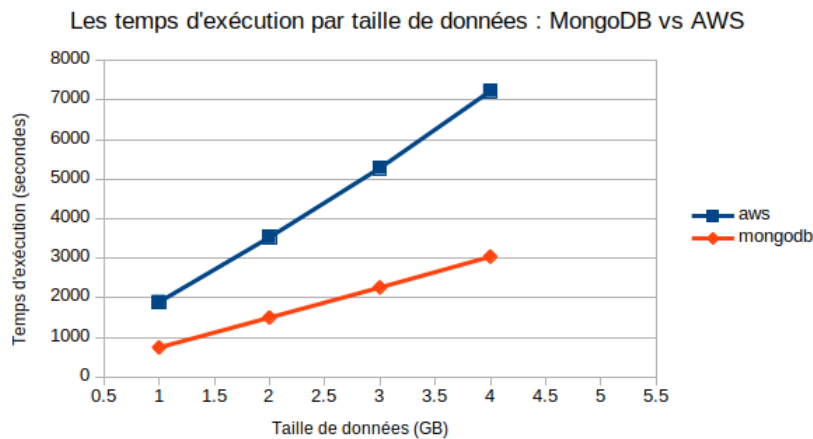


FIGURE V.9 – Comparaison de l'évolution des temps d'exécution entre MongoDB et Amazon Athena avec Amazon S3

### Apache Spark vs MongoDB

La Figure V.10 illustre la moyenne des temps d'exécution pour MongoDB et Spark. Ces résultats concernent les mêmes ensembles de données.

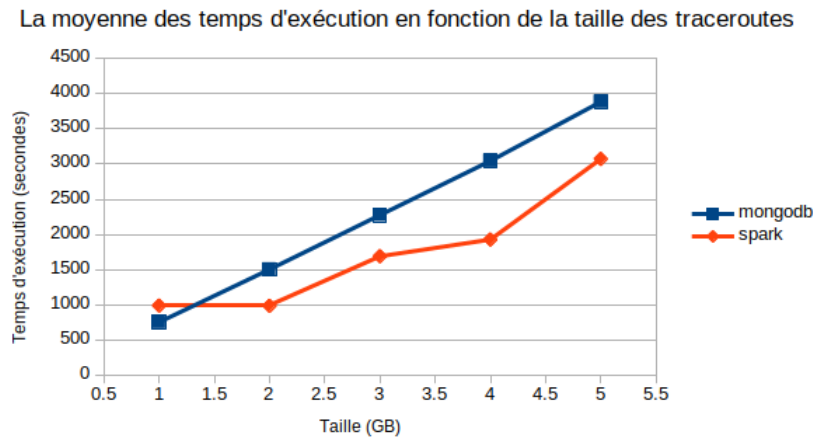


FIGURE V.10 – La moyenne des temps d'exécution pour MongoDB et Spark

**MongoDB vs Amazon Athena et Amazon S3 vs Apache Spark** A présent, nous comparons les temps d'exécution pour les trois implémentations : MongoDB, AWS, et Apache Spark comme illustré dans la Figure V.11.

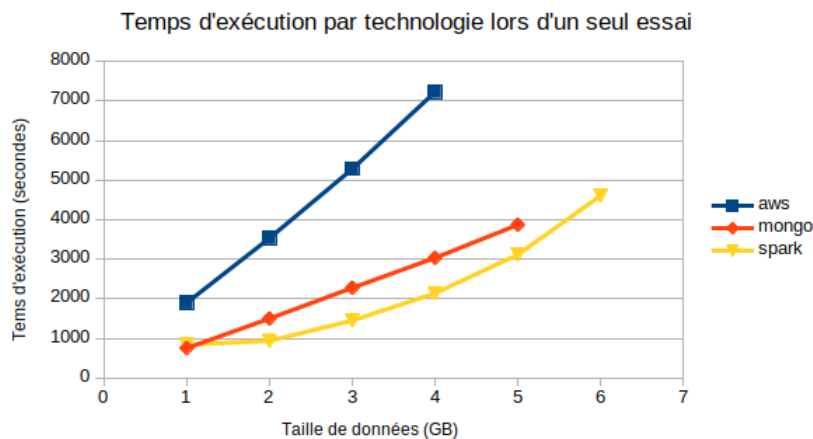


FIGURE V.11 – Comparaison des temps d'exécution entre MongoDB, AWS et Apache Spark

## V.9 Conclusion

Nous avons évalué la convenance des technologies Big Data à l'analyse souhaitée des traceroutes. Dans un premier temps, nous avons utilisé deux technologies conçues pour le stockage de données à grande échelle : MongoDB et DynamoDB, ensuite, nous avons expérimenté trois services d'Amazon, le premier pour le stockage de données, le deuxième pour la découverte de données et le troisième service est conçu pour l'interrogation de données. Enfin, nous avons utilisé un framework qui gère le traitement distribué de données dans un cluster de machines.

Nous avons à disposition une variété de données, à titre indicatif, nous pouvons récupérer des années de mesures effectuées par toutes les sondes Atlas. Notre premier objectif est d'évaluer la convenance des technologies choisies à l'analyse de données souhaitée. Précisément, pouvoir utiliser l'outil de détection avec ces technologies. C'est pourquoi nous n'avons pas défini des critères pour choisir l'ensemble de données de test. Nous avons évalué les performances

des technologies en terme de temps écoulé tout au long de l'analyse de différents échantillons en fonction de la taille de données. On peut conclure que les performances de MongoDB en terme de temps d'exécution, version locale, dépendent des ressources de la machine dans laquelle MongoDB tourne. Sachant qu'il existe une version disponible sur le cloud, les performances de celle-ci dépendent, à priori, des options choisies. Toutefois, les performances des trois services d'Amazon dépendent du nombre de requêtes destinées vers Amazon Athena, la complexité des traitements assurés par la requête Athena et aussi du partitionnement adopté. Tandis que Apache Spark, version local, dépendent de la taille de mémoire allouée pour le *driver* ainsi que d'autres configurations comme le *timeout* du *driver* et autres.

# Conclusion

L'objectif du présent travail est d'évaluer quelques technologies Big Data sur des données massives et réelles. Etant donné que l'évaluation des technologies Big Data peut prendre plusieurs formes, nous avons limité cette évaluation à la mise en place de la technologie ainsi qu'au temps d'exécution. Nous avons choisi des données dans le domaine des réseaux informatiques, disponibles sur le dépôt de RIPE Atlas, d'où l'intérêt de bien détailler ce projet.

Notre premier objectif est de montrer les limites des outils traditionnels à manipuler les données à grande échelle, et la force des technologies Big Data à répondre aux besoins liés au passage à l'échelle. Pour ce faire, nous avons choisi de réutiliser un travail dans lequel un outil de détection d'anomalies a été conçu, basé sur les données du RIPE Atlas.

D'après les quelques technologies expérimentées, il est très important d'avoir à l'avance les informations utiles à la prise de décision concernant une technologie proposée. Comme la fréquence de l'analyse, la fréquence de la génération de données, l'évolution de la quantité des nouvelles données générées dont la solution proposée doit en prendre en compte, les frais générés à l'utilisation de la technologie, le temps écoulé pour avoir les résultats finaux d'une analyse lancée, l'évolutivité de la solution mise en place, la disponibilité des outils de la visualisation des résultats et d'autres éléments. Ces critères reflètent les questions posées lors de la manipulation des technologies évaluées. Enfin, malgré que les évaluations des technologies Big Data ont été effectuées en mode local, nous avons pu découvrir, en pratique, les défis de la manipulation des données massives, comme la présence des données manquantes, les données incomplètes, la limite des outils traditionnels pour lancer le premier test impliquant des données massives, etc.

La disponibilité des outils informatiques permettant de stocker et de traiter des données à grande échelle, avec efficacité, est importante. Toutefois, le modèle qui dirige l'ensemble de données est aussi de même degré d'importance dans un processus d'analyse de données, comme le cas du modèle créé par Fontugne et al. [17] pour la détection des anomalies. Nous avons compris le fonctionnement ainsi que le paramétrage du modèle. Comme continuité du présent travail, on note quelques possibilités. Par exemple, il est possible de réévaluer la précision de ce modèle avec de nouvelles données, varier les paramètres du modèle de la détection comme la méthode adoptée au calcul des intervalles de confiance, etc. Du fait que RIPE Atlas dispose du mode Streaming dans lequel les données peuvent être récupérées en temps réel, il est intéressant d'évaluer l'intégration de l'extension *Spark streaming* pour analyser les données RIPE Atlas en temps réel.

# Annexe A

## Amazon Athena

### A.1 Création de la table traceroutes

La création d'une table reprend plusieurs parties :

- les colonnes de la table avec le type correspondant (int, string, array pour définir une liste, struct pour définir un objet) ;
- LOCATION : c'est l'endroit où les données sont stockées dans Amazon S3, il faut préciser le chemin vers le compartiment de données ;
- ROW FORMAT SERDE : elle définit la manière dont chaque ligne d'un fichier de données est sérialisée/désérialisée par Amazon Athena ;
- PARTITIONED BY : elle définit la manière dont les données sont organisées dans le compartiment de données ;
- WITH serdeproperties : elle définit les options de la sérialisation/désérialisation.

Listing A.1 – Création de la table des traceroutes dans Amazon Athena

```
CREATE EXTERNAL TABLE traceroutes_api (  
    af int ,  
    bundle int ,  
    dst_addr string ,  
    dst_name string ,  
    fw int ,  
    endtime int ,  
    'from' string ,  
    group_id int ,  
    lts int ,  
    msm_id int ,  
    msm_name string ,  
    paris_id int ,  
    prb_id int ,  
    proto string ,  
    size int ,  
    src_addr string ,  
    'timestamp' int ,  
    ttr float ,  
    type string ,  
    result array< struct< hop :int , error :string , result :array<
```

```

    struct<x:string, err:string, 'from':string, ittl:int, edst:string,
        late:int, mtu:int, rtt:float, size:int, ttl:int, flags:string,
        dstoptsize:int, hbhoptsize:int, icmpext:
            struct<version:int, rfc4884:int, obj:array<
                struct<class:int, type:int, mpls:array<struct<exp:
                    int, label:int, s:int, ttl:int>>>>>>>>>>>>
            >
    )
PARTITIONED BY (
    af_ string,
    type_ string,
    msm string,
    year string,
    month string,
    day string,
    hour string
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH serdeproperties ('paths'='af,bundle,dst_addr,dst_name,fw,endtime,
    from,lts,msm_id,paris_id,prb_id,proto,size,src_addr,timestamp,
    type,fw,msm_name')
LOCATION 's3://ripeatlasdata/traceroute/source=api/'

```

## A.2 Partitionnement de données sur Amazon Athena

### A.2.1 Présentation du partitionnement

Le partitionnement de données présentes dans un compartiment Amazon S3 permet de limiter la quantité de données à analyser par une requête Amazon Athena. Le partitionnement améliore les performances d'Amazon Athena. D'une part, on obtient une réponse rapidement, d'autre part, on réduit les coûts engendrés suite à l'utilisation du service car on est facturé selon la quantité de données analysées.

Les partitions créées jouent un rôle similaire à celui d'une colonne durant l'interrogation d'une table dans Athena.

Prenons un exemple, nous avons des traceroutes ayant comme adressage IP la version 4 et d'autres traceroutes ont l'adressage IPv6 :

```

s3://ripeatlasdata/traceroute/
                                type=4/
                                type=6/

```

Sans l'utilisation du partitionnement et si on souhaite récupérer que les traceroutes ayant comme adressage IPv4, toutes les données (type = 4 et type = 6) sont analysées. Toutefois, en partitionnant les données suivant le type d'adressage, seuls les fichiers dans le dossier type = 4 qui sont analysés. Par conséquent, le partitionnement permet de réduire les coûts d'utilisation du service Amazon Athena, surtout si la quantité de données est très importante.

### A.2.2 Application du partitionnement sur les traceroutes Atlas

Les traceroutes en provenance des sondes Atlas sont organisés comme est illustré dans la Figure :

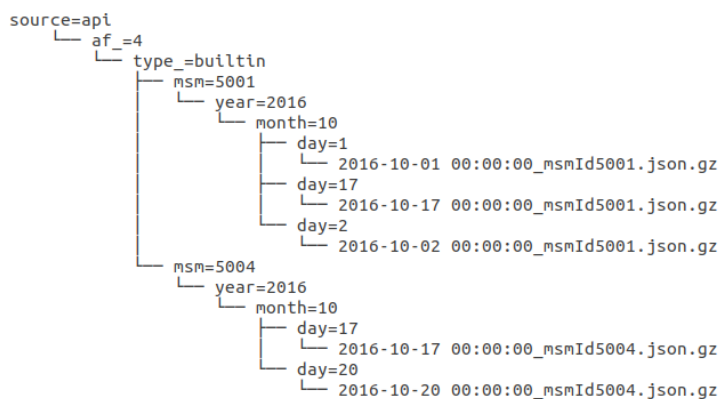


FIGURE A.1 – L’organisation des traceroutes dans un compartiment Amazon S3

### A.2.3 L’interrogation de données sur Amazon Athena

L’interrogation de données présentes sur Amazon S3 via Amazon Athena est effectué avec une requête SQL. On peut décomposer une requête SQL en trois parties principales :

**Les données sollicitées** Ça peut être une colonne ou bien une colonne transformée suite à l’application d’une ou de plusieurs fonctions de Presto.

**Les partitions de données concernées** La requête SQL est paramétrée de sorte à limiter les données à analyser sur Amazon S3.

**Les paramètres appliqués sur les colonnes** La requête SQL doit filtrer les données suivant les conditions sur les colonnes de la table.

En pratique, nous avons créé la requête SQL reprise dans la Figure A.2.

Pour les données sollicitées, ce sont les trois colonnes *prb\_id*, *from*, *msm\_id* (ligne 2) et la liste de saut (*hops*) obtenue après quelques vérifications (lignes 3 à 9). A la ligne 10, c’est la table créée pour les traceroutes (voir les détails de la table dans A.1). Les traceroutes à analyser sont ceux obtenus en vérifiant les conditions dans la ligne 12 et 13. C’est à dire, Athena va regarder les traceroutes qui se trouvent dans les sous dossiers year=2016

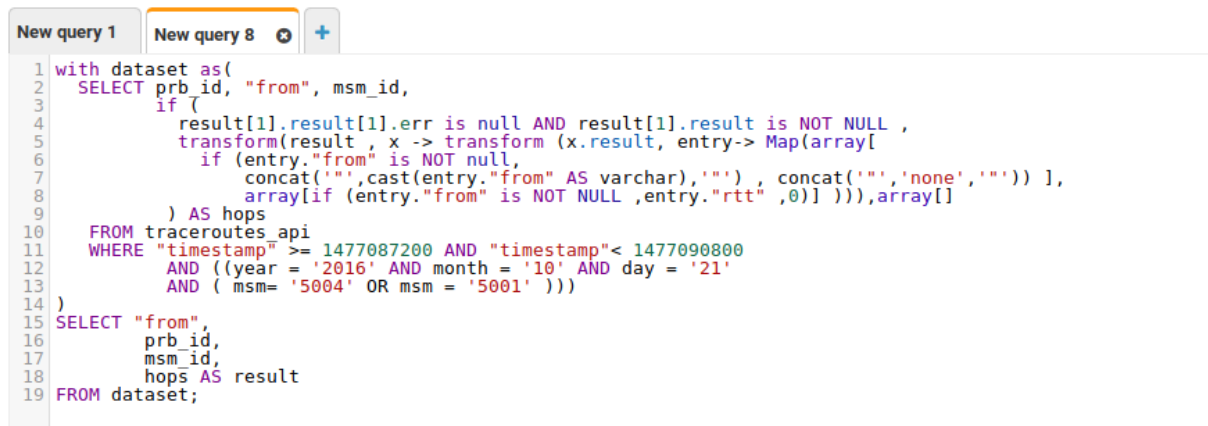
month=10

day=21 qui se trouvent aussi dans les deux dossiers msm=5001 et msm=5004. Ce que revient à chercher les traceroutes dans les deux endroits suivants :

```
s3://repeatlasdata/traceroute/source=api/af_=4/type_=builtin/msm=5001/year=2016/month=10/day=21
```

```
s3://repeatlasdata/traceroute/source=api/af_=4/type_=builtin/msm=5004/year=2016/month=10/day=21
```





```
1 with dataset as(
2   SELECT prb_id, "from", msm_id,
3     if (
4       result[1].result[1].err is null AND result[1].result is NOT NULL ,
5       transform(result , x -> transform (x.result, entry-> Map(array[
6         if (entry."from" is NOT null,
7           concat('"' ,cast(entry."from" AS varchar),'"' , concat('"' , 'none' , '"') ) ],
8         array[if (entry."from" is NOT NULL ,entry."rtt" ,0)] ))) ,array[]
9     ) AS hops
10  FROM traceroutes_api
11  WHERE "timestamp" >= 1477087200 AND "timestamp"< 1477090800
12        AND ((year = '2016' AND month = '10' AND day = '21'
13              AND ( msm= '5004' OR msm = '5001' )))
14 )
15 SELECT "from",
16        prb_id,
17        msm_id,
18        hops AS result
19 FROM dataset;
```

FIGURE A.2 – Une exemple d’une requête SQL sur Amazon Athena

# Annexe B

## Important à savoir

### B.1 Pourquoi à partir des temps RTTs bruts on arrive à détecter les anomalies ?

La détection des anomalies se base sur la caractérisation de la distribution des RTTs différentiels. Tel que dans un premier temps, on construit une référence et ensuite identifier les déviations par rapport à cette référence.

En pratique, la détection des anomalies repose sur une variante du théorème central limite (CLT).

Dans notre cas, la distribution des moyennes d'un nombre assez large d'échantillons des RTTs différentiels sont distribuées normalement. En cas de changement du délai entre les deux routeurs X et Y, la moyenne résultante dévie de la distribution normale ainsi une anomalie est détectée.

Les premières expériences ont montré que les valeurs "bruits" peuvent affecter largement les valeurs des moyennes calculées d'où la nécessité d'avoir un nombre très important d'échantillons, ce qui n'est pas praticable. Afin de résoudre ce besoin, les autres du travail ont utilisé la médiane à la place de la moyenne car :

- elle est robuste aux valeurs bruits des RTTs ;
- la convergence des médianes vers la distribution normale nécessite moins d'échantillons.

### B.2 Rappel

La programmation fonctionnelle est parmi les paradigmes de programmation. Ce paradigme se base sur les fonctions au sens mathématique du terme.

### B.3 Processus de détection des anomalies

#### B.3.1 Les phases de la détection

**Phase I : préparation des données** La Figure D.1 reprend les étapes de la phase I. Chaque étape est définie par ses entrées et ses sorties. L'objectif de chaque étape est détaillé dans ce qui suit :

- **findAllBins (I.1)** : nous cherchons toutes périodes ([bin]) entre start et end dont leur durée est de timeWindow.
- **findTraceroutesByBin (I.2)** : pour toute période bin, nous récupérons les traceroutes capturés durant cette dernière. Le nombre de traceroutes associés à chaque bin dépend des traceroutes disponibles à l'analyse ([traceroute]).
- **aggregateRttsByHop (I.3)** : certains traceroutes ne réussissent pas à atteindre la destination prévue. Dans ce cas, ces traceroutes sont ignorés. Les saut ayant des RTTs invalides sont aussi ignorés.
- **aggregateRttsByHop (I.4)** : étant donné que chaque saut (hop) est décrit par plus d'un RTT, nous calculons la médiane de ces RTTs (rttAgg), et ce par source de signal.
- **linkInference (I.5)** : chaque deux sauts consécutifs d'un traceroute forment un lien. Pour tout traceroute d'un bin donné, nous inférons les liens possibles. Ensuite, nous caractérisons ce lien (ip1, ip2) avec son RTT différentiel (rttDiff).
- **sortLinks (I.6)** : à cette étape, nous ordonnons les couples d'adresses IP (ip1, ip2) suivant l'ordre alphabétique. Par exemple,
- **mergeLinks (I.7)** : nous fusionnons les liens avec leurs caractéristiques. En résultat, les liens (ip1, ip2) et (ip2, ip1) sont fusionnés et sont représentés par le lien (ip1, ip2) si  $(ip1, ip2) < (ip2, ip1)$ .
- **resumeData (I.8)** : jusqu'à cette étape, les liens sont organisés par bin. Vu qu'un lien peut être identifié durant plusieurs périodes, nous résumons un lien en fusionnant leurs RTTs différentiels. Toutefois, chaque RTT différentiel doit garder le bin pendant lequel il a été identifié, ainsi les deux listes [rttDiff] et [bin] ont la même longueur.

**Phase II : détection des anomalies** A la fin de la phase I, nous avons à notre disposition tous les liens identifiés avec leurs caractéristiques. La Figure D.2 illustre les étapes principales de la phase II.

- **findAllBins (II.1)** : nous générons les mêmes périodes (bin) que celles générées dans l'étape (I.1).
- **findRTTDiffByBin (II.2)** : nous reprenons les périodes dans l'ordre chronologique et pour chaque bin nous cherchons les RTTs différentiels correspondants à chaque bin. Nous notons (dist) l'ensemble des RTTs différentiels correspondants à un bin donné. Note : La période ayant enregistré moins de 4 RTTs différentiels est négligée et nous passons à la période suivante.
- **wilsonScoreProcess (II.3)** : le calcul de l'interface de confiance à partir du score de Wilson nécessite d'avoir le nombre total des expériences, le nombre d'expérience réussies et le risque d'erreur. Nous calculons les deux bornes du score de Wilson suivant les étapes suivantes :
  1. Soit  $n$  la taille de la distribution, nous calculons le score de Wilson, ce que donne le couple de valeurs ( $low$ ,  $high$ ).
  2. nous multiplions  $low$  et  $high$  par  $n$ , ainsi nous avons  $(lo, hi) = (n * low, n * high)$ .
  3. nous récupérons la partie entière de  $lo$  et  $hi$ , ainsi nous avons  $(l, h) = (int(lo), int(hi))$ <sup>1</sup>.

---

1. La fonction  $int()$  est utilisée pour avoir la partie entière d'un nombre.

4. nous parcourons la distribution `dist` pour trouver les RTTs différentiels qui se trouvent au rang  $l$  et  $h$  dans `dist`, respectivement sont  $rttDiffL$ ,  $rttDiffH$ . Ainsi :  
 $rttDiffL = dist[l]$  et  $rttDiffH = dist[h]$ , avec  $l, h \in [0, n[$ .
- **updateCurrentLinkState (II.4)** : `current` est l'état courant du lien à une période donnée. Nous calculons l'état du lien en ce bin en calculant :
    - le RTT différentiel médian (`median`) pour le bin courant.  
 $currentMedian = median(dist)$
    - la borne inférieure de l'intervalle de confiance (`ciLow`)  
 $currentCiLow = median - rttDiffL$
    - la borne supérieure de l'intervalle de confiance (`ciHight`).  
 $currentCiHight = rttDiffH - median$
  - **updateReferenceLinkState(II.5)** : `reference` est l'état de référence du lien à une période donnée. Son calcul est réalisé selon trois cas. De même que l'état courant, l'état référence est décrit par la médiane du RTT différentiel, la borne inférieure de l'intervalle de confiance de référence et enfin la borne supérieure de l'intervalle de confiance de référence.
    1. Tant que nous n'avons pas construit une référence assez représentable de l'état du lien (cas 1). nous mettons à jour la référence (`reference`) comme suit :  
 $referenceMedian = median(dist)$   
 $referenceCiLow = rttDiffL$   
 $referenceCiHight = rttDiffH$
    2. Une fois nous atteignons le nombre nécessaire de mise à jour de la référence assurant la représentativité de la référence (cas 2), nous mettons à jour l'état référence en remplaçant la médiane de chaque période par `aggre_median`, la borne inférieure de chaque période par `aggr_ciLow` et la borne supérieure de chaque période par `aggr_ciHigh`.
      - soit `aggre_median` la médiane de tous les RTTs différentiels médians de référence calculés pour toutes les périodes précédentes.
      - soit `aggr_ciLow` la médiane de toutes les bornes inférieures calculées pendant les périodes précédentes.
      - soit `aggr_ciHigh` la médiane de toutes les bornes supérieures calculées pendant les périodes précédentes.
    3. Une fois la référence est représentable (cas 3), nous mettons à jour cette dernière comme suit :  
 $referenceMedian = 0.99 * referenceMedian[-1] + 0.01 * currentMedian$   
 $referenceCiLow = 0.99 * referenceLow[-1] + 0.01 * rttDiffL$   
 $referenceCiHight = 0.99 * referenceHight[-1] + 0.01 * rttDiffH$
  - **alarmsDetection (II.6)** : la détection des anomalies est déclenché à partir du cas 3. Si une anomalie est détectée, nous mettons à jour la liste des alarmes (`alarmsValues`) avec le RTT différentiel médian (`currentMedian`) ainsi que la liste des dates (`alarmsDates`) qui correspondent à ces anomalies. Les anomalies sont détecté en évaluant le chevauchement

entre l'intervalle de confiance courant et celui de référence. La période courante présente une anomalie si la condition suivante est vraie<sup>2</sup> :

Listing B.1 – Illustration

```
1 currentMedian[-1]-currentCiLow[-1] > referenceHi[-1] or  
2 currentMedian[-1]+currentCiHigh[-1] < referenceCiLow[-1]) and  
3 np.abs(currentMedian[-1]-reference[-1])>1
```

### Illustrations des deux phases de détection

---

2. Nous notons par [-1] l'élément précédent d'une mesure. Par exemple, `currentMedian[-1]` est la médiane des RTTs différentiels de la période qui précède la période courante.

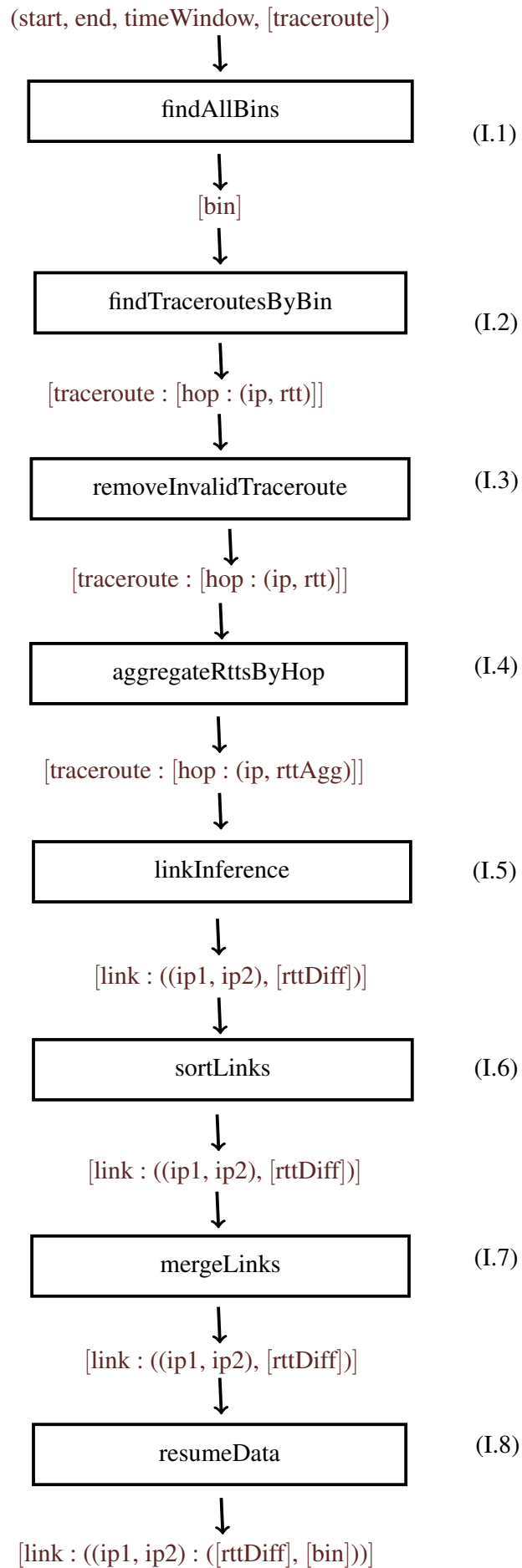


FIGURE B.1 – Phase I : préparation des données

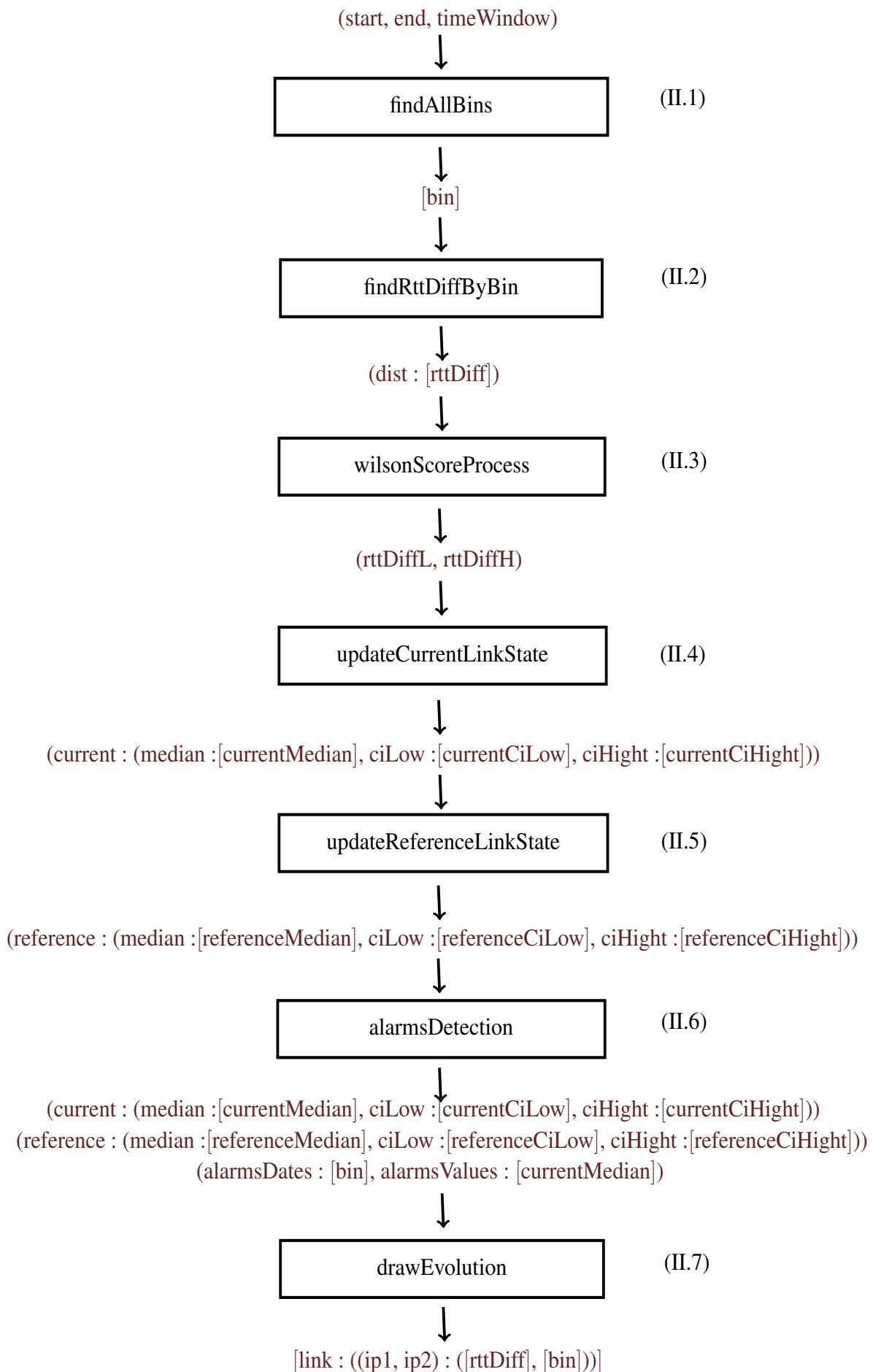


FIGURE B.2 – Phase II : détection des changements

### B.3.2 Exemple illustratif

#### Description de l'échantillon :

L'échantillon des traceroutes qui va nous permettre d'illustrer l'algorithme de la détection est défini comme suit :

- le nombre total des traceroutes est de 24 ;
- le premier traceroute a été capturé à 1514769800 (GMT : Monday, January 1, 2018 1 :23 :20) ;
- le dernier traceroute capturé à 1514787809 (GMT : Monday, January 1, 2018 6 :23 :29 ) ;
- la durée d'une période est 1 heure (3600) ;
- les débuts des périodes entre 1514769800 et 1514789809 sont 1514769800, 1514773400, 1514777000, 1514780600, 1514784200, 1514787800 ;
- la liste des traceroutes analysés est disponible sur[1].

étape	outputs		
(I.1)	1514769800, 1514773400, 1514777000, 1514780600, 1514784200, 1514787800		
(I.2)	<b>Id</b>	<b>Période</b>	<b>nombre de traceroutes</b>
	1	[1514769800, 1514769800+3600]	4
	2	[1514773400, 1514773400+3600]	4
	3	[1514777000, 1514777000+3600]	4
	4	[1514780600, 1514780600+3600]	4
	5	[1514784200, 1514784200+3600]	4
	6	[1514787800, 1514787800+3600]	4

Une fois les traceroutes sont groupés par période, nous parcourons chaque traceroute afin d'éliminer ceux invalides (`removeInvalidTraceroute` I.3).

Ensuite, nous agrégeons chaque saut par source de signal en calculant leur médiane. Nous donnons un exemple d'un traceroute qui illustre cette agrégation. Le même traitement est appliqué sur tout traceroute.

Listing B.2 – Les saut du traceroute T7 (sans agrégation)

```

1 hop_id : 1, hops : [(from : 89.105.200.57, rtt : 1.955, x :
  None), (from : 89.105.200.57, rtt : 1.7, x : None), (from : 89
  .105.200.57, rtt : 1.709, x : None)]
2 hop_id : 2, hops : [(from : 185.147.12.31, rtt : 8.543, x :
  None), (from : 185.147.12.31, rtt : 4.103, x : None), (from :
  185.147.12.31, rtt : 4.41, x : None)]
3 hop_id : 3, hops : [(from : 185.147.12.19, rtt : 4.347, x :
  None), (from : 185.147.12.19, rtt : 2.876, x : None), (from :
  185.147.12.19, rtt : 3.143, x : None )]
```

Listing B.3 – Les saut du traceroute T7 (après l'agrégation)

```

1 hop_id : 1, hops : [(from : 89.105.200.57, rttAgg : 1.709)]
2 hop_id : 2, hops : [(from : 185.147.12.31, rttAgg : 4.41)]
3 hop_id : 3, hops : [(from : 185.147.12.19, rttAgg : 3.143)]
```



L'inférence (I.5) des liens du traceroute T7 donne les liens suivants :

Listing B.4 – Exemple des liens inférés du traceroute T7

```
1 lien 1 : (link : (185.147.12.31, 89.105.200.57), rttDiff : 2.701)
2 lien 2 : (link : (185.147.12.19, 185.147.12.31), rttDiff : -1.267)
```

A l'étape (I.6), nous ordonnons chaque lien, nous ordonnons plutôt leurs adresses IP comme illustré ci-dessous avec la fonction *sort()*.

Listing B.5 – Illustration de l'ordre des liens

```
1 sort(("185.147.12.31", "89.105.200.57")) = ("185.147.12.31", "89.105.200.57")
2 sort(("89.105.200.57", "185.147.12.31")) = ("185.147.12.31", "89.105.200.57")
```

L'ordonnancement à l'étape (I.6) est une préparation à la fusion. Nous fusionnons les liens ayant impliqué les mêmes routeurs dans une même période. Prenons les liens inférés à partir de 5 traceroutes comme c'est montré ci-dessous :

Listing B.6 – Liste des liens possibles inférés via les traceroutes T1, T2, T3, T4 et T5

```
1 T1 (link : (185.147.12.31, 89.105.200.57), rttDiff : 2.463)
2   (link : (185.147.12.19, 185.147.12.31), rttDiff : -1.029)
3
4 T2 (link : (185.147.12.31, 89.105.200.57), rttDiff : 3.463)
5   (link : (185.147.12.19, 185.147.12.31), rttDiff : -2.029)
6
7 T3 (link : (185.147.12.31, 89.105.200.57), rttDiff : 2.991)
8   (link : (185.147.12.19, 185.147.12.31), rttDiff : -1.557)
9
10 T4 (link : (185.147.12.31, 89.105.200.57), rttDiff : 4.631)
11   (link : (185.147.12.19, 185.147.12.31), rttDiff : -3.197)
12
13 T5 (link : (89.105.200.57, 185.147.12.31), rttDiff : -7.75)
14   (link : (185.147.12.19, 89.105.200.57), rttDiff : 3.803)
```

La fusion de ces liens nous donne les trois liens ci-dessous. Chaque lien est caractérisé par les sondes l'ayant capturé, leurs RTTs Différentiels et la période dupliquée, pendant laquelle il a été identifié, en nombre de RTT différentiel.

Listing B.7 – Caractérisation des liens identifiés lors de la période 1514769800 avec les traceroutes T1, T2, T3, T4 et T5

```
1 [(link : (185.147.12.19, 89.105.200.57),
2  probes : [4247],
3  rttDiffs : [3.803],
4  bins : [1514769800]),
5
6  (link : (185.147.12.19, 185.147.12.31),
7  probes : [4247, 4247, 4247, 4247],
8  rttDiffs : [-1.029, -2.029, -1.557, -3.197],
9  bins : [1514769800, 1514769800, 1514769800, 1514769800]),
10
11 (link : (185.147.12.31, 89.105.200.57),
12 probes : [4247, 4247, 4247, 4247, 4247],
13 rttDiffs : [2.463, 3.463, 2.991, 4.631, -7.75],
14 bins : [1514769800, 1514769800, 1514769800, 1514769800, 1514769800]) ]
```

Après avoir analysé toutes les périodes, nous résumons les liens en fusionnant les données relatives à toutes les périodes, c'est l'objectif de l'étape (I.8).



itération	1
bin	1514769800
lien	(185.147.12.31, 89.105.200.57)
dist	[-7.75, 2.463, 2.991, 3.463, 4.631]
(low, hight)	(0.11762077423264793, 0.7692757187239873)
(lo, hi)	(0.5881038711632396, 3.8463785936199364)
(l, h)	(0, 3)
rttDiffL	dist[0] = -7.75
rttDiffH	dist[3]= 3.463
currentMedian	2.991
currentCiLow	2.991 - (-7.75) = 10.741
currentCiHi	3.463 - 2.991 = 0.472
cas 1 ? 2 ? 3 ?	cas 1 (aucun intervalle de confiance calculé)
referenceMedian	2.991
referenceCiLow	-7.75
referenceCiHi	3.463
détection des alarmes ?	non (car cas 3 pas encore atteint)
alarmesDates	[]
alarmesValues	[]

itération	2
bin	1514773400
lien	(185.147.12.31, 89.105.200.57)
dist	[2.691, 2.711, 3.451, 4.402]
(low, hight)	(0.15003898915214947, 0.8499610108478506)
(lo, hi)	(0.6001559566085979, 3.3998440433914023)
(l, h)	(0, 3)
rttDiffL	dist[0] = 2.691
rttDiffH	dist[3]= 4.402
currentMedian	3.081
currentCiLow	0.39
currentCiHi	1.321
cas 1 ? 2 ? 3 ?	cas 1 (1 intervalle de confiance calculé)
referenceMedian	3.081
referenceCiLow	2.691
referenceCiHi	4.402
détection des alarmes ?	non (car cas 3 pas encore atteint)
alarmesDates	[]
alarmesValues	[]

itération	3
bin	1514777000
lien	(185.147.12.31, 89.105.200.57)
dist	[2.394, 2.421, 3.401, 4.635]
(low, hight)	(0.15003898915214947, 0.8499610108478506)
(lo, hi)	(0.6001559566085979, 3.3998440433914023)
(l, h)	(0, 3)
rttDiffL	dist[0] = 2.394
rttDiffH	dist[3]= 4.635
currentMedian	2.9109999999999996
currentCiLow	0.5169999999999996
currentCiHi	1.7240000000000004
cas 1 ? 2 ? 3 ?	cas 1 (2 intervalle de confiance calculé)
referenceMedian	2.9109999999999996
referenceCiLow	2.394
referenceCiHi	4.635
détection des alarmes ?	(car cas 3 pas encore atteint)
alarmesDates	[]
alarmesValues	[]

itération	4
bin	1514780600
lien	(185.147.12.31, 89.105.200.57)
dist	[681.463, 801.463, 846.463, 999.463]
(low, hight)	(0.15003898915214947, 0.8499610108478506)
(lo, hi)	(0.6001559566085979, 3.3998440433914023)
(l, h)	(0, 3)
rttDiffL	dist[0] = 681.463
rttDiffH	dist[3]= 999.463
currentMedian	823.963
currentCiLow	142.5
currentCiHi	175.5
cas 1 ? 2 ? 3 ?	cas 2 (3 intervalle de confiance calculé)
referenceMedian	2.991 (car median([2.991, 3.081, 2.9109999999999996])= 2.991)
referenceCiLow	2.394 (car median ([-7.75, 2.691, 2.394])= 2.394)
referenceCiHi	4.402 (car median([3.463, 4.402, 4.635])=4.402)
détection des alarmes ?	non (car cas 3 pas encore atteint)
alarmesDates	[]
alarmesValues	[]

itération	5
bin	1514784200
lien	(185.147.12.31, 89.105.200.57)
dist	[1046.463, 1061.463, 1081.463, 1099.463]
(low, high)	(0.15003898915214947, 0.8499610108478506)
(lo, hi)	(0.6001559566085979, 3.3998440433914023)
(l, h)	(0, 3)
rttDiffL	dist[0] = 1046.463
rttDiffH	dist[3]= 1099.463
currentMedian	1071.463
currentCiLow	25.0
currentCiHi	28.0
cas 1 ? 2 ? 3 ?	cas 3 (car la référence est assez représentable )
referenceMedian	13.67572 (car $0.992.991 + 0.011071.463 = 13.67572$ )
referenceCiLow	12.83469 (car $0.992.394 + 0.011046.463 = 12.83469$ )
referenceCiHi	15.352609999999999 (car $0.994.402 + 0.011099.463 = 15.35261$ )
détection des alarmes ?	oui (car : cas 3)
alarmesDates	[1514784200]
alarmesValues	[1071.463]

itération	6
bin	1514787800
lien	(185.147.12.31, 89.105.200.57)
dist	[2.701, 2.924, 3.841, 4.463]
(low, high)	(0.15003898915214947, 0.8499610108478506)
(lo, hi)	(0.6001559566085979, 3.3998440433914023)
(l, h)	(0, 3)
rttDiffL	dist[0] = 2.701
rttDiffH	dist[3]= 4.463
currentMedian	3.3825000000000003
currentCiLow	0.6815000000000003
currentCiHi	15.243713899999999
cas 1 ? 2 ? 3 ?	cas 3 (car la référence est assez représentable )
referenceMedian	13.5727878
referenceCiLow	12.7333531
referenceCiHi	15.352609999999999
détection des alarmes ?	oui (car : cas 3)
alarmesDates	[1514784200, 1514787800]
alarmesValues	[1071.463, 3.3825000000000003]

## B.4 Implémentation Spark/Scala

Nous avons reproduit le processus de la détection des anomalies avec l’outil Spark avec Scala comme API. Le processus de détection est décrite dans la section ??.

Nous avons implémenté l’outil de détection en utilisant le framework Spark présenté dans la section III.4.5. Afin d’accéder aux différentes fonctionnalités, nous avons utilisé l’API Scala.

En pratique, nous avons utilisé le mode standalone du Spark (voir les modes du Spark dans ??). L'implémentation Spark/Scala reprend des éléments suivants :

### **Le programme driver**

**La création des case class** Les case class sont utilisées pour modéliser les données immuables. Nous avons utilisé plusieurs classes de ce type afin de modéliser les données tout au long de l'analyse.

### **La création RDDs**

### **La création des transformations**

# Annexe C

## Exemple d'une réponse traceroute

Nous présentons ci-après un exemple d'une réponse d'une requête traceroute à destination de l'adresse IP 192.5.5.241.

Listing C.1 – Exemple d'une réponse traceroute

```
1 {
2     "lts":38,
3     "size":40,
4     "from":"80.64.105.210",
5     "dst_name":"192.5.5.241",
6     "fw":4900,
7     "proto":"UDP",
8     "af":4,
9     "msm_name":"Traceroute",
10    "stored_timestamp":1517965930,
11    "prb_id":30070,
12    "result":[
13    {
14        "result":[
15        {
16            "rtt":0.688,
17            "ttl":64,
18            "from":"172.19.19.5",
19            "size":68
20        },
21        {
22            "rtt":0.452,
23            "ttl":64,
24            "from":"172.19.19.5",
25            "size":68
26        },
27        {
28            "rtt":0.447,
29            "ttl":64,
30            "from":"172.19.19.5",
31            "size":68
32        }
33    ]
34    }
```

```
33         ],
34         "hop":1
35     },
36     {
37         "result":[
38             {
39                 "rtt":1.601,
40                 "ttl":254,
41                 "from":"80.64.105.193",
42                 "size":68
43             },
44             {
45                 "rtt":1.214,
46                 "ttl":254,
47                 "from":"80.64.105.193",
48                 "size":68
49             },
50             {
51                 "rtt":0.987,
52                 "ttl":254,
53                 "from":"80.64.105.193",
54                 "size":68
55             }
56         ],
57         "hop":2
58     }
59 ],
60 "timestamp":1517965878,
61 "src_addr":"172.19.19.200",
62 "paris_id":8,
63 "endtime":1517965878,
64 "type":"traceroute",
65 "dst_addr":"192.5.5.241",
66 "msm_id":5004
67 }
```



# Table des figures

I.1	Les trois générations des sondes . . . . .	8
I.2	La connexion des sondes à l'infrastructure RIPE Atlas [24] . . . . .	9
I.3	L'architecture du système Atlas . . . . .	10
I.4	Les étapes d'établissement d'une connexion entre une sonde et l'architecture Atlas . . . . .	12
I.5	La corrélation entre la moyenne des AS paths et la moyenne des RTTs [18] . . . . .	24
I.6	Visualisation des changements des chemins traceroute [15] . . . . .	25
II.1	(a) Le RTT entre la sonde P et les routeurs B et C. (b) La différence entre les chemins de retour depuis les routeurs B et C vers la sonde P. Source : [17] . . . . .	29
II.2	Illustration des périodes de l'analyse entre la date de début et la date de fin. . . . .	32
II.3	Phase I : préparation de données . . . . .	36
II.4	Phase II : détection des alarmes . . . . .	37
III.1	Architecture standard du Big Data [30] . . . . .	47
III.2	Illustration d'une base de données NoSQL de type clé-valeur . . . . .	49
III.3	Illustration d'une base de données NoSQL de type document . . . . .	49
III.4	Illustration d'une base de données NoSQL de type colonne . . . . .	50
III.5	Illustration d'une base de données NoSQL de type graphe . . . . .	51
III.6	Bases de données NoSQL suivant le théorème de CAP . . . . .	52
III.7	Un classement des SGBDs sur <i>DB-Engines Ranking</i> du 1 août 2018 . . . . .	53
III.8	Interaction entre une application Spark et le cluster manager. Source : [28] . . . . .	59
III.9	Spark Unified Stack. Source : [29] . . . . .	60
III.10	Exemple d'un flux de données avec Spark . . . . .	61
IV.1	Groupement des traceroutes avec MongoDB . . . . .	67
V.1	. . . . .	84
V.2	Une combinaison des services web d'Amazon : Amazon S3, Amazon Glue, Amazon Athena, Amazon Quicksight et Amazon Redshift . . . . .	86
V.3	L'organisation des traceroutes dans le compartiment Amazon S3 <i>s3://ripeatlas-data</i> . . . . .	87
V.4	Les temps d'exécution de la détection des anomalies en fonction de la taille de données (Amazon S3 et Amazon Athena) . . . . .	90
V.5	Mesure des temps d'exécution de l'application Spark selon différentes tailles de mémoire allouées au <i>driver</i> et pour deux ensembles de données différentes . . . . .	92
V.6	. . . . .	94
V.7	Regroupement des traceroutes d'une journée par leurs nombre de sauts . . . . .	95
V.8	Les liens et les anomalies identifiés par ensemble de traceroutes . . . . .	96
V.9	Comparaison de l'évolution des temps d'exécution entre MongoDB et Amazon Athena avec Amazon S3 . . . . .	96
V.10	La moyenne des temps d'exécution pour MongoDB et Spark . . . . .	97

V.11	Comparaison des temps d'exécution entre MongoDB, AWS et Apache Spark . .	97
A.1	L'organisation des traceroutes dans un compartiment Amazon S3 . . . . .	102
A.2	Une exemple d'une requête SQL sur Amazon Athena . . . . .	103
B.1	Phase I : préparation des données . . . . .	108
B.2	Phase II : détection des changements . . . . .	109

# Liste des tableaux

I.1	Les caractéristiques du matériel des trois générations des sondes . . . . .	8
I.2	Comparaison entre les sondes et les ancres . . . . .	16
I.3	Comparaison entre sondes Atlas et ProbeAPI . . . . .	21
I.4	Les détails des mesures effectuées dans le travail de C. Anderson [13] . . . . .	23
II.1	Récapitulatif des traceroutes utilisés dans le travail de référence . . . . .	29
III.1	Les tarifs du AWS S3 (formule Stockage standard S3) . . . . .	56
V.1	Les temps d'exécution d'analyse de traceroutes en fonction de la taille de données avec MongoDB . . . . .	84
V.2	Exemple des partitions créées dans un compartiment Amazon S3 . . . . .	87
V.3	Les temps d'exécution par taille de l'ensemble de données (Amazon Athena et Amazon S3) . . . . .	90
V.4	Les temps d'exécution des analyses lancées sur des traceroutes (approche 1) . .	93
V.5	Les temps d'exécution lors de la détection des anomalies en utilisant Spark (approche 2) . . . . .	94

# Listings

II.1	Conditions de présence d'une anomalie pour un lien . . . . .	35
II.2	Les sauts du traceroute T7 (sans agrégation) . . . . .	38
II.3	Les sauts du traceroute T7 (après l'agrégation) . . . . .	38
II.4	Exemple des liens inférés du traceroute T7 . . . . .	39
II.5	Illustration de l'ordre des liens . . . . .	39
II.6	Liste des liens possibles inférés via les traceroutes T1, T2, T3, T4 et T5 . . . . .	39
II.7	Caractérisation des liens identifiés lors de la période 1514769800 avec les tra- ceroutes T1, T2, T3, T4 et T5 . . . . .	39
II.8	Illustration de l'ordre des liens . . . . .	39
IV.1	Exemple de configuration avec SparkConf . . . . .	65
IV.2	Définition de la classe Traceroute . . . . .	65
IV.3	Définition de la classe Hop . . . . .	66
IV.4	Définition de la classe Signal . . . . .	66
IV.5	La lecture des données traceroutes . . . . .	66
IV.6	Etape FindBins (I.1) . . . . .	67
IV.7	Définition de la classe TracerouteWithPeriod . . . . .	69
IV.8	Elimination des traceroutes non concernés par l'analyse . . . . .	69
IV.9	Le groupement des traceroutes . . . . .	69
IV.10	Définition de la classe TraceroutesPerPeriod . . . . .	69
IV.11	La classe TracerouteWithPeriod . . . . .	69
IV.12	Deduction des liens par groupe de traceroutes . . . . .	70
IV.13	Définition de la méthode removeInvalidSignals . . . . .	71
IV.14	Définition de la méthode checkSignal . . . . .	71
IV.15	Définition de la fonction computeMedianRTTByhop . . . . .	71
IV.16	Définition de la classe MedianByHopTraceroute . . . . .	72
IV.17	Définition de la classe PreparedHop . . . . .	72
IV.18	Définition de la classe PreparedSignal . . . . .	72
IV.19	Définition de la fonction findLinksAndRttDiffByTraceroute . . . . .	72
IV.20	Définition de la fonction findAllLinks . . . . .	73
IV.21	Définition de la fonction . . . . .	73
IV.22	Définition de la classe Link . . . . .	73
IV.23	Définition de la fonction summarizeLinksTraceroute . . . . .	74
IV.24	Définition de la classe DetailedLink . . . . .	74
IV.25	Définition de la fonction sortLinks . . . . .	74
IV.26	Définition de la classe ResumedLink . . . . .	74
IV.27	Collecte des résultats intermédiaires . . . . .	75
IV.28	Fusion des liens de toute la période de l'analyse . . . . .	75
IV.29	Détection des alarmes des liens . . . . .	75
IV.30	Définition de la méthode listAlarms . . . . .	76
IV.31	Définition de la méthode findAlarms . . . . .	77

IV.32 Exemple de la soumissions d'un traitement sur Spark . . . . .	79
IV.33 Exemple des résultats de l'analyse d'un lien . . . . .	80
A.1 Création de la table des traceroutes dans Amazon Athena . . . . .	100
B.1 Illustration . . . . .	107
B.2 Les saut du traceroute T7 (sans agrégation) . . . . .	110
B.3 Les saut du traceroute T7 (après l'agrégation) . . . . .	110
B.4 Exemple des liens inférés du traceroute T7 . . . . .	111
B.5 Illustration de l'ordre des liens . . . . .	111
B.6 Liste des liens possibles inférés via les traceroutes T1, T2, T3, T4 et T5 . . . . .	111
B.7 Caractérisation des liens identifiés lors de la période 1514769800 avec les tra- ceroutes T1, T2, T3, T4 et T5 . . . . .	111
B.8 Illustration de l'ordre des liens . . . . .	112
C.1 Exemple d'une réponse traceroute . . . . .	117

# Bibliographie

- [1] Archipelago (Ark) Measurement Infrastructure. URL : <http://www.caida.org/projects/ark/>. (consulté le 19/01/2018).
- [2] Center for Applied Internet Data Analysis (CAIDA). URL : <http://www.caida.org/>. (consulté le 06/04/2018).
- [3] Create a New Measurement - RIPE Atlas. URL : <https://atlas.ripe.net/measurements/form/>. consulté le 05/08/2018.
- [4] Le dépôt des données RIPE Atlas. URL : <https://data-store.ripe.net/datasets/atlas-daily-dumps/>. (consulté le 26/07/2018).
- [5] Les archives des détails des sondes atlas. URL : <https://ftp.ripe.net/ripe/atlas/probes/archive/>. (consulté le 28/01/2018).
- [6] RIPE Atlas - Known Bugs and Limitations. URL : <https://atlas.ripe.net/docs/bugs/>. (consulté le 05/04/2018).
- [7] Samknows. URL : <https://www.samknows.com/global-platform>. (consulté le 23/01/2018).
- [8] Xport pro. URL : <https://www.lantronix.com/products/xport-pro/>. (consulté le 08/08/2018/).
- [9] Lissages exponentiels. URL : [https://www.math.u-psud.fr/~goude/Materials/time\\_series/cours3\\_lissage\\_expo.pdf](https://www.math.u-psud.fr/~goude/Materials/time_series/cours3_lissage_expo.pdf), 2017. (consulté le 30/09/2018).
- [10] ABEN, E. How RIPE Atlas Helped Wikipedia Users. URL : <https://labs.ripe.net/Members/emileaben/how-ripe-atlas-helped-wikipedia-users>, 2014. (consulté le 18/08/2018).
- [11] ABEN, E. Looking at France-IX with RIPE Atlas and RIS. URL : <https://labs.ripe.net/Members/emileaben/looking-at-france-ix-with-ripe-atlas-and-ris>, 2015. (consulté le 08/08/2018).
- [12] ABEN, E. Measuring Countries and IXPs with RIPE Atlas. URL : <https://labs.ripe.net/Members/emileaben/measuring-ixps-with-ripe-atlas>, 2015. (consulté le 08/08/2018).
- [13] ANDERSON, C., WINTER, P., AND ROYA. Global network interference detection over the RIPE atlas network. In *4th USENIX Workshop on Free and Open Communications on the Internet (FOCI 14)* (San Diego, CA, 2014), USENIX Association.

- [14] CARLO STROZZI. Nosql : A relational database management system. URL : [http://www.strozzi.it/cgi-bin/CSA/tw7/I/en\\_US/nosql/Home%20Page](http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page). (consulté le 04/05/2019).
- [15] DONATO, V. D. Traceroute Consistency Check. URL : <https://github.com/vdidonato/Traceroute-consistency-check>, 2015. (consulté le 08/08/2018).
- [16] FONTUGNE, R. InternetHealthReport - tartiflette. URL : <https://github.com/InternetHealthReport/tartiflette/tree/master/dataManipulation>.
- [17] FONTUGNE, R., ABEN, E., PELSSER, C., AND BUSH, R. Pinpointing delay and forwarding anomalies using large-scale traceroute measurements. *CoRR abs/1605.04784* (2016).
- [18] GASMI, S. Visualising RIPE Atlas Anchor Measurements. URL : [https://labs.ripe.net/Members/salim\\_gasmi/visualising-ripe-atlas-anchor-measurements](https://labs.ripe.net/Members/salim_gasmi/visualising-ripe-atlas-anchor-measurements), 2015. (consulté le 08/08/2018).
- [19] GUILLAUME VALADON, FRANCOIS CONTAT, M. H., AND HOLTERBACH, T. BGP Atlas Monito (BAM). URL : <https://github.com/guedou/bam>, 2015. (consulté le 08/08/2018).
- [20] HEROLD, J. Bgp + traceroute presentation. URL : <https://labs.ripe.net/Members/becha/ripe-atlas-hackathon-presentations/bgp-traceroute>, 2015. (consulté le 08/08/2018).
- [21] HEROLD, J. BGP + Traceroute using RIPE NCC Atlas. URL : <https://github.com/wires/bgp-traceroutes>, 2015. consulté le 08/08/2018.
- [22] HOLTERBACH, T., PELSSER, C., BUSH, R., AND VANBEVER, L. Quantifying interference between measurements on the ripe atlas platform. In *Proceedings of the 2015 Internet Measurement Conference* (New York, NY, USA, 2015), IMC '15, ACM, pp. 437–443.
- [23] KISTELEKI, R. The AMS-IX Outage as Seen with RIPE Atlas. URL : <https://labs.ripe.net/Members/kistel/the-ams-ix-outage-as-seen-with-ripe-atlas>, 2015. (consulté le 23/01/2018).
- [24] KISTELEKI, R. RIPE Atlas Architecture - how we manage our probes. URL : <https://labs.ripe.net/Members/kistel/ripe-atlas-architecture-how-we-manage-our-probes>, 2017. consulté le (08/08/2018).
- [25] KISTELEKI, R. RIPE Atlas probes as IoT devices. URL : <https://labs.ripe.net/Members/kistel/ripe-atlas-probes-as-iot-devices>, 2017. (consulté le 21/12/2017).
- [26] LE BOUDEC, J.-Y. *Performance Evaluation of Computer and Communication Systems*. EPFL Press, Lausanne, Switzerland, 2010.
- [27] LEFEBVRE, M. *Cours et exercices de probabilités appliquées : incluant les notions de base de statistique*. Presses internationales Polytechnique, 2003.
- [28] LUU, H. *Beginning Apache Spark 2 - 2018*, 1 ed. Apress, 2018, p. 4.

- [29] LUU, H. *Beginning Apache Spark 2 - 2018*, 1 ed. Apress, 2018, p. 7.
- [30] MAHESHWARI, A. *Big Data*. 2017.
- [31] RIPE NCC. Test Traffic Measurement Service (TTM). URL : <https://www.ripe.net/analyse/archived-projects/ttm>. (consulté le 14/01/2018).
- [32] RODERICK, F. On the Diversity of Interdomain Routing in Africa. URL : [https://labs.ripe.net/Members/fanou\\_roderick/on-the-diversity-of-interdomain-routing-in-africa](https://labs.ripe.net/Members/fanou_roderick/on-the-diversity-of-interdomain-routing-in-africa), 2015. (consulté le 11/01/2018).
- [33] SAGIROGLU, S., AND SINANC, D. Big data : A review. In *2013 International Conference on Collaboration Technologies and Systems (CTS)* (May 2013), pp. 42–47.
- [34] SHAH, A., FONTUGNE, R., AND PAPADOPOULOS, C. Towards characterizing international routing detours. In *Proceedings of the 12th Asian Internet Engineering Conference (AINTEC)* (Bangkok, Thailand, Nov. 2016), ACM, p. to appear.
- [35] SHAO, W., ROUGIER, J., DEVIENNE, F., AND VISTE, M. Missing measurements on RIPE atlas. *CoRR abs/1701.00938* (2017).
- [36] SHAVITT, Y., AND SHIR, E. Dimes : Let the internet measure itself. *SIGCOMM Comput. Commun. Rev.* 35, 5 (Oct. 2005), 71–74.
- [37] SPEED CHECKER. Global Internet testing - ProbeAPI. URL : <http://probeapi.speedchecker.xyz/>. (consulté le 06/08/2018).
- [38] VARAS, C. A Practical Comparison Between RIPE Atlas and ProbeAPI. URL : [https://labs.ripe.net/Members/cristian\\_varas/a-practical-comparison-between-ripe-atlas-and-probeapi](https://labs.ripe.net/Members/cristian_varas/a-practical-comparison-between-ripe-atlas-and-probeapi), 2016. (consulté le 19/01/2018).