

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

I	RIPE Atlas	6
I.1	Introduction	6
I.2	A propos RIPE NCC	6
I.3	Présentation du projet RIPE Atlas	7
I.3.1	Les mesures actives et passives de l'Internet	7
I.3.2	Généralités sur les sondes Atlas	7
I.3.3	Les générations des sondes Atlas	8
I.3.4	Les versions du firmware des sondes Atlas	9
I.3.5	La connexion des sondes Atlas à Internet	9
I.3.6	Architecture du système RIPE Atlas	10
I.3.7	Les sondes Atlas et la vie privée	14
I.3.8	La sécurité dans RIPE Atlas	14
I.3.9	Les ancres VS sondes Atlas	14
I.3.10	Les mesures intégrées : Built-in	15
I.3.11	Le système de crédits Atlas	16
I.3.12	Les mesures personnalisées : User Defined measurement	17
I.3.13	La sélection des sondes Atlas	18
I.3.14	Les sources de données Atlas	18
I.3.15	Les limitations du RIPE Atlas	19
I.3.16	Confiance aux données Atlas	20
I.4	Projets existants de mesures d'Internet	20
I.4.1	Test Traffic Measurement Service	20
I.4.2	ProbeAPI	21
I.4.3	Archipelago	22
I.4.4	DIMES	22
I.4.5	SamKnows	22
I.5	Quelques cas d'utilisation des données collectées par les sondes Atlas	23
I.5.1	Détection des coupures d'Internet	23
I.5.2	Aide à la prise de décision	23
I.5.3	Le suivi des censures	23
I.5.4	Le suivi des performances d'un réseau	24
I.5.5	Le suivi des détours dans un trafic local	27
I.5.6	Visualisation : indicateurs et dashboard	27
I.6	Conclusion	27
II	La détection des anomalies dans les délais d'un lien	28
II.1	Présentation générale	28
II.2	Pourquoi analyser les délais des liens réseaux	29
II.3	L'étude des délais des liens	30

II.3.1	Les données utilisées dans l'analyse des délais	30
II.3.2	Le principe de la détection des changements des délais	31
II.4	L'étude des délais des liens en pratique : l'évolution du RTT différentiel des liens	32
II.4.1	Les étapes principales de détection	33
II.4.2	Description des paramètres de l'analyse des délais	33
II.4.3	L'évolution du RTT différentiel d'un lien et la détection des anomalies .	34
II.4.4	Processus de détection des anomalies	35
II.4.5	Vue globale des étapes de détection	38
II.5	Conclusion	39
III	Important à savoir	41
III.1	Pourquoi à partir des temps RTTs bruts on arrive à détecter les anomalies ? . . .	41
III.2	Rappel	41
III.3	Processus de détection des anomalies	41
III.3.1	Les phases de la détection	41
III.3.2	Exemple illustratif	47
III.4	Implémentation Spark/Scala	52
IV	Introduction au Big Data	54
IV.1	Introduction	54
IV.2	Processus d'analyse de données massives	54
IV.3	Quelques concepts associés au Big Data	55
IV.3.1	Définition du Big Data : Volume, Vitesse, Variété et Véracité	55
IV.3.2	L'architecture standard du Big Data	56
IV.3.3	Les bases de données NoSQL (Not Only SQL)	58
IV.3.4	Schema on Write VS Schema on Read	62
IV.3.5	L'informatique distribuée et l'analyse de données massives	63
IV.4	Parcours de quelques technologies du Big Data	64
IV.4.1	MongoDB	64
IV.4.2	Amazon DynamoDB	64
IV.4.3	Amazon S3, Amazon Glue et Amazon Athena	65
IV.4.4	Apache Spark	67
IV.5	Conclusion	70
V	Implémentation de la détection des anomalies dans les délais en Spark/Scala	71
V.1	Introduction	71
V.2	Implémentation	71
V.2.1	Description de l'environnement	71
V.2.2	Éléments de l'implémentation	71
V.2.3	Exécution d'une application Spark	82
V.3	Conclusion	82
VI	Application de quelques technologies Big Data sur l'analyse des traceroutes	83
VI.1	Introduction	83
VI.2	Critères d'évaluation des technologies Big Data	83
VI.3	Caractéristiques de l'environnement de test	84
VI.4	Application 1 : MongoDB	84
VI.5	Application 2 : Amazon DynamoDB	86
VI.6	Application 3 : Amazon S3, Amazon Glue et Amazon Athena	86

VI.7 Application 4 : Spark Apache avec Scala	89
VI.8 Récapitulatif	93
VI.9 Conclusion	96
A Amazon Athena	98
A.1 Création de la table traceroutes	98
A.2 Partitionnement de données sur Amazon Athena	99
A.2.1 Présentation du partitionnement	99
A.2.2 Application du partitionnement sur les traceroutes Atlas	99
A.2.3 L'interrogation de données sur Amazon Athena	100
B Exemple d'une réponse traceroute	102

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 de présent projet.

I.2 A propos 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 assure différents services relatifs à la gestion des réseaux informatiques. Il maintient multiples projets pour un nombre de protocoles comme DNS (Domain Name System) (DNSMON¹), BGP (Routing Information Service ou RIS²) 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 de collecter des données relatives aux réseaux informatiques.

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.

I.3 Présentation du projet RIPE Atlas

RIPE NCC a créé le projet RIPE Atlas en 2010. Le nombre de sondes déployées est en augmentation constante, sachant qu'elles sont déployées par des volontaires.

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

Il existe plusieurs approches pour analyser l'état d'un réseau informatique. 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 surveiller le flux de ce trafic. Cette injection a pour objectif la collecte des données relatives aux performances du réseau en question. Par exemple, la mesure du temps de réponse, le suivi du chemin des paquets, etc.

Les données collectées permettent de surveiller les réseaux pour ensuite 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 [?].

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 [?].

I.3.2 Généralités sur les sondes Atlas

- Les sondes Atlas 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 Atlas 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 Atlas se situent dans différents emplacements dans le monde, cette répartition permet de diversifier les mesures (voir les sections des mesures [I.3.10](#) et [I.3.12](#)).
- Les sondes Atlas 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 période³.
- La participation au projet RIPE 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.
- RIPE Atlas simule le comportement de la couche IP. Par exemple, avec RIPE Atlas, il est possible de :
 - Suivre l'accessibilité d'une destination⁴ depuis différents emplacements dans le monde et depuis différents réseaux. Car les sondes Atlas 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 Atlas.

3. Si le nombre de crédits (voir la section des crédits [I.3.11](#)) disponibles le permet et qu'il n'y a pas de dépassement du nombre de mesures autorisé.

4. Une destination représente une adresse IP.

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

La section **I.5** reprend quelques cas d’utilisation et les sujets qu’on peut étudier en se basant sur le projet RIPE Atlas.

I.3.3 Les générations des sondes Atlas

Depuis leur création en 2010, les sondes Atlas ont connu trois générations du matériel : v1, v2 et v3. Le Tableau **I.1** reprend quelques caractérisations de ces trois générations et la Figure **I.4** montre le matériel utilisé dans chaque génération.

	v1	v2	v3
Matériel informatique	Lantronix XPort Pro [?]	Lantronix XPort Pro [?]	tp-link tl-mr3020
Début d’utilisation	2010	2011	2013
Mémoire RAM	8 Mo	16 Mo	32 Mo
Mémoire Flash	16 Mo	16 Mo	4 Mo
CPU	32-bit	32-bit	32-bit
Support du Wi-Fi	Non	Non	oui
Support du NAT	oui	oui	oui
Vitesses supportées	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 Atlas



FIGURE I.1 – Génération 1



FIGURE I.2 – Génération 2



FIGURE I.3 – Génération 3

FIGURE I.4 – Les trois générations des sondes Atlas

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 Atlas 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 Atlas. D’autre part, les sondes virtuelles peuvent être installées sans contraintes physiques ou organisationnelles. Parmi les inconvénients, une complexité sera ajoutée au système RIPE 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 Atlas

En principe, toutes les sondes Atlas collectent la même information, indépendamment de leur version du firmware. On trouve les mêmes attributs⁵ 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, entre les parenthèses. Cet identifiant sera utilisé dans la suite de ce document.

Il existe plusieurs versions du firmware :

- version 1 est identifié 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 firmware⁶ est 4610 (6).

I.3.5 La connexion des sondes Atlas à Internet

Les générations 1 et 2 des sondes Atlas 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.5. L'architecture du système RIPE Atlas est détaillée dans la section I.3.6.

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

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

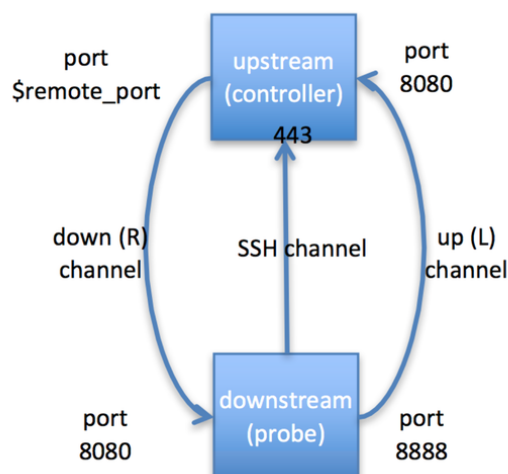


FIGURE I.5 – La connexion des sondes Atlas à l’infrastructure RIPE Atlas [?]

I.3.6 Architecture du système RIPE Atlas

Il existe deux catégories d’outils de surveillance du réseau : des outils matériels et d’autres logiciels. Les sondes Atlas 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 RIPE Atlas est conçu pour qu’il soit opérationnel de façon distribuée. La plupart des composants ont assez de connaissances pour remplir leurs rôles, sans nécessairement avoir besoin de connaître les états des autres composants du système. Cela assure 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.6 montre une vue d’ensemble de l’architecture du RIPE Atlas.

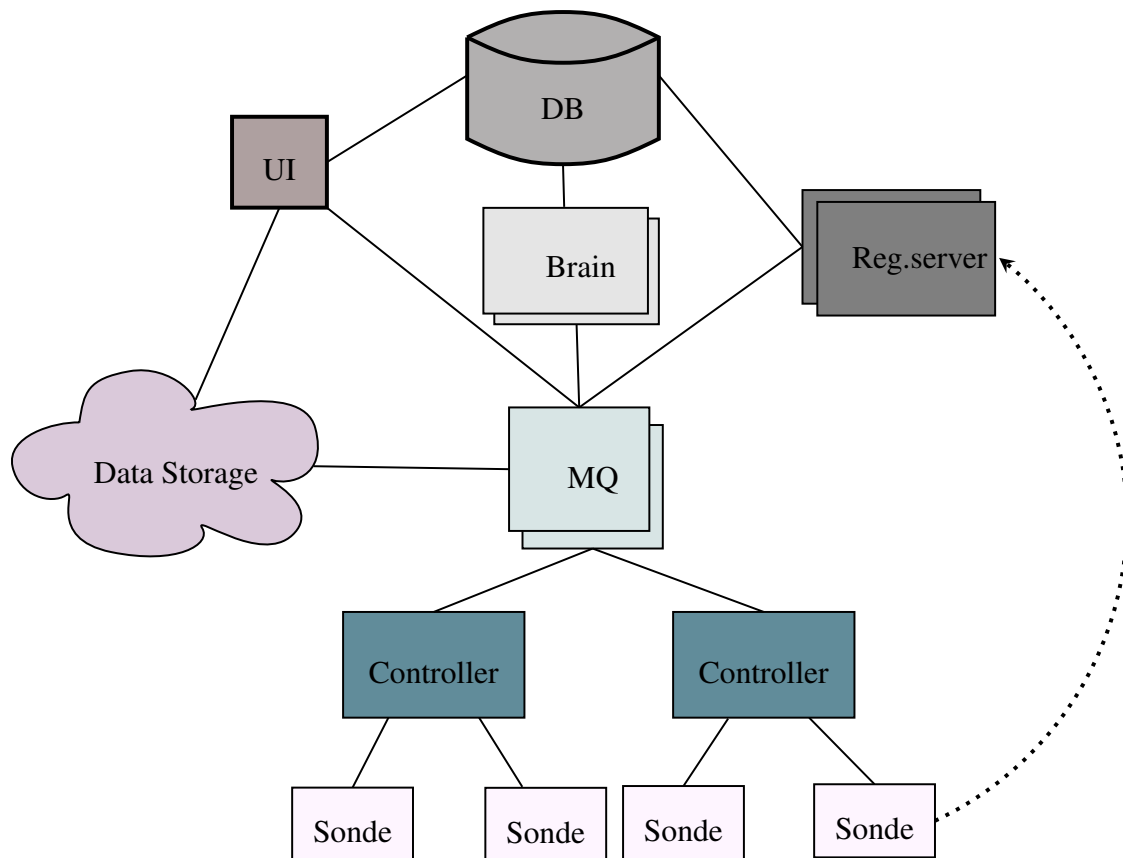


FIGURE I.6 – L’architecture du système RIPE Atlas

Source: Schéma repris du travail de Kisteleki [?]

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

Registration server (Reg.server) : c’est le seul point d’entrée de confiance pour les sondes Atlas. Son rôle est de recevoir toutes les sondes désirant se connecter au système RIPE Atlas. Ensuite, il redirige chaque sonde vers le contrôleur adéquat, qui est le plus proche de la sonde et que 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 recevoir les résultats et prévenir la sonde des mesures à effectuer. Le rôle du contrôleur est de communiquer avec les sondes, associer les mesures aux sondes en se basant sur la disponibilité de la sonde 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^a. »

a. Source : 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 du RIPE Atlas. Il gère la connectivité entre les composantes de l'infrastructure et assure 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 [?]. Elle traite les appels en provenance de l'API⁷ 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 Atlas et la négociation avec les contrôleurs pour voir la disponibilité des sondes Atlas.

DB : c'est une base de données SQL contenant toutes les informations du système RIPE 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. Cette technologie permet aussi d'effectuer des calculs d'agrégation périodiques et d'autres tâches.

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.7 illustre les étapes d'établissement de la connexion entre une sonde Atlas et l'infrastructure RIPE Atlas.

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

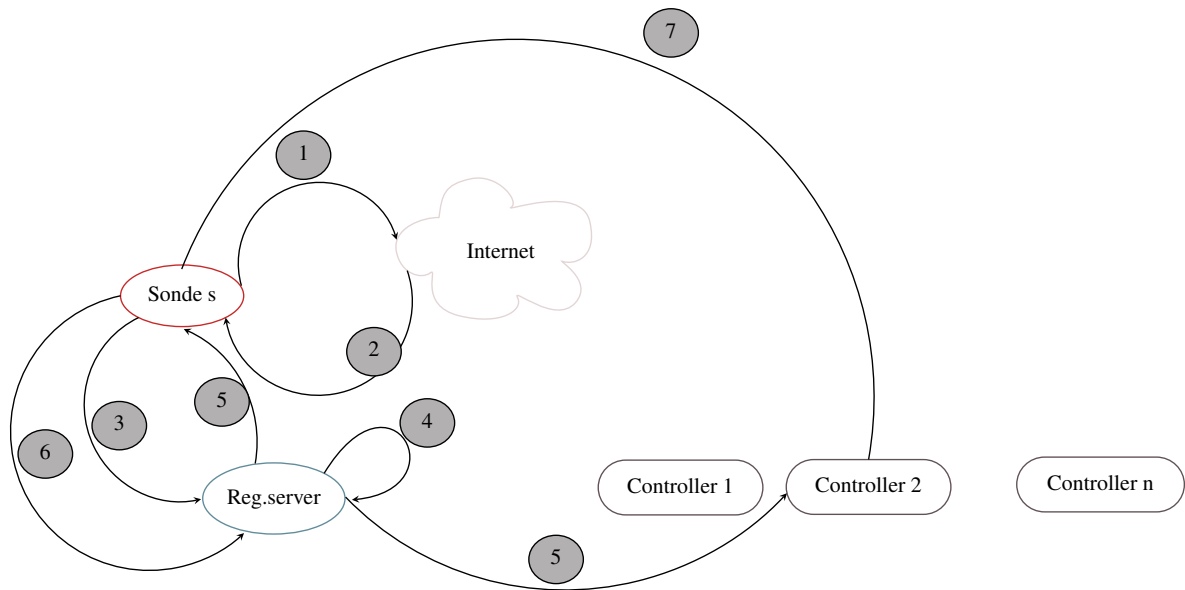


FIGURE I.7 – Les étapes d’établissement d’une connexion entre la sonde Atlas et l’architecture RIPE Atlas

Les étapes suivantes illustrent le déroulement de la connexion d’une sonde Atlas *s* à l’infrastructure RIPE Atlas.

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

La connexion précédemment établie permet aux sondes Atlas 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.3.7 Les sondes Atlas et la vie privée

La sonde Atlas 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 Atlas 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.

I.3.8 La sécurité dans RIPE Atlas

La connexion entre les composantes de l'infrastructure RIPE 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 ces sondes, les précautions suivantes sont prises :

- Les hébergeurs des sondes Atlas ne disposent d'aucun service qui leur permet de se connecter aux sondes (dans le sens de TCP/IP).
- Les sondes Atlas 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 Atlas sont chez les hébergeurs, il est impossible qu'elles soient résilientes au démontage. Cependant, si c'était le cas, cela ne devrait pas affecter les autres sondes Atlas.
- Toutes les communications au sein de l'infrastructure RIPE 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 Atlas peut être facilement mis à niveau ; la sonde Atlas est capable de vérifier l'authenticité d'une nouvelle version du firmware et cela via les signatures cryptographiques.

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

I.3.9 Les ancres VS sondes Atlas

Les ancres Atlas sont des dispositifs agissant comme cibles aux différentes mesures lancées par les sondes Atlas. Il est possible de planifier des mesures entre les ancres RIPE Atlas, ces mesures permettent de vérifier l'état des réseaux qui hébergent ces ancres. Les ancres Atlas peuvent être considérées comme cibles aux mesures suivantes :

- Ping.
- Traceroute.
- DNS : les ancres 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 Atlas vers une ancre Atlas est :

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

Les ancres 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 ancres Atlas.

	Sonde Atlas	Ancre Atlas
Mesures originaires de	oui	oui
Mesures à destination de	— ⁸	ping, traceroute, DNS, HTTP(S).
Nomination	—	structurée ⁹
Crédit gagnés	N	$10 * N$
Besoin en bande passante	léger	important
Coût : gratuite	oui	non ¹⁰

TABLE I.2 – Comparaison entre sondes et ancres RIPE Atlas

I.3.10 Les mesures intégrées : Built-in

Une fois une sonde Atlas connectée, elle lance automatiquement un ensemble de mesures prédéfinies, appelées *Built-in Measurements*. Les mesures personnalisées sont détaillées dans la section I.3.12. 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 Atlas.

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 de temps.

De base, les sondes Atlas assurent les mesures intégrées suivantes :

- Les informations sur la configuration du réseau dans lequel la sonde Atlas est déployée.
- L'historique de la disponibilité de la sonde Atlas.
- Les mesures du RTT (Round Trip Time) par 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.

8. — : Non disponible.

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

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

- 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 ¹¹.

En plus des mesures intégrées, les sondes Atlas peuvent effectuer des mesures personnalisées. Ces mesures peuvent être lancées via l'interface web [?] ou bien via HTTP REST API. Toutefois, la planification des mesures personnalisées nécessite l'acquisition de ce qu'on appelle les "crédits" au sens RIPE Atlas.

I.3.11 Le système de crédits Atlas

Le système de crédits RIPE Atlas est une sorte de reconnaissance de la contribution des participants à ce projet. Un hébergeur d'une sonde Atlas 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.3.12). Les personnes ayant gagné des crédits peuvent les transférer vers une autre personne ayant besoin de ces crédits. Les crédits peuvent être obtenus via :

- L'hébergement d'une sonde Atlas ; à chaque utilisation d'une sonde, son hébergeur reçoit un nombre de crédits. La connexion d'une sonde Atlas au système durant une minute apporte 15 crédits.
- L'hébergement d'une ancre Atlas ¹².
- La recommandation à une personne d'héberger une sonde Atlas.
- En étant un sponsor du RIPE NCC. Le parrainage des sondes Atlas est disponible 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 RIPE 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 du type de la mesure 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).

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

12. Les ancres Atlas sont décrites dans la section I.3.9.

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.

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

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.3.12 Les mesures personnalisées : User Defined measurement

En plus des mesures intégrées, par défaut, dans une sonde Atlas, il est possible de planifier des mesures personnalisées. Ce sont les mêmes types de mesures : ping, traceroute, HTTP Get, SSLCert, DNS, NTP et TLS. Cette planification coûte des crédits, en effet, il faut avoir assez de crédits pour lancer des 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 Atlas réalisant la mesure ;
- la fréquence de la mesure et sa durée.

Chaque mesure est suivie via son état. Plusieurs états à distinguer : *specified*, *scheduled*, *ongoing*, *stopped*, *Forced to stop*, *no suitable probes* et enfin *failed*.

I.3.13 La sélection des sondes Atlas

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

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

Il existe une autre manière de regrouper les sondes : avec des étiquettes. Le système d’étiquettes sert comme indicateur des propriétés, des capacités, de la topologie du réseau et d’autres classifications des sondes. On distingue les étiquettes système et utilisateur. Chaque nom d’étiquette est lisible par un humain.

Les étiquettes utilisateurs sont associées à une sonde librement par son hébergeur. Les étiquettes système sont attribuées uniquement par l’équipe RIPE Atlas et sont mises à jour périodiquement, à priori chaque 4 heures. Des exemples d’étiquettes système sont disponibles sur RIPE Atlas¹³.

I.3.14 Les sources de données Atlas

Les sondes Atlas génèrent trois types de données : leurs détails de connexions d’un jour donné, leurs résultats des mesures intégrées et personnalisées et les descriptions des mesures effectuées (meta-data).

Premièrement on trouve les données sur les sondes Atlas par jour. Les détails sur les sondes reprennent les informations des connexions, des réseaux et autres. A priori, les détails des connexions des sondes sont disponibles pour la période du 13 mars 2014 jusqu’à ce jour¹⁴, un fichier JSON par jour. Les données de certains jours sont manquantes. La totalité des archives se trouve dans [?]. La taille d’une seule archive est entre 120 KB et 921 KB.

Deuxièmement, les résultats des mesures sont aussi archivés dans un serveur FTP. Seules les données des derniers 30 jours sont conservées en archives¹⁵. Les fichiers ont été nommés jusqu’au 15 mars 2018 de façon structurée comme suit :

```
$TYPE-$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.

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

14. 15/08/2018.

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

En considérant toutes les possibilités des types, la quantité de données générées quotidiennement est environ 25 Go¹⁶ 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 (identifiant > 1, 000, 000).

Streaming API propose un service de récupération des résultats de mesures en temps réel, depuis les sondes publiques. Ainsi, elle fournit continuellement de nouveaux résultats en temps réel, obtenus par les sondes Atlas publiques, via une connexion de type HTTPS web-socket active tout le temps.

Troisièmement, on trouve des archives sauvegardées chaque semaine, elles décrivent les méta-datas des mesures. Une ligne objet JSON pour chaque mesure publique. Au moment de la consultation, la taille de chaque archive était entre 124 Mo et 1.5 Go. L'accès à ces données se fait de deux façons, via le téléchargement direct depuis un serveur FTP ou bien via streaming API. Les noms des archives sont bien structurés.

I.3.15 Les limitations du RIPE Atlas

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

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

Afin d'éviter la surcharge des sondes et de l'infrastructure, RIPE 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. RIPE 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. Pour la version 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, RIPE Atlas limite cette consommation à 1.000.000 crédits.

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

I.3.16 Confiance aux données Atlas

De nombreux travaux ont exploité les données Atlas, cependant, peut-on faire confiance à la qualité de ces données ? les données sont-elles complètes ?

La question de la complétude des données est plus présente pour les mesures périodiques, celles qui se déroulent pendant une durée d et à un intervalle i . W. Shao et al. [?] ont traité les mesures manquantes. L'approche qu'ils ont adopté repose sur la corrélation entre l'absence de certaines mesures et les périodes durant lesquelles les sondes Atlas ont été déconnectées. Pour précision, RIPE Atlas maintient les détails des connexions/déconnexions des sondes Atlas. W. Shao et al. [?] 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, leur approche se base sur l'attribut *timestamp* qui est présent dans chaque résultat de mesure et dans les états de connexions.

Nous avons discuté des limitations du RIPE Atlas en terme de mesures autorisées par jour. Cela n'empêche qu'il est possible qu'un nombre important de mesures soit effectué. De plus, plus d'un utilisateur peut s'intéresser à la même sonde Atlas. C'est la question traitée dans le travail de T. Holterbach et al. dans [?] : 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 Atlas 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 Atlas. 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 d'Atlas. Dans ce cas, l'amélioration du matériel ne peut pas résoudre le problème.

I.4 Projets existants de mesures d'Internet

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

I.4.1 Test Traffic Measurement Service

Avant l'arrivée du RIPE Atlas, Le RIPE NCC (Réseaux IP Européens Network Coordination Centre) a assuré la mesure de la connectivité entre les réseaux via d'autres plateformes, comme la plateforme Test Traffic Measurement Service (TTM). Il s'agit d'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. De plus, il n'a pas l'accès aux données du réseau dans lequel il est mis en place.

Ce service a été assuré et géré, pendant une période de 6 ans, par une équipe au sein du RIPE NCC. Les fonctionnalités assurées par ce service étaient de tester l'accessibilité à une destination via le *ping*, ainsi, les mesures effectuées étaient indépendantes des applications, elles dépendaient du réseau lui-même. RIPE NCC a arrêté la maintenance du TTM depuis le 1 juillet 2014 [?].

I.4.2 ProbeAPI

ProbeAPI [?] est une plateforme de mesure de l'état d'un réseau, cette plateforme couvre 170 pays et des milliers d'ISPs. *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 : dans des 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 [?] entre les sondes Atlas 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 Atlas 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.* »^a

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

RIPE 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

Malgré le niveau de couverture assuré par ProbeAPI, cependant 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.

I.4.3 Archipelago

Archipelago (Ark) [?] est l'infrastructure de mesures actives du CAIDA [?]. Elle est au service des chercheurs en réseau depuis 2007. L'objectif de ce projet est de couvrir un maximum de régions afin de collecter un maximum de mesures. Ensuite, produire des visualisations qui améliorent la vue globale de l'Internet. Pour précision, c'est un Raspberry Pi 2nd gen.

I.4.4 DIMES

DIMES [?] 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.4.5 SamKnows

SamKnows [?] est une plateforme globale des performances d'Internet, elle regroupe les ISPs, ingénieurs, universitaires, 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.5 Quelques cas d'utilisation des données collectées par les sondes Atlas

Plusieurs travaux ont exploité les données collectées par les sondes Atlas. 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 Atlas 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 RIPE 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 aux données par type de mesure : traceroute, ping, HTTP, etc. Dans ce qui suit, nous allons présenter brièvement quelques travaux par thème.

I.5.1 Détection des coupures d'Internet

Les données collectées par les sondes Atlas ont permis de valider certaines coupures d'Internet. Par exemple, la coupure concernant le point d'échange AMS-IX (Amsterdam Internet Exchange). En 2015, R. Kisteleki et al. [?] ont évalué l'état des pings en provenance des sondes Atlas à destination de trois ancres Atlas 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. Ils 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 DNS, ils ont constaté l'absence des données DNS sensées être collectées par les ancres Atlas à destination du K-root pendant la période de la coupure.

I.5.2 Aide à la prise de décision

L'utilisation des sondes Atlas n'est pas limitée au domaine de recherche seulement, elle a permis aussi d'aider à la prise de décision pour certaines implantations et pour la mise en place des équipements comme les routeurs, les data-centers, les IXPs, etc.

Les ingénieurs de *Wikimedia Foundation* et du RIPE NCC ont collaboré dans un projet [?] pour étudier la latence vers les sites du Wikimedia. L'idée était d'exploiter la distribution des sondes Atlas 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 futurs emplacements de leurs data-centers.

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 [?]. Après avoir analysé les données des mesures collectées par les sondes Atlas, 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 dans ce continent.

I.5.3 Le suivi des censures

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 aussi élaboré un aperçu comparatif des différents outils permettant de mesurer les réseaux [?]. 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 Atlas.

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 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 Atlas, ils ont planifié des mesures vers plusieurs destinations et depuis un nombre de sondes. Ces mesures sont reprises en détail 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 [?]

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

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.5.4 Le suivi des performances d'un réseau

Les ancrs Atlas

Les ancrs Atlas ont des capacités avancées que les sondes Atlas. Les ancrs 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 Atlas, a développé un outil disponible au public¹⁷. A partir des données collectées par les ancrs Atlas, 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 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 Atlas. La Figure I.8 reprend les résultats obtenus. Aucun renseignement sur la période des données. Pour

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

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.

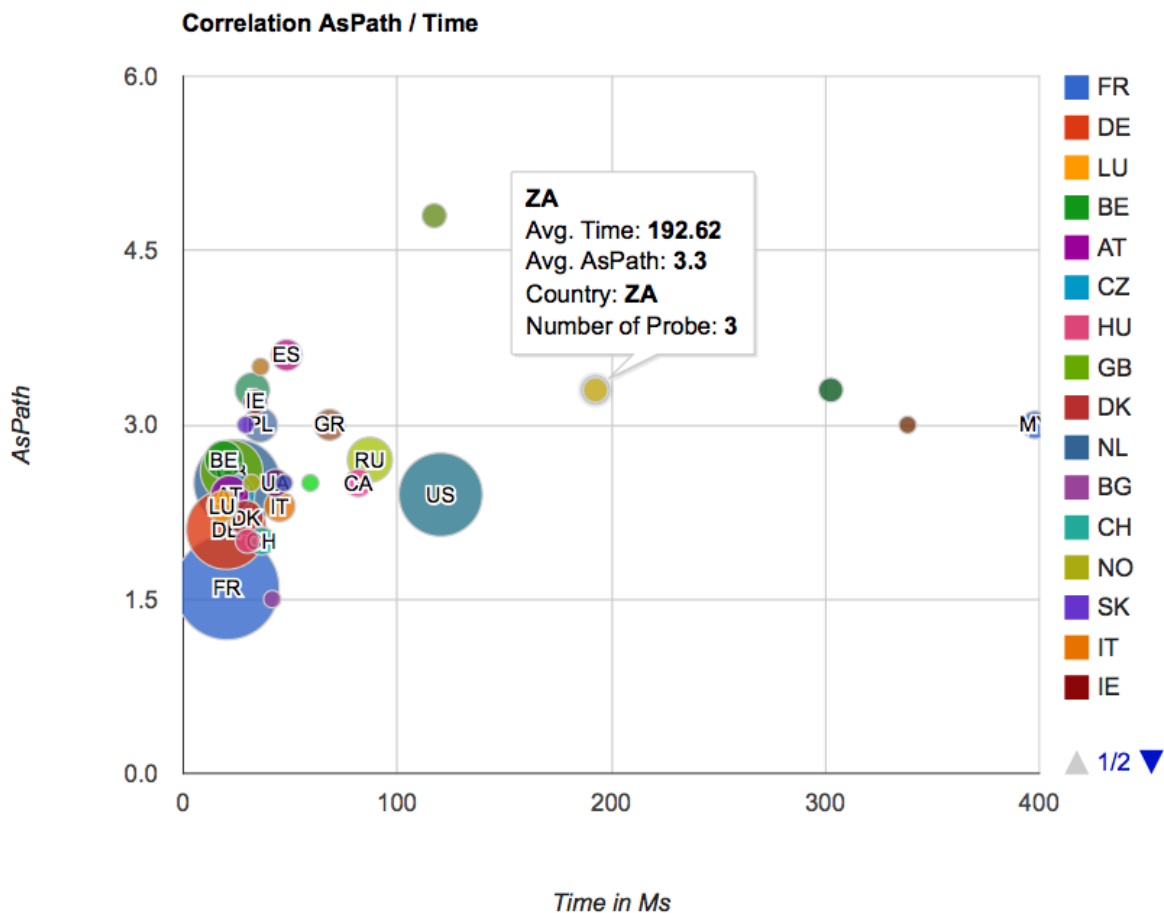


FIGURE I.8 – La corrélation entre la moyenne des AS paths et la moyenne des RTTs [?]

Ensuite, S. Gasmi a mesuré le RTT entre des sondes Atlas dans le monde et son ancre, il a aussi visualisé le nombre de sauts parcourus entre des sondes Atlas à travers le monde et son ancre. Ces deux visualisations permettent d'avoir une idée sur la latence entre certains pays et le pays de l'ancre en question. Plus de détails sur l'approche sont disponibles dans [?].

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 [?]. La Figure I.9 présente un exemple de la visualisation proposée. Ce résultat concerne la mesure 1663314¹⁸. Ce sont des traceroutes à destination de l'adresse 213.171.160.1, effectués entre le

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

02/05/2014 13 : 00 et le 03/05/2014 15 : 00.

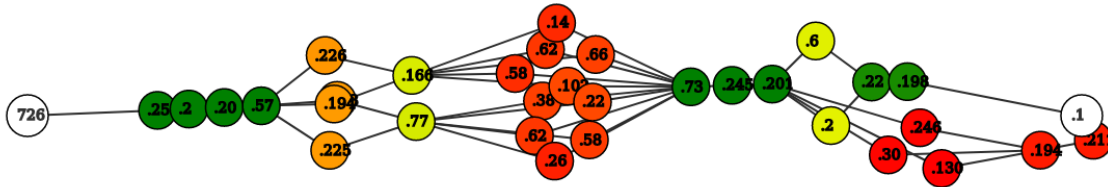


FIGURE I.9 – Visualisation des changements des chemins traceroute [?]

BGP+traceroute

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 GitHub [?, ?].

BGP Atlas Monitor (BAM)

Le projet *BAM* vise 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 Atlas. Le code source est disponible sur GitHub [?]. En fournissant un ASN (identifiant d'un AS), *BAM* récupère les préfixes IPv4 et IPv6 et leur visibilité et il montre 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 [?], 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.5.5 Le suivi des détours dans un trafic local

Dans leur travail [?], E. Aben et al. avaient l'objectif de voir comment les mesures du RIPE 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 est d'analyser les chemins identifiés dans le trafic d'Internet entre les sondes Atlas dans un pays donné et essayer d'identifier si le trafic traverse les 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 Atlas et 9 ancrés. 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 Atlas 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 [?] s'intéresse au trafic depuis et vers une sonde en France en se basant sur l'étude [?].

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

I.5.6 Visualisation : indicateurs et dashboard

L'objectif de certains travaux est d'exploiter les données collectées par les sondes Atlas pour concevoir des tableaux des indicateurs. Par exemple, à partir des données de connexion/déconnexion des sondes Atlas, 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.

La liste des travaux basés sur le projet RIPE 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.6 Conclusion

Dans ce chapitre, nous avons présenté les sondes Atlas 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 II.

Chapitre II

La 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 de Fontugne et al. [?]. Nous avons choisi de réutiliser ce travail qui se base sur les traceroutes collectés par les sondes Atlas, et ce en vue de l'évaluer avec quelques technologies Big Data.

II.1 Présentation générale

Dans leur travail [?], Fontugne et al. ont exploité la distribution répandue des sondes Atlas dans le monde afin d'étudier un des problèmes relatifs aux performances des réseaux informatiques.

Le travail de Fontugne et al. [?] se base principalement sur une des mesures effectuées par les sondes Atlas : la requête traceroute, et ce dans le but d'analyser le délai d'un lien topologique sur les réseaux informatiques et de tracer son évolution. L'idée de ce travail est de collecter les résultats des requêtes traceroute effectuées par les sondes Atlas, d'en déterminer une valeur de référence du délai du lien en question sur base de l'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 qu'à partir d'avoir une référence assez représentable de l'état réel du lien. A savoir que le déclenchement de cette comparaison est personnalisable.

En pratique, il est difficile d'avoir une idée globale et exacte sur la topologie d'Internet. Toutefois, les opérateurs des réseaux informatiques disposent d'un aperçu de l'état des entités qui forment leurs réseaux, les relations entre ces entités ainsi que les éventuels problèmes. Avec la distribution abondante des sondes Atlas dans le monde en terme de type d'adressage : sondes Atlas supportant seulement l'adressage IPv4, d'autres qui supportent en plus l'adressage IPv6, en terme de la diversité géographique, la diversité en terme d'ASs hébergeant les sondes Atlas, etc, il était possible d'aborder les délais dans les réseaux informatiques à travers de nouvelles approches, reposées sur des fondements statistiques. Parmi les points forts de l'analyse menée par Fontugne et al., c'est la possibilité de valider les méthodes proposées avec des événements marquants sur Internet.

Le travail de Fontugne et al. [?] reprend trois méthodes basées sur les traceroutes collectées par les sondes Atlas :

1. la détection des changements des délais que subissent les liens intermédiaires dans les

traceroutes ;

2. la conception d'un modèle de forwarding pour un routeur donné. Ce modèle prédit l'acheminement du trafic afin d'identifier les routeurs et les liens en panne dans le cas d'un problème de perte de paquets ;
3. la création d'un score par Système Autonome afin d'évaluer l'état de ce dernier.

Dans la suite de ce travail, nous reprenons seulement la première méthode. Il s'agit d'étudier le délai d'un lien topologique ; c'est le délai entre deux routeurs adjacents sur Internet. Nous aurions aimé avoir suffisamment de temps pour pouvoir aborder les autres méthodes.

II.2 Pourquoi analyser les délais des liens réseaux

Il existe un bon nombre de sujets à traiter en exploitant les données collectées par les sondes Atlas. Quelques exemples de sujets ont été déjà présentés dans la section I.5. Notre choix de reprendre le travail de Fontugne et al. [?] a été fait après avoir parcouru un ensemble de travaux basés sur le projet RIPE Atlas.

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 travaux¹. 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 leur travail [?] les contraintes relatives à ce sujet : la difficulté d'avoir 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 [?] pour l'évaluation de quelques technologies Big Data s'inscrit dans les travaux traitant les performances des réseaux. 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 la base de données du RIPE Atlas. Ainsi, il n'y a pas besoin de lancer des mesures qui nécessitent la possession d'assez de crédits (voir la section I.3.11). 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 RIPE Atlas est active en nombre de travaux publiés. Dans certains cas, ces travaux reprennent seulement les résultats. Pour certains, la méthodologie est bien détaillée. Pour d'autres, la méthodologie adoptée est très brève. En ce qui concerne le travail choisi, la méthodologie est bien détaillée.

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

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 [?]. 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 [?], 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 L'étude 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 échantillons² considérés est grande. Afin de surveiller un grand nombre de liens sur Internet, il faut avoir un grand nombre de sondes Atlas avec une certaine diversité. Les deux catégories de traceroutes utilisés dans le travail de référence sont :

- *builtin* : ce sont les traceroutes effectués par toutes les sondes Atlas vers les instances des 13 serveurs DNS racines. Les traceroutes sont effectués chaque 30 minutes. En pratique, certains serveurs racines DNS déploient l'anycast. Au moment de la réalisation du travail de référence³, c'étaient des traceroutes vers 500 instances des serveurs DNS racines ;

DNS Anycast est une solution utilisée pour accélérer le fonctionnement des serveurs DNS. Les serveurs DNS adoptant cette approche fournissent des temps de réponse plus courts, et ce partout dans le monde. Les requêtes en provenance de l'utilisateur sont redirigées vers un nœud adéquat suivant un algorithme prédéfini.

- *anchoring* : ce sont les traceroutes effectués par environ 400 sondes Atlas à destination de 189 serveurs⁴ et ce chaque 15 minutes.

En ce qui concerne le nombre de traceroutes analysés dans le cadre du travail de référence, le Tableau II.1 reprend plus de détails : le nombre de traceroutes ainsi que le nombre de sondes impliquées dans ces traceroutes, et ce pour les deux types d'adressages (IPv4 et IPv6). Ces chiffres correspondent la période du 1 mai à 31 décembre 2015.

	Nombre de traceroutes	Nombre de sondes
IPv4	2.8 milliards	11.538
IPv6	1.2 milliards	4.307

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

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

3. Année 2017.

4. Sondes Atlas ayant des fonctionnalités avancées.

Pour précision, l'étude des délais des liens ne concerne pas les adresses privées, ainsi, le suivi des délais ne concerne pas les réseaux privés. De plus, ce suivi se base sur les requêtes de type traceroute, et traceroute reprend une partie de la topologie de l'Internet. En effet, les liens considérés sont ceux topologiques et ne sont pas les liens physiques.

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

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

Définition du RTT différentiel

Avant de définir le RTT différentiel d'un lien, soit la définition du RTT :

ICMP (Internet Control Message Protocol) est un protocole utilisé pour véhiculer des messages de contrôle sur Internet.

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 de cette destination. C'est une métrique 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 quering et du traitement.

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 Δ_{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 est illustré dans la Figure II.1 (b).

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

Le RTT différentiel Δ_{PBC} du lien BC est donné par l'équation II.1 :

$$\Delta_{PBC} = \delta_{BC} + \varepsilon_{PBC} \quad (\text{II.1})$$

où δ_{BC} est le délai du lien BC et ε_{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).

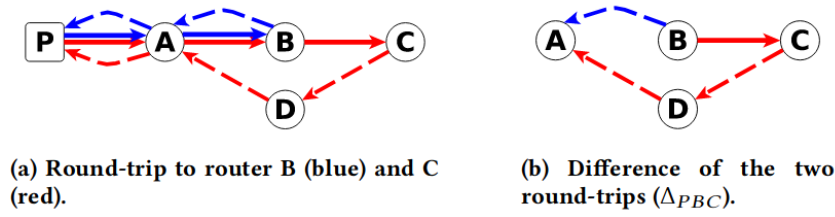


FIGURE II.1 – (a) Le RTT entre la sonde P et les routeurs B et C. (b) La différence entre les chemins de retour depuis les routeurs B et C vers la sonde P. Source : [?]

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 : $\delta_{BC} + \varepsilon_{PBC}$.

La valeur de δ_{BC} dépend de l'état des deux routeurs B et C et ne dépend pas de la sonde P. Tandis que la valeur de ε_{PBC} dépend de la sonde P ; la destination des deux chemin de retours.

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 ε_{P_iBC} différent depuis B et depuis C. Les n RTTs différentiels Δ_{P_iBC} du lien BC pour chacune des sondes P_i partagent la même composante δ_{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 ε_{P_iBC} sont indépendantes. L'indépendance de ces valeurs implique que la distribution Δ_{P_iBC} est estimé stable au cours du temps si δ_{BC} est constant. Toutefois, un changement significatif de la valeur de δ_{BC} influence les valeurs des RTTs différentiels, ainsi, la distribution des RTTs différentiels Δ_{P_iBC} change. D'où l'idée de l'évolution du délai d'un lien est déduit de l'évolution de son RTT différentiel.

La détection des anomalies dans les délais repose sur le théorème central limite (TCL). Ce théorème annonce que si on a une suite de variables aléatoires X_i indépendantes ayant la même espérance μ et la même variance σ^2 , la moyenne de ces variables aléatoires est une variable aléatoire qui suit une loi normale. Ce théorème a été utilisé pour calculer la médiane des RTTs différentiels d'un lien d'une période d_k . Les auteurs du travail de référence ont ajusté ce théorème, ils ont utilisé la médiane au lieu de la moyenne. Cet ajustement a été validé en évaluant les performances de la moyenne ainsi que de la médiane sur des échantillons des RTTs différentiels des liens dont ils connaissent leur état en matière de délais.

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

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

différentiel, il y a aussi l'identification d'éventuelles anomalies pour ce lien.

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 d'un nombre de paramètres. Ces paramètres ont des valeurs par défaut définies dans le travail de référence. Ci-dessous les paramètres à ajuster afin d'obtenir l'évolution des RTTs différentiels d'un lien ainsi que les éventuelles anomalies.

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. Les traceroutes sont obtenus en utilisant l'API fournie par RIPE Atlas.

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

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

timeWindow : ce paramètre est exprimé en secondes. Il correspond à la durée des périodes de l'analyse. Ces périodes ont la même durée. Autrement dit, la durée entre *start* et *end* est divisée sur cette même taille : *timeWindow*, c'est qui est illustré dans la Figure II.2. Par défaut, *timeWindow* a 3600 secondes comme valeur.

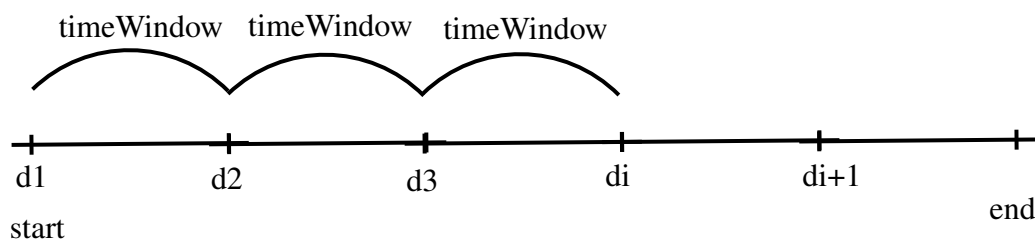


FIGURE II.2 – Illustration des périodes de l'analyse entre la date de début et la date de fin

minSeen : ce paramètre est un entier, il indique le déclenchement de la comparaison du RTT différentiel de référence d'un lien avec sa valeur courante. Par exemple, si *minSeen* = 24 et *timeWindow* = 3600 (1 heure), cela implique que la référence de tout lien est construite des valeurs obtenues tout au long la durée $24 * timeWindow$, donc après avoir analysé les traceroutes d'une journée. A la 25^{ème} *timeWindow*, la détection des anomalies s'effectue.

alpha : noté α , $\alpha \in [0, 1]$, c'est le paramètre de la moyenne mobile exponentielle calculée.
 « Les méthodes de lissage exponentiel sont un ensemble de techniques empiriques de prévision qui accordent plus ou moins d'importance aux valeurs du passé d'une série temporelle⁵. »

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

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

m_t la médiane des RTTs différentiels observée pour l durant le *timeWindow* t .

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

Pour précision, $\{m_t\}$ et $\{\bar{m}_t\}$ désignent deux ensembles différents. La taille de chacune de ces deux ensembles est la taille des périodes d_k . 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 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 α .

Le paramètre α contrôle l'importance des mesures précédentes par rapport aux mesures récentes. De ce fait, « plus α est proche de 1 plus les observations récentes influent sur la prévision, à l'inverse un α proche de 0 conduit à une prévision très stable prenant en compte un passé lointain. »[?]. Dans le travail de référence, le paramètre α prend par défaut 0.01 comme valeur.

confInterval : l'intervalle de confiance est défini comme représentant les valeurs probables que peut prendre une moyenne, si l'on accepte une marge d'erreur définie à l'avance à travers la valeur de la marge d'erreur. Il existe plusieurs méthodes pour calculer l'intervalle de confiance d'une proportion. Parmi les critères impliqués sur le choix de la méthode de calcul, on note le nombre total d'expériences.

le RTT différentiel courant et celui de référence d'un lien donné sont représentés, en pratique, par des intervalles de confiance. le calcul des intervalles de confiance est approché par le score 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.

II.4.3 L'évolution du RTT différentiel d'un lien et la détection des anomalies

Nous avons décrit les étapes principales de détection dans la section II.4.1. Nous présentons dans ce qui suit l'algorithme de détection en détail. Les paramètres d'entrée de cet algorithme sont détaillés dans la section II.4.2. L'algorithme fournit en résultat l'évolution du RTT différentiel de chaque lien, tout au long de la période de l'analyse, ainsi que son RTT différentiel

5. Source : <https://perso.math.univ-toulouse.fr/lagnoux/files/2013/12/Chap6.pdf>, consultée le 30/09/2018.

de référence. Dans le cas d'anomalies détectées, celles-ci sont identifiées tout en précisant la date pendant laquelle l'anomalie a été détectée ainsi que la valeur du RTT différentiel ayant engendré cette anomalie.

Un traceroute est un ensemble de sauts h_i dans l'ensemble H auxquels sont joints l'identifiant de la sonde ayant effectué la requête traceroute et la destination de la requête. Chaque saut est décrit par un ensemble de signaux dans S . Chaque signal décrit le routeur ayant émis une réponse à la sonde parmi les routeurs traversés avant d'atteindre la destination finale. Pour le saut h_i , on note trois⁶ signaux $s_{i,j}$, $j \in [1, 3]$ dont le routeur émettant le signal est $from_{i,j}$ et son RTT est égal à $rtt_{i,j}$. C'est ce qu'illustre la Figure II.3.

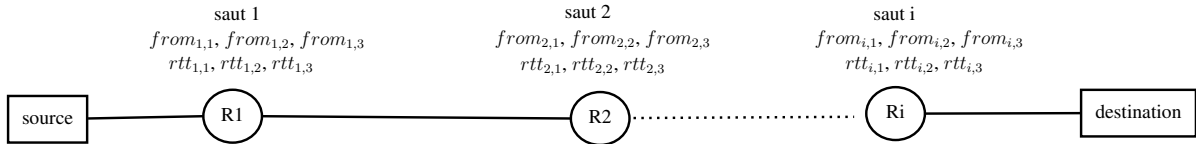


FIGURE II.3 – Illustration des sauts d'un traceroute avec leurs informations

II.4.4 Processus de détection des anomalies

Soient $d_1, d_2, d_k, \dots, d_N$ l'ensemble D de N périodes entre *start* et *end* avec $k \in [1, N]$. La différence entre le début de la période d_{k+1} et le début de la période d_k est égale à *timeWindow* pour tout $k + 1 \leq N$.

Le processus de la détection des anomalies passe par plusieurs étapes. D'abord on regroupe les traceroutes à analyser par période d_k (étape 1). Ensuite, on prépare les traceroutes de toute période d_k en y appliquant un nombre d'opérations (étapes 2 à 6). A la fin de la préparation des traceroutes de toutes les périodes, on ne considère que les données qui concernent le lien à suivre. Ce sont les données qui servent à la comparaison des délais d'un lien avec les valeurs de référence (étapes 7 à 9). Les étapes de détection sont les suivantes :

1. Regroupement des traceroutes par période d_k . En effet, chaque période d_k est associée à un ensemble de traceroutes ayant été effectués entre d_k et $d_{k+1} - 1$, avec $k \in [1, N]$, $k + 1 \leq N$ et d_k est exprimé en secondes (Unix Timestamp). Les étapes (2 à 6) concernent les traceroutes de chaque d_k .

2. Vérification de la validité de chaque traceroute. Les traceroutes sont filtrés en prenant en considération les points suivants :

- élimination des traceroutes échoués complètement ;
- élimination des signaux contenant une adresse IP privée ;
- élimination des signaux qui ne contiennent pas un RTT ou qui contiennent un RTT négatif ;
- élimination des signaux échoués.

Il existe deux sortes d'échecs dans un traceroute : échec complet et échec partiel. Dans le premier, la sonde Atlas ne réussit pas à atteindre la destination, dans ce cas, la liste des sauts est vide. Dans le deuxième cas, l'échec peut concerner un ou plusieurs saut, ou bien il peut

6. Le nombre trois peut varier dans certains cas. L'outil de détection conçu est adapté à tout nombre de signaux.

concerner un, deux ou trois signaux d'un saut. A la fin de cette étape, nous obtenons une liste de traceroutes.

3. Calcul de la médiane des RTTs par saut. Pour tout saut d'un traceroute, on calcule la médiane des RTTs par $from_{i,j}$. Soit le saut $h_i = \{s_{i,j} | s_{i,j} \in S\}$ La médiane des RTTs d'un saut h_i est :

$median_rtt(h_i) = \{median(\{rtt_{i,j}\})\}$ pour tout $rtt_{i,j}$ ayant la même valeur du $from_{i,j}$. Autrement dit, le nouveau saut du traceroute est reconstruit en regroupant les signaux par adresse IP ($from_{i,j}$) et ensuite en calculant la médiane de leurs RTTs ($rtt_{i,j}$). Par exemple, si on a un saut dont les trois signaux sont en provenance du même routeur, le nouveau saut est présenté par ce routeur avec un RTT obtenu en calculant la médiane des trois RTTs.

4. Inférence des liens topologiques par traceroute. Un lien est formé par chaque paire de routeurs consécutifs dans un traceroute. De manière générale, la Figure II.4 illustre la constitution des liens possibles dans un traceroute. Soient $R_{a,i}$, avec $i \in 1, 2, \dots, N$, l'ensemble de routeurs pour le saut a et $R_{b,j}$, avec $j \in \{1, 2, \dots, M\}$, l'ensemble de routeurs pour le saut b , avec N et M deux entiers. Les liens construits sont ceux partant de tout $R_{b,j}$ vers tout $R_{a,i}$, où a et b sont deux sauts consécutifs. A l'issue de cette étape, pour tout traceroute, on obtient la liste des liens possibles tout en reprenant des informations générales de la requête traceroute.

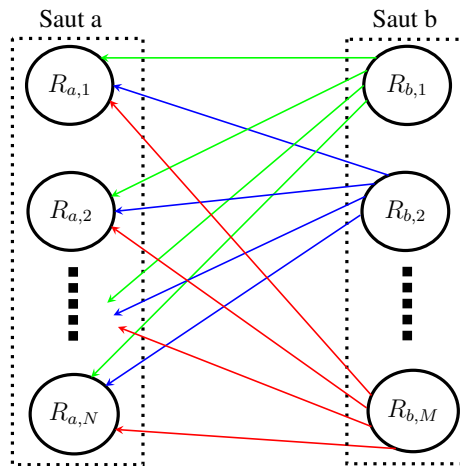


FIGURE II.4 – Inférence des liens possibles entre les routeurs des deux sauts consécutifs $R_{a,i}$ et $R_{b,j}$

5. Caractérisation des liens avec leurs RTTs différentiels. A cette étape, on calcule le RTT différentiel de chaque lien, au sein d'un traceroute donné, en calculant la différence entre les RTTs des deux routeurs impliqués dans le lien en question. En plus du RTT différentiel, on note aussi la sonde Atlas ayant effectué la requête traceroute où le lien a été identifié.

6. Fusion des informations d'un lien Etant donné qu'un lien (IP1, IP2) peut être identifié plusieurs fois pendant une même période d_k d'une part, et le lien (IP2, IP1) est similaire au lien (IP1, IP2) d'autre part, la fusion permet de construire une nouvelle distribution des RTTs différentiels caractérisant le lien (IP1, IP2) qui reprend les RTTs différentiels du lien (IP1, IP2) ainsi que ceux du lien (IP2, IP1).

Dans le travail de référence, la similarité de deux liens implique que les RTTs différentiels qui caractérisent un lien (IP2, IP1) caractérisent aussi le lien (IP1, IP2) ; ces deux liens partagent les mêmes routeurs.

A la fin de l'étape 6, tous les traceroutes sont préparés. A présent, l'objectif est d'identifier les dates pendant lesquelles des anomalies ont été détectées. Pour ce faire, l'idée du travail de référence est de conserver, pour un lien donné, une référence du RTT différentiel médian. Cette référence est d'abord comparée avec la médiane courante des RTTs différentiels, puis, mise à jour au fur et à mesure tout au long de la période de l'analyse.

7. Calcul de la médiane des RTTs différentiels et l'intervalle de confiance courant du lien analysé. Pour un lien donné, on calcule la médiane des RTTs différentiels d'une d_k , ensuite on calcule les deux bornes de l'intervalle de confiance courant pour ce d_k .

8. Mise à jour de la médiane et de l'intervalle de confiance de référence du lien analysé. La médiane des RTTs différentiels de référence ainsi que l'intervalle de confiance de référence sont mise à jour tant que la période déclenchant la détection des anomalies n'est pas atteinte. La mise à jour de la référence : médiane et les bornes de l'intervalle de confiance se base sur la moyenne mobile exponentielle. Cette méthode prend en compte les valeurs antérieures ainsi que celles récentes en étant dirigée par le paramètre *confInterval* décrit dans la section II.4.2.

9. Comparaison des intervalles de confiance La comparaison de l'état courant du lien avec celui de référence est effectuée en analysant le chevauchement d'intervalles de confiance courant et de référence. Le délai d'un lien est jugé normal si son intervalle de confiance courant est inclus dans l'intervalle de confiance de référence. C'est le cas 1 dans la Figure II.5, *referenceLow* et *referenceHeight* sont les bornes de l'intervalle de confiance de référence. *currentLow* et *currentHeight* sont les bornes de l'intervalle de confiance courant.

D'après le travail de référence, on distingue quatre cas possibles illustrés dans la Figure II.5 :

Cas 1 : le délai du lien est normal.

Cas 2 : le délai du lien est anormal.

Cas 3 : le délai du lien est anormal.

Cas 4 : le délai du lien est anormal.

Dans le cas où le délai est jugé anormal, on introduit ce qu'on appelle la *dévi*ation. Cette métrique caractérise l'anomalie détectée. Elle est calculée différemment dans le cas où le délai est anormal.

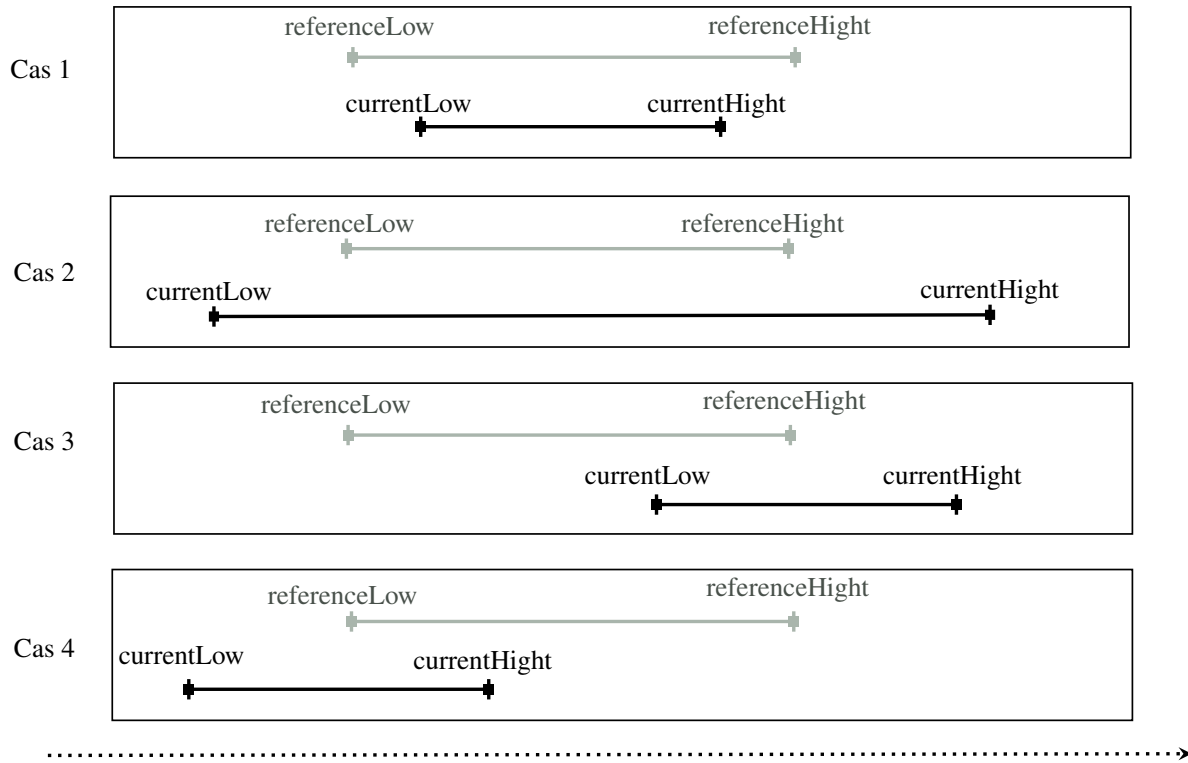


FIGURE II.5 – La comparaison des deux intervalles de confiance : courant et référence

II.4.5 Vue globale des étapes de détection

La Figure II.6 présente la succession des étapes de la détection des anomalies dans les délais d'un lien donné. Ce sont les étapes décrites dans la section II.4.4. Afin de mieux comprendre le processus décrit dans la Figure II.6, on note que :

- *periode* est une des périodes d_k entre *start* et *end*.
- *traceroute* est un objet JSON reprenant tous les détails d'une réponse à une requête traceroute effectuée par une sonde Atlas. La liste complète des attributs d'une réponse est disponible sur la documentation du RIPE Atlas ⁷.
- *medianByHopTraceroute* est un traceroute avec des sauts agrégés ; au sein d'un saut, regrouper les signaux d'un même routeur et calculer la médiane de leurs RTTs.
- *link* représente les deux adresses IP d'un lien donné ainsi que leur RTT différentiel, ainsi *link* : $\{ip1, ip2, rttDiff\}$.
- En plus du RTT différentiel et les adresses IP, *detailedLink* caractérise un lien en reprenant sa trace ; la sonde l'ayant identifié, la destination finale pour laquelle ce lien a été identifié, etc. *detailedLink* : $\{prb_id, msm_id, from, timestamp, linkIPs\}$, où *linkIPs* : $\{ip1, ip2\}$.
- L'état courant, à la période d_k , d'un lien est donné par la médiane des RTTs différentiels (*currentMedian*) enregistrés par ce lien durant d_k et son intervalle de confiance : la borne inférieure (*currentLow*) et le borne supérieure (*currentHight*).

7. Source : https://atlas.ripe.net/docs/data_struct/#v4750_traceroute, consultée le 05/01/2018.

- L'état référence d'un lien à la période d_k est obtenu par la médiane des RTTs différentiels (*referenceMedian*) enregistrés depuis d_1 jusqu'à la période courante d_k . Par contre, cette médiane est calculée en utilisant la moyenne mobile exponentielle. De plus, l'état référence d'un lien est caractérisé par l'intervalle de référence : une borne inférieure (*referenceLow*) et une borne supérieure (*referenceHight*)
- Les anomalies dans les délais du lien analysé, si elles existent, sont caractérisées par la période pendant laquelle elles étaient identifiées et la médiane des RTTs différentiels, de cette période, ayant donné cette anomalie.

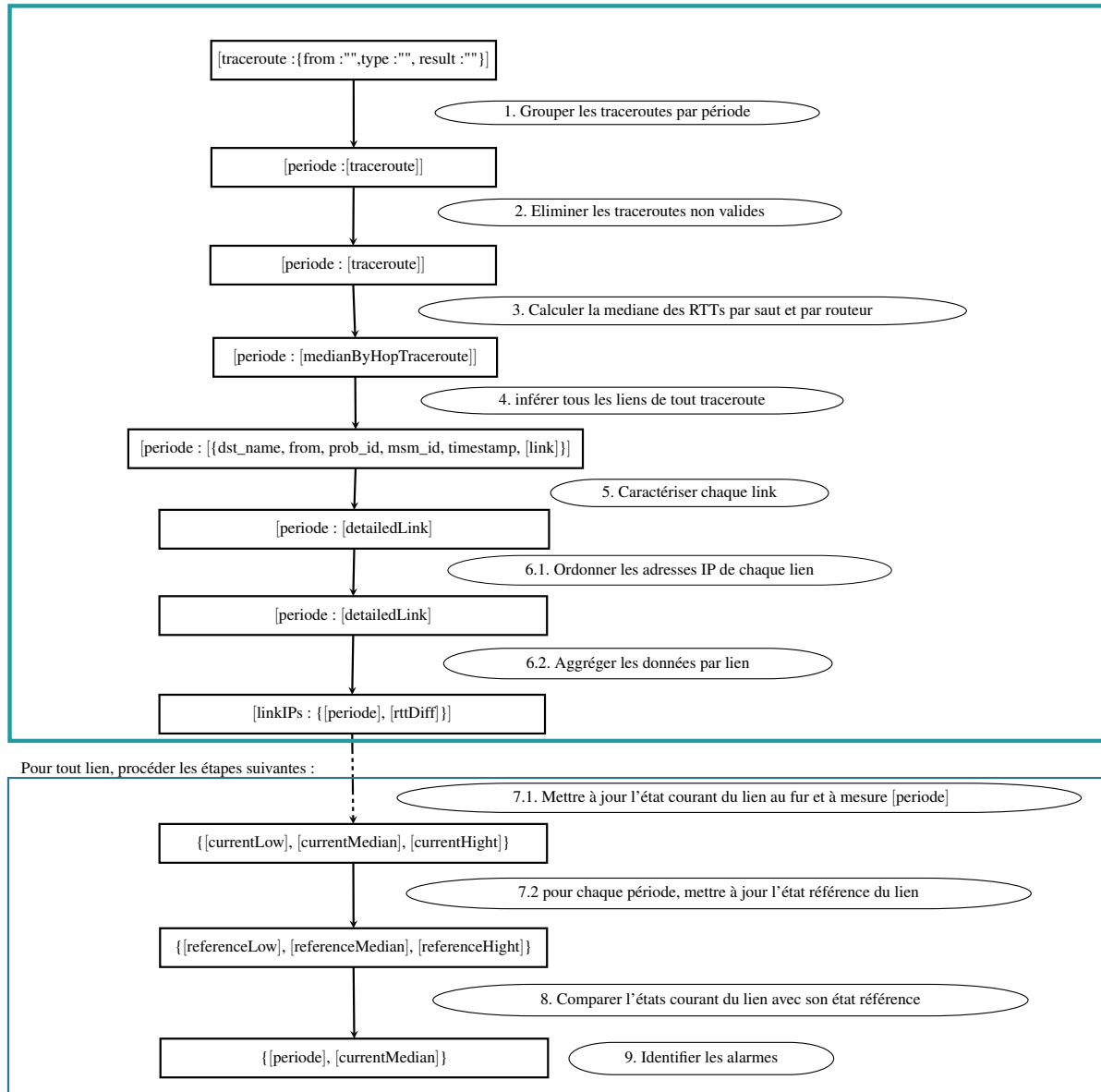


FIGURE II.6 – Le processus de la détection des anomalies dans les délais des liens

II.5 Conclusion

Nous avons abordé les différentes étapes de la création de l'évolution du RTT différentiel des liens identifiés dans les traceroutes analysés. Cette évolution présente aussi les anomalies

si sont présentes. C'est une évolution des RTTs différentiels d'un lien mais aussi un indicateur sur les délais des liens. Les approches adoptées dans la création de l'outil de détection suppose qu'on dispose d'assez de traceroutes pour assurer les performances des méthodes conçues. Etant donnée que les sondes Atlas sont nombreuses, la quantité de données générées est très grande, d'où la nécessité d'utiliser des technologies adaptées à la quantité importante de données. C'est l'objet du chapitre **IV**. Ce dernier reprend une introduction au Big Data ainsi qu'une liste non exhaustive des technologies Big Data.

Chapitre III

Important à savoir

III.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.

III.2 Rappel

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

III.3 Processus de détection des anomalies

III.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 du lo et hi , ainsi nous avons $(l, h) = (int(lo), int(hi))$ ¹.

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 :
- $$rttDiffL = dist[l] \text{ et } rttDiffH = dist[h], \text{ avec } l, h \in [0, n[.$$

- **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 vraie² :

Listing III.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.

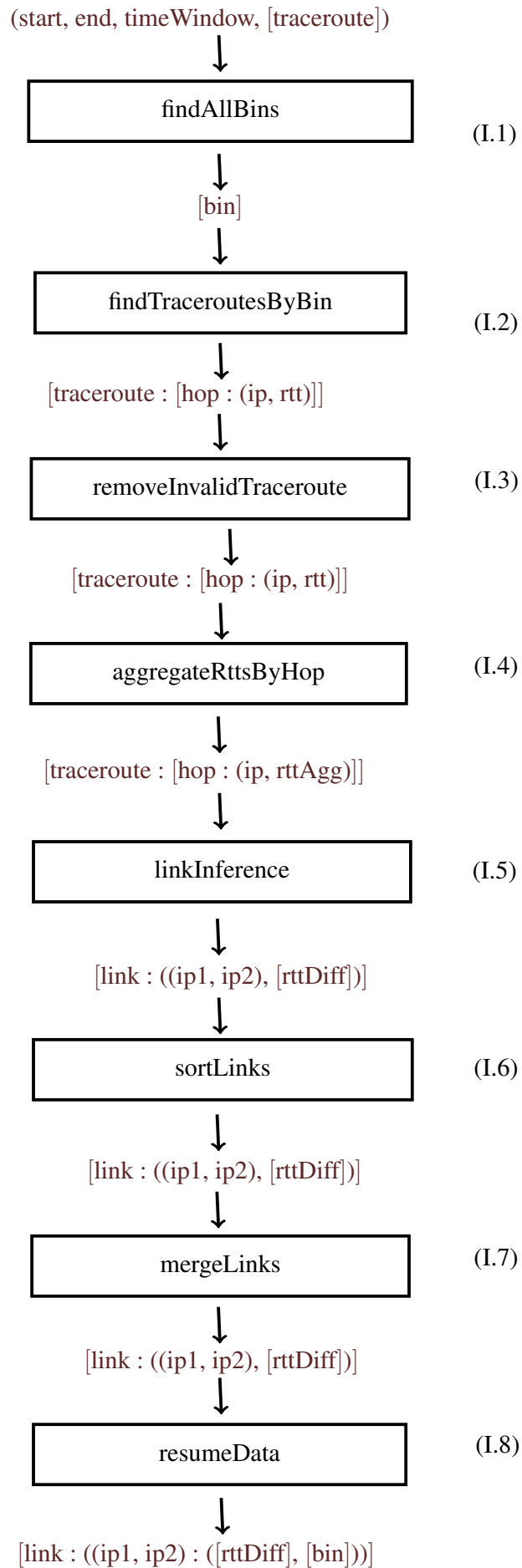


FIGURE III.1 – Phase I : préparation des données

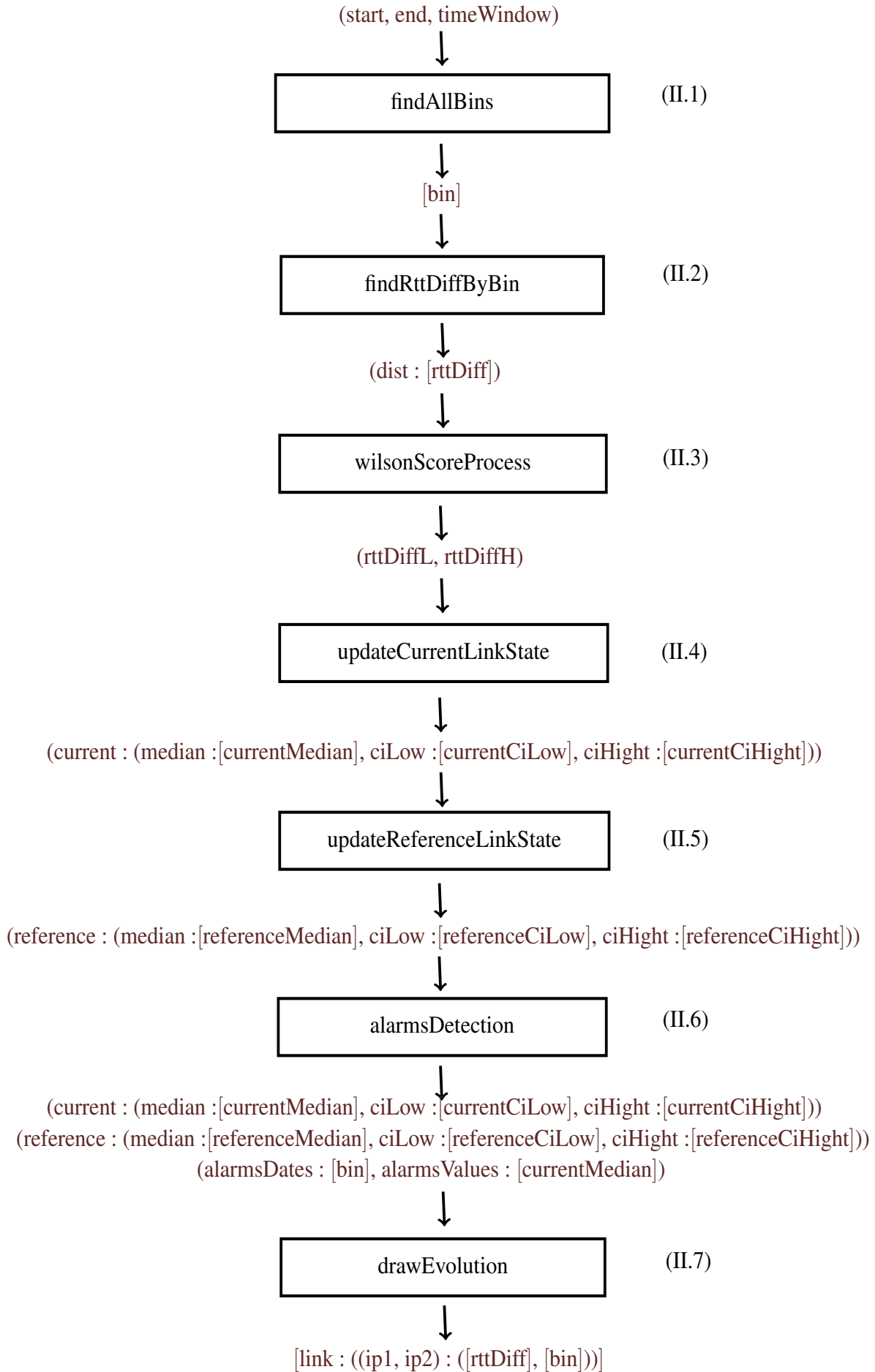


FIGURE III.2 – Phase II : détection des changements

III.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[!].

étape	outputs																						
(I.1)	1514769800, 1514773400, 1514777000, 1514780600, 1514784200, 1514787800																						
(I.2)	<table> <tr> <th>Id</th><th>Période</th><th>nombre de traceroutes</th></tr> <tr> <td>1</td><td>[1514769800, 1514769800+3600]</td><td>4</td></tr> <tr> <td>2</td><td>[1514773400, 1514773400+3600]</td><td>4</td></tr> <tr> <td>3</td><td>[1514777000, 1514777000+3600]</td><td>4</td></tr> <tr> <td>4</td><td>[1514780600, 1514780600+3600]</td><td>4</td></tr> <tr> <td>5</td><td>[1514784200, 1514784200+3600]</td><td>4</td></tr> <tr> <td>6</td><td>[1514787800, 1514787800+3600]</td><td>4</td></tr> </table>	Id	Période	nombre de traceroutes	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	
Id	Période	nombre de traceroutes																					
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 III.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 III.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 III.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 III.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 III.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 III.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, high)	(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, high)	(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	15.243713899999998
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]

III.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 IV.4.4. 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 [IV.4.4](#)). 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

Chapitre IV

Introduction au Big Data

IV.1 Introduction

Big Data est un terme associé aux données massives, rapidement générées, ayant une grande diversité où les outils traditionnels sont incapables de gérer ces données. La complexité du Big Data est dû 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.

IV.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 au stockage de données massives, au traitement de données massives et le besoin à 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 Center¹ 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, 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 des 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 aussi 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. Tel qu'un schéma comporte les différentes tables, les attributs pertinents et autres.

Modélisation A l'issue de cette phase, un modèle de qualité est créé. Un modèle qui doit répondre aux objectifs précédemment établis. 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 prend 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 qui convient aux besoins spécifiques en terme de visualisation des résultats de l'analyse.

IV.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.

IV.3.1 Définition du Big Data : Volume, Vitesse, Variété et Véracité

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

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 ([?], 2013) de l'International Data Corporation (IDC), le volume de données va atteindre 40 Zettaoctets² 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 = 1000000000000 GB

2.5 Pétaoctets de données chaque heure via les transactions de ses consommateurs³.

Variété de données Le Big Data inclut toutes les formes des 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, du réseau 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.

IV.3.2 L'architecture standard du Big Data

L'architecture standard du Big Data présentée dans ce qui suit est celle proposée dans [?]. 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 IV.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 envoyées vers les bases de données NoSQL (IV.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 IV.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

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

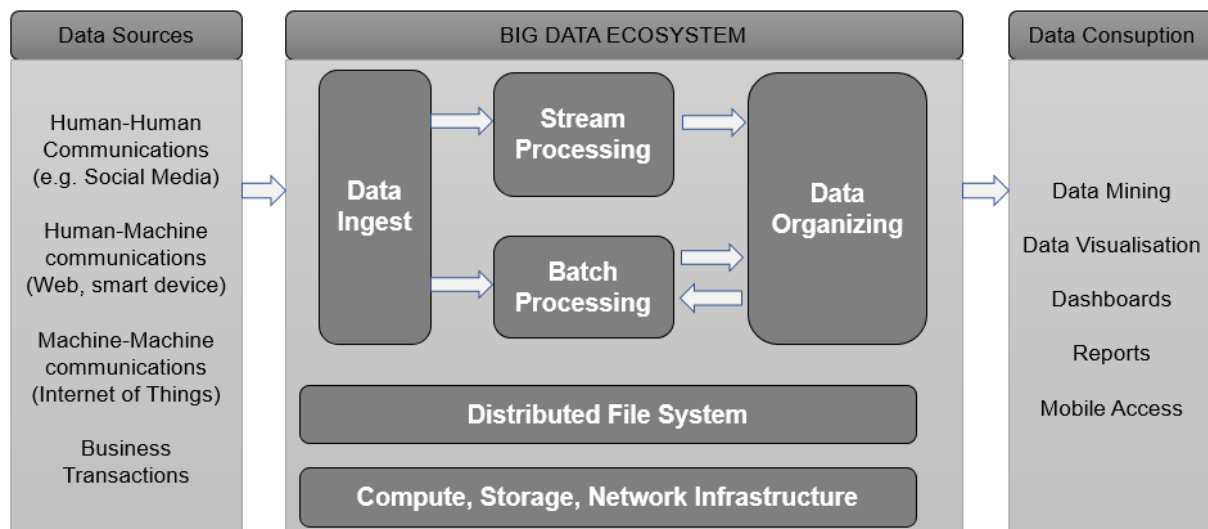


FIGURE IV.1 – Architecture standard du Big Data [?]

quantité aussi variable. Après avoir traversé la couche *Data Ingest*, les données sont envoyées au *batch processing system*, au *realtime 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, les types de données, les algorithmes qui vont travailler sur ces données et enfin les résultats souhaités. Les résultats des traitements peuvent être utilisés par une des applications ou bien sauvegarder ces données 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 que 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 afin de faciliter l'accès à ces dernières. Ce sont les données obtenues de la part de la couche *Stream Processing* ainsi que 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, les ressources du calcul et la gestion de la communication. Par exemple, les fonctionnalités de cette couche sont 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 toute les couches qui forment un système du Big Data. C'est ce que 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.

IV.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 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 des bases de données NoSQL, c'est d'abord assurer la capacité de stocker des données à grande échelle dont leur 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 le cas des 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. Notamment, 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 IV.2 reprend un exemple d'une base de données NoSQL de type clé-valeur.

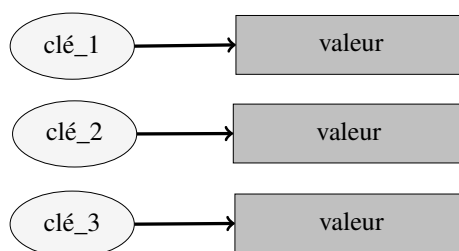


FIGURE IV.2 – Illustration d’une base de données NoSQL de type clé-valeur

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 la Figure IV.3.

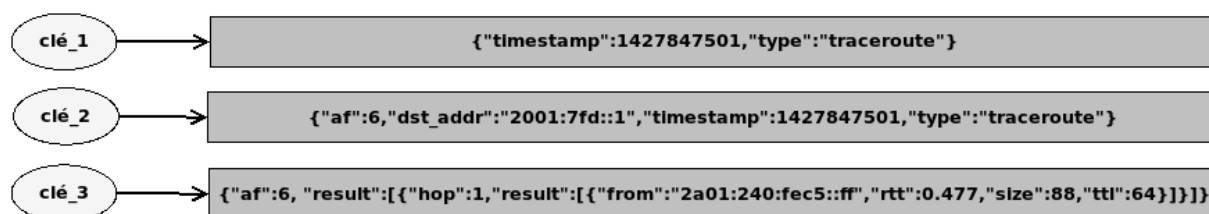


FIGURE IV.3 – Illustration d’une base de données NoSQL de type document

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, illustrée dans la Figure IV.4, est adaptée pour les requêtes analytiques comme les requêtes d’agrégation (moyennes, maximum, etc).

clé	prob_id	clé	af	clé	msm_id
1	233	1	4	1	5001
4	10	2	6	2	5006
		3	4	3	16345
		4	4	4	6026

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

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. Comme il est illustré dans la Figure IV.5.

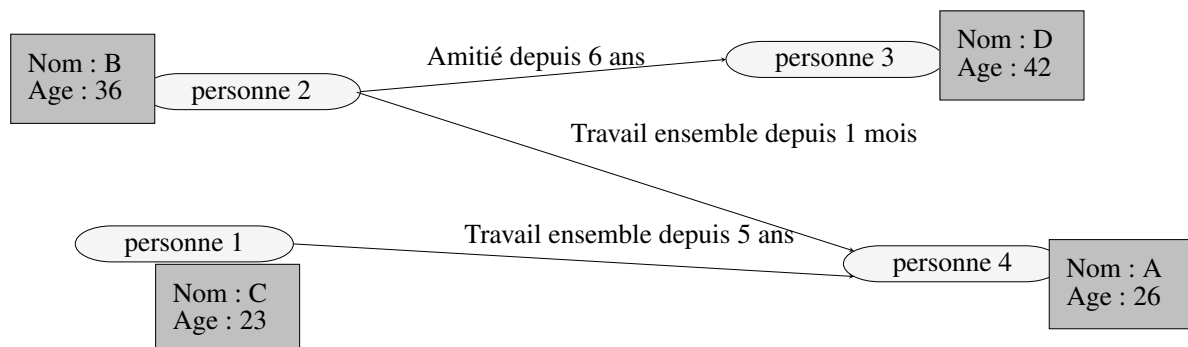


FIGURE IV.5 – Illustration d’une base de données NoSQL de type graphe

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.

Big Data et le théorème CAP :

Dans le but d’assurer un traitement rapide de données à grande échelle, ces dernières sont réparties 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 répllication, ces implémentations assurent la cohérence des données même en présence de mises à jour concurrentielles.

- 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 IV.6 présente des implémentations des différents types de bases de données NoSQL pour chaque couple CA, CP et AP.

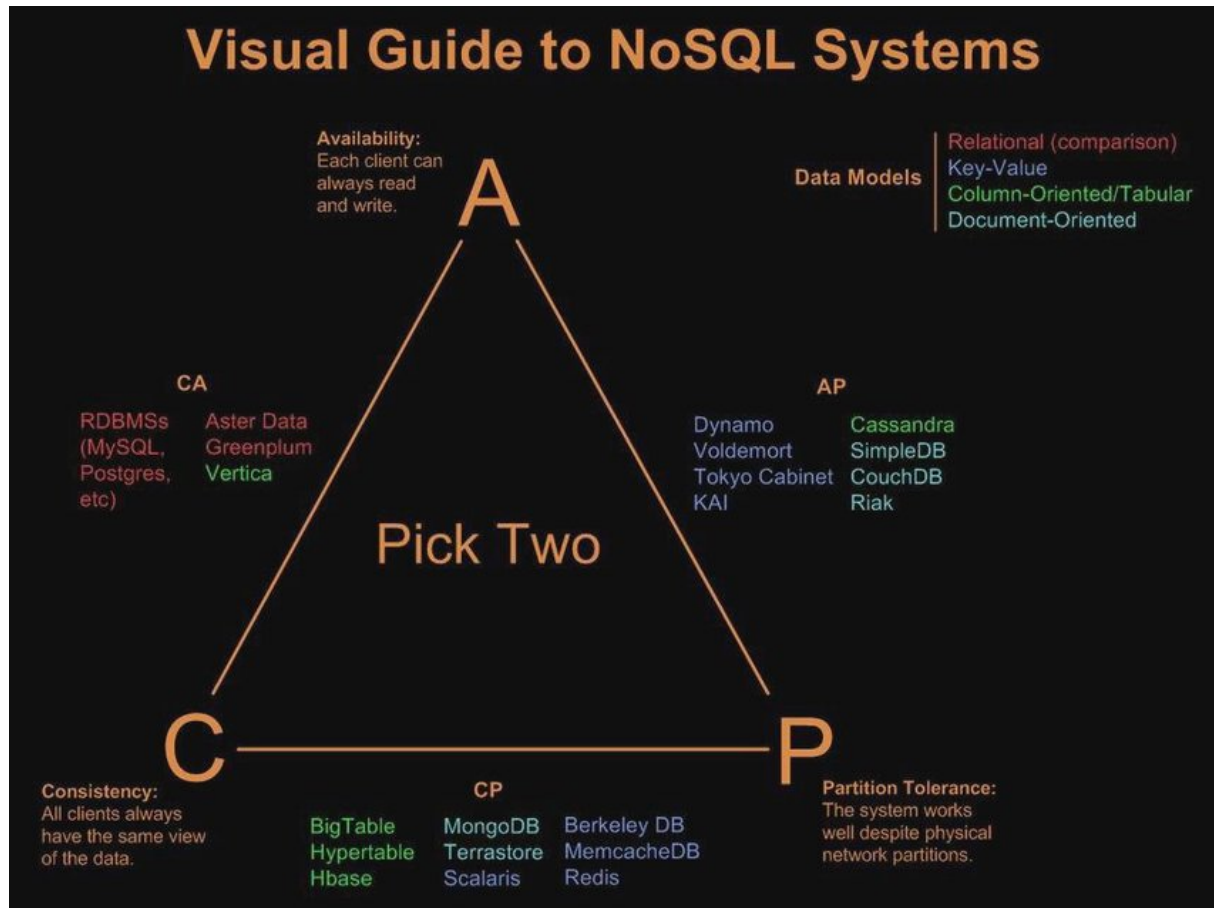


FIGURE IV.6 – Bases de données NoSQL suivant le théorème de CAP

Source: <https://payberah.github.io/files/download/p2p/nosql.pdf>, consultée le 05/08/2018.

Le choix d'une base de données relationnelle ou NoSQL dépend des besoins des entreprises. En terme de tendances, la Figure IV.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*⁴. Parmi les critères du classement, on trouve le nombre de références du SGBD sur les sites Internet.

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

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 IV.7 – Un classement des SGBDs sur *DB-Engines Ranking* du 1 août 2018

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

IV.3.4 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, mais le type de la donnée qui diffère. C'est l'approche utilisée par Amazon Athena (voir IV.4.3). Les Figures IV.8 et IV.9 illustrent la différence entre ces deux approches.

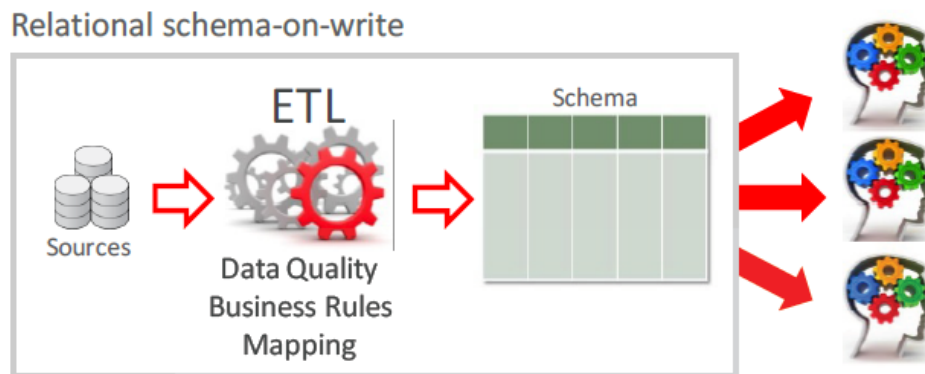


FIGURE IV.8 – Schema on Write (SGBDR)

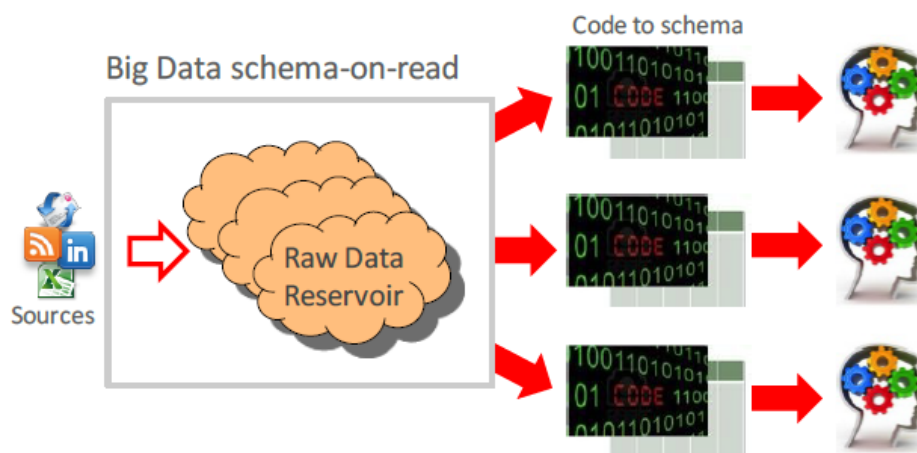


FIGURE IV.9 – Schema on Read (Big Data)

Source: <https://blogs.oracle.com/datawarehousing/big-data-sql-quick-start-schema-on-read-and-schema-on-write-part11>, consultée le 05/08/2018.

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.

IV.3.5 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. D'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^a.

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

IV.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.

IV.4.1 MongoDB

MongoDB⁵ est une base de données NoSQL de type Document⁶. MongoDB est classé parmi les SGBDs adoptant le couple CP (Consistency et Partition Tolerance) dans le théorème CAP⁷. 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 Atlas⁸. 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 proposées⁹, y inclut l'offre gratuite pour expérimenter MongoDB Atlas. Les frais d'utilisation du service MongoDB Atlas dépend du stockage, de la RAM allouées 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^a.

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

IV.4.2 Amazon DynamoDB

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

6. Une base de données NoSQL de type document est décrite dans la section IV.3.3.

7. Le théorème CAP est décrit dans la section IV.3.3

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

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

Amazon DynamoDB¹⁰ 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 de façon abstraite pour l'utilisateur final. Elle 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 éléments¹¹ :

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

IV.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 le stockage de données, le chargement de données et l'interrogation de données.

Amazon S3¹² est un service de stockage d'objets dans le cloud. Il est conçu pour stocker et récupérer toute quantité de données. Il peut assurer 99,999999999 % de durabilité. La sécurité et l'accès aux données sont assurés. Il existe plusieurs classes de stockage qui répondent aux différents besoins.

Les fichiers des données sont organisés dans ce qu'on appelle un compartiment, c'est une simulation de dossier dans un système d'exploitation. A l'intérieur d'un compartiment, il est possible de créer des compartiments imbriqués. C'est une simulation d'arborescence de dossiers car physiquement cet arborescence n'existe pas. En ce qui concerne les frais du service AWS S3, le Tableau IV.1 décrit les tarifs de la formule standard.

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

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

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

Amazon Glue¹³ 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 rendre accessibles dans un

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

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

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

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

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, ils 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 DPU¹⁴, il est facturé à la seconde avec un minimum de 10 minutes par robot d'analyse exécuté. Plus de détails sont disponibles sur Amazon Glue¹⁵.

Amazon Athena¹⁶ 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 IV.3.4) 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. C'est un mécanisme présent beaucoup sur le cloud.

L'exécution des requêtes SQL est effectuée par le moteur de requêtes SQL Presto. pour les instruction DDL, elles sont effectuées par *Hive Data Definition Language*¹⁷. 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.

14. DPU : unité de traitement des données.

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

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

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

Presto^a 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. Ce qui 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, un tableau d'éléments, etc. Il supporte aussi des opérations complexes sur les données.

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

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

IV.4.4 Apache Spark

Apache Spark¹⁸ 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 existe 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 YARN¹⁹ ou Apache Mesos²⁰. 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 IV.10 montre l'interaction entre une application spark, le cluster manager et les *workers*.

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

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

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

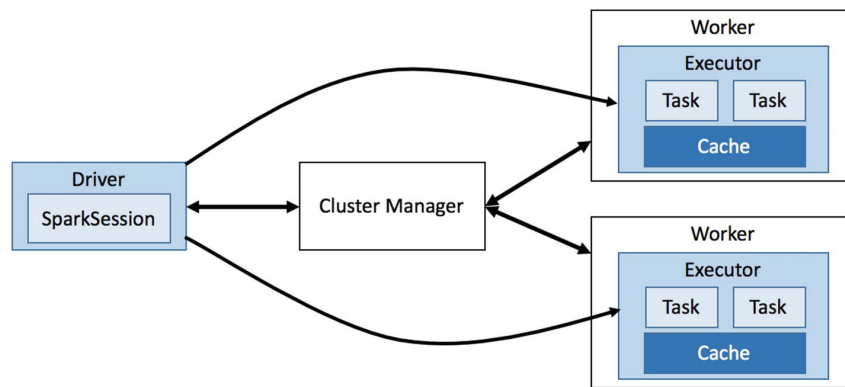


FIGURE IV.10 – Interaction entre une application Spark et le cluster manager. Source : [?]

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 APIs²¹ 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 IV.10, 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 de la totalité des données à analyser. De cette manière, Spark est capable de traiter les données de façon parallèle. La Figure IV.11 illustre un exemple d'un cluster. Ce dernier est composé de trois *executors*.

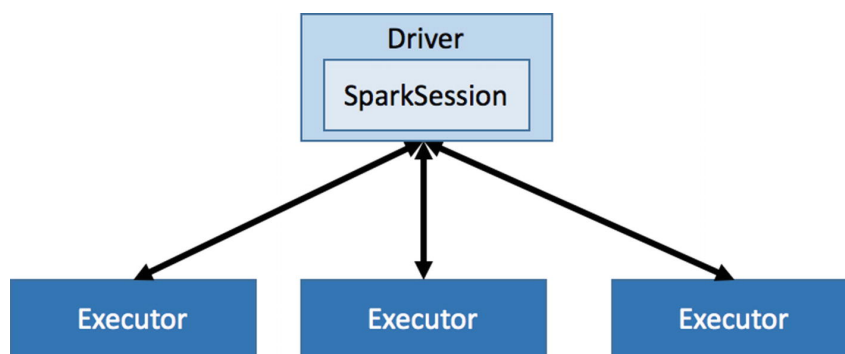


FIGURE IV.11 – Un exemple d'un cluster formé de trois executors. Source : [?]

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

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

- Spark SQL est conçu pour le traitement interactive ;
- Spark Streaming est utilisé pour les traitements en temps réel ;
- GraphX est destiné au traitement de graphe ;
- MLib est conçu pour 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 distribué du calcul. Cette dernière est responsable de la distribution, 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 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 IV.12 présente les différentes entités du Spark Unified Stack avec Spark Core.

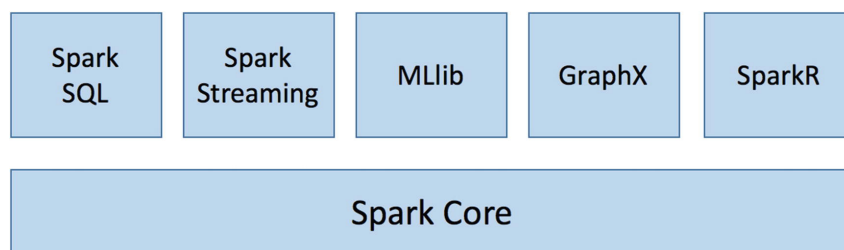


FIGURE IV.12 – Spark Unified Stack. Source : [?]

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 persisté dans la mémoire pour éventuelle réutilisation. D'ailleurs de cette façon, on constate l'amélioration des performances. En outre, un RDD peut être persisté sur 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. Toutefois, une action appliquée sur un RDD retourne une seule valeur, par exemple l'action *count* calcule le nombre d'objets d'un RDD. La Figure IV.13 illustre un flux de données avec l'utilisation de Spark.

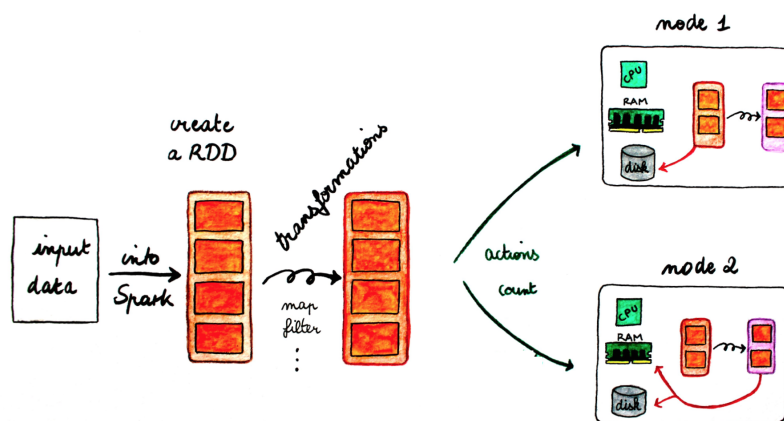


FIGURE IV.13 – Exemple d'un flux de données avec Spark

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

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 sont partagées sur les nœuds du cluster qu'à suite à l'appel d'une action. Une fois une partition est localisé 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.

Les modes d'exécution du framework Spark [!]

Standalone

Les APIs du framework Spark Il existe de nombreuses API permettant d'utiliser les fonctionnalités du Spark. A savoir, il y a Java, Python, Scala et R. Nous avons utilisé Scala dans l'implémentation exposée dans le chapitre V.

A propos lazy evaluation

IV.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 VI l'utilisation de ces technologies dans le cas de l'analyse des délais d'un lien décrite dans la chapitre II.

Chapitre V

Implémentation de la détection des anomalies dans les délais en Spark/Scala

V.1 Introduction

Dans le présent chapitre, nous décrivons l'implémentation de l'outil de détection en utilisant le framework Spark, décrit dans la section IV.4.4, en Scala. L'outil implémenté est décrit dans le chapitre II. L'implémentation proposée implique plusieurs éléments relatifs au langage Scala comme les case class, les fonctions et le RDD relatif au Spark.

V.2 Implémentation

V.2.1 Description de l'environnement

Dans l'implémentation proposée, nous avons utilisé le mode local, intitulé *Standalone*. Le code source qui traduit l'ensemble de 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*¹.

V.2.2 Eléments de l'implémentation

Un programme Spark implique un ensemble d'éléments. La première chose à faire est de configurer le programme Spark et ce à travers un objet *SparkConf*. Ce dernier contient les informations sur l'application. Ensuite, nous utilisons la composante Spark SQL du Spark Unifified Stack (voir la section IV.4.4) pour lire les traceroutes présents dans des fichiers d'entrée. Cela permet de créer un Dataset d'objet Traceroute. Nous convertissons ensuite un Dataset à un RDD afin d'appliquer différentes transformations aboutissant à l'identification des anomalies dans les délais des liens.

Configuration d'une application Spark

Une application Spark nécessite l'ajustement de quelques paramètres, qu'il s'agit d'une application qui tourne en mode local ou bien en mode cluster. On peut ajuster ces paramètres selon trois possibilités. La première possibilité est à travers l'objet *SparkConf* comme illustré dans l'exemple dans le Listing V.1 où nous donnons un nom à l'application Spark (ligne 5) et nous précisons l'URL du cluster (ligne 6).

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

Listing V.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 du traitement, dont un exemple est illustré dans le Listing V.19. Enfin, quelques paramètres peuvent être lu depuis un fichier de configuration disponible sur *conf/spark-defaults.conf*².

Point d'entrée vers les fonctionnalités du Spark

Le point d'entrée vers les fonctionnalités du 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.

Les 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, la date du début de l'analyse (1517961600), la date de fin (1518134400) et enfin la durée de la période (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.

Notation relatives au langage Scala

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

<i>case class</i>	: créer une case classe.
<i>Seq</i>	: créer une séquence, équivalent à une liste. Par exemple <i>Seq[String]</i> représente le type d'une liste dont ses éléments sont de type <i>String</i> .
<i>Dataset</i>	: est une collection distribuée de données.
<i>map</i>	: permet de transformer le contenu d'une liste en appelant une fonction sur chaque élément de la liste, elle renvoi une liste transformée.
:	
:	
:	

La lecture des données

L'outil de détection proposé par R. Fontugne et al. n'exploite qu'une partie des données d'une réponse traceroute, un exemple d'une réponse traceroute est donné dans l'annexe B. En

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

particulier, on peut utiliser Spark afin de ne lire que les données qui nous intéressent, c’est le principe du *Schema-On-Read* décrit dans la section [IV.3.4](#).

Chaque réponse traceroute est structurée dans un objet JSON et par 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 [V.2](#).

Listing V.2 – Description du case class *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 [V.3](#)). Un saut est défini par son rang (*hop*), ce dernier indique l’ordre du saut en question. Etant donné que la sonde reçoit trois³ signaux de chaque saut, un saut est donc défini par un ensemble de signaux (*Seq[Signal]*).

Listing V.3 – Description du case class *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. Un signal est modélisé par la classe *Signal* (voir le Listing [V.4](#)).

Listing V.4 – Description du case class *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 V.5 – Le mapping entre données et cas class

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

3. Dans certains cas, le nombre de signaux dépasse trois.

Nous obtenons la liste des traceroutes dans la variable *rawTraceroutes*. Ce dernier est un Dataset d'objets Traceroute. A la ligne 5 du Listing V.5, nous appelons certaines fonctionnalités nécessaires à la lecture des données. Il est important de noter que Apache Spark adopte ce qu'on appelle *lazy evaluation* (voir la section IV.4.4). Ainsi, l'évaluation des différentes transformations ne s'effectuent qu'au moment du déclenchement d'une action sur le résultat de cette transformation.

Trouver les périodes de l'analyse

Dès présent, la liste des traceroutes est prête à toute transformation de la phase I. Tout d'abord, nous devons trouver les périodes entre la date de début et la date de fin (étape *FindBins* (I.1)). Ensuite, nous cherchons les traceroutes capturés durant ces périodes. Dans l'implémentation du travail de référence les données sont organisées dans des collections MongoDB (voir la section IV.4.1), le groupement des traceroutes par période se base sur la structuration des noms des collections⁴. Pour une période donnée, seulement les collections concernées qui seront interrogées. Cette étape est illustrée par la fonction *generateDateSample* dans le Listing V.6, ensuite 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*) et *start+timewindow*, *timewindow* est équivalent à une heure.

Listing V.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).

Le groupement des traceroutes

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

Algorithm 1

```

for period ∈ rangeDatesTimewindows do
  collection ← Trouver la collection incluant period
  rawTraceroutes ← les traceroutes stockés dans collection et
    enregistrés durant period
  ...
end for

```

▷ Traitements appliqués sur l'ensemble de traceroutes

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

4. Voir la section VI.4.

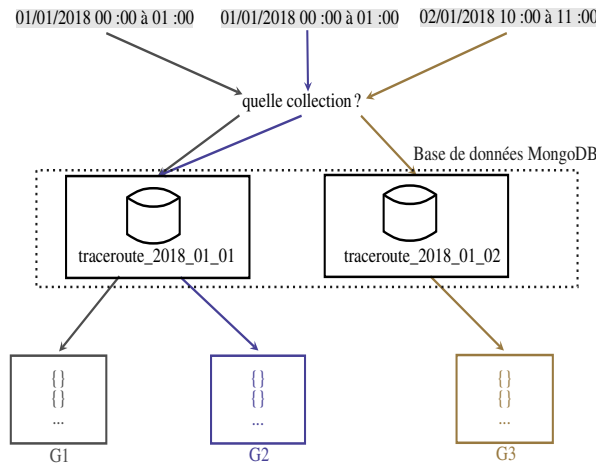


FIGURE V.1 – Groupement des traceroutes avec MongoDB

Spark/Scala En ce qui concerne l’implémentation en Spark/Scala, nous avons groupé les traceroutes autrement. Les traceroutes sont organisés dans des fichiers de données qui peuvent être relatifs à une heure, une journée ou toute autre période. 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.

Algorithm 2

```

traceroutePerPeriod ← []
for traceroute ∈ rawTraceroutes do
  for period ∈ rangeDatesTimewindows do
    Vérifier si traceroute appartient à period
  end for
  traceroutePerPeriod.append( (traceroute, period) )
end for
traceroutesPerPeriods ← traceroutePerPeriods.groupBy(period)
for element ∈ traceroutesPerPeriods do
  ...
  ▷ Traitements appliqués sur l’ensemble de traceroutes
end for

```

En pratique, nous avons chargé les traceroutes disponibles à l’analyse sur un RDD. Ce dernier crée des partitions de données, sensé 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 V.2.2 :

– 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 *TracerouteWithTimewindow*.

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

Listing V.7 – La classe *TracerouteWithTimewindow*

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

Élimination des traceroutes invalides

Il se peut qu'un traceroute ne fait pas partie de la période de l'analyse. Ce sont les objets de type *TracerouteWithTimewindow* ayant *period* de valeur 0. C'est pourquoi nous filtrons ces traceroutes, comme il illustre le Listing V.8.

Listing V.8 – La classe *TracerouteWithTimewindow*

```
1 val onlyConcernedTraceoutes = 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 V.10. L'objectif de cette agrégation est de créer des groupes de traceroutes et y appliquer les traitements relatifs à la détection. Le Listing V.9 illustre l'étape de l'agrégation.

Listing V.9 – La classe *TracerouteWithTimewindow*

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

Listing V.10 – La classe *TracerouteWithTimewindow*

```
1 case class TraceroutesPerPeriod (
2     traceroutes : Seq [ Traceroute ] ,
3     timeWindow : 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 V.11 :

Listing V.11 – La classe *TracerouteWithTimewindow*

```
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 V.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. résumer 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 V.12 – Deduction des liens par groupe de traceroutes

```

1  def linksInference(spark: SparkSession, rawtraceroutes :
    TraceroutesPerPeriod): Seq[ResumeLink] = {
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 =>
        removeNegative(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 =>
        findLinksByTraceroute(spark, t))
15
16     // Create a set of DetailedLink objects for every traceroute
17     val detailedLinks = tracerouteLinks.map(resumeLinksTraceroute)
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     // Resume the link
29     val resumeData = mergedLinks.map(f => ResumeLink(f._1, f._2.map(_
        .probe), f._2.map(_.rttDiff), generateDatesSample(f._2.size,
        rawtraceroutes.timeWindow)))
30
31     resumeData.toSeq
32 }

```

Et la définition de la classe *ResumedLink* est donnée dans le Listing V.13. Cette définition 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 *bins* qui représentent les périodes pendant lesquelles les *rttDiffs* ont été identifiés.

Listing V.13 – Définition de la classe ResumedLink

```
1 case class ResumedLink(
2     link:      LinkIPs ,
3     probes:    Seq[ BigInt ],
4     rttDiffs:  Seq[ Double ],
5     var bins:  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 machine, il s'agit de la collecte de ces résultats de chaque machine (datanode). La collecte des résultats est illustré par le Listing V.14.

Listing V.14 – Deduction des liens par groupe de traceroutes

```
1 val collectedRTTDiff = 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 V.15.

Listing V.15 – Deduction des liens par groupe de traceroutes

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

A cette étape, nous avons une liste de type *ResumedLink*.

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 V.16, 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.

Listing V.16 – Définition de la classe ResumedLink

```
1 val rawDataLinkFiltred = spark.sparkContext.parallelize(finalRawRttDiff.
   toSeq)
2 .map(p => listAlarms(spark, p, timewindow, rangeDates))
```

La méthode *listAlarms* est détaillée dans le Listing V.17. Dans cette dernière :

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 V.17 – Définition de la méthode listAlarms

```

1  def listAlarms(spark: SparkSession, rawDataLinkFiltred: ResumedLink,
2      timewindow: Int, rangeDates: Seq[Int]): String = {
3      // Save the reference state of a link
4      var reference = LinkState(Seq(), Seq(), Seq(), Seq())
5
6      // Save the current state of a link
7      var current = LinkState(Seq(), Seq(), Seq(), Seq())
8
9      // Save the RTT differentials anomalies
10     var alarmsValues = AlarmsValues()
11
12     // Save the dates having delay anomalies
13     var alarmsDates = AlarmsDates()
14
15     // Save all the dates to draw the evolution
16     var dates = AllDates()
17
18     val rawDataLink = rawDataLinkFiltred
19
20     /* Regardless of the period specified in the inputs, the evolution
21        is created for one or more days
22     * Eg : if the period is only 2 hours, the evolution is created for
23        24 hours,
24     * and the begin date is the begin date given in inputs
25     */
26     val start = rawDataLink.bins.min
27     val max = rawDataLink.bins.max
28     val differenceDays = (max - start) / 60 / 60 / 24
29     val end = start + ((differenceDays + 1) * 86400)
30
31     // Find all the bins in the selected days
32     val datesEvolution = start.to(end - timewindow).by(timewindow)
33
34     // For each bin, find the data (RTTs differentials) and find alarms
35     datesEvolution.foreach(f => findAlarms(spark, f, reference,
36         rawDataLink, current, alarmsDates, alarmsValues, dates))
37
38     // create a JSON string to save the results
39     implicit val formats = DefaultFormats
40     val linkEvolution = LinkEvolution(rawDataLink.link, reference,
41         current, alarmsDates.dates, alarmsValues.medians, dates.dates)
42     val linkEvolutionJsonStr = write(linkEvolution)
43     linkEvolutionJsonStr
44 }

```

La définition de la fonction *findAlarms* est donnée dans le Listing V.18. 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 détaillées ci-dessus.[!]

Listing V.18 – 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)))
39
40         reference.valueHi = newReferenceValueHi
41         reference.valueLow = newReferenceValueLow
42         reference.valueMedian = newReferenceValueMedian
43
44       } //Case : 2
45       else if (reference.valueMedian.size == 3) {
46         val newReferenceValueMedian1 = tmpReference.
47           valueMedian :+ medianCalculator(tmpReference.
48             valueMedian)
49         val newReferenceValueHi1 = tmpReference.valueHi :+
50           medianCalculator(tmpReference.valueHi)
51         val newReferenceValueLow1 = tmpReference.valueLow
52           :+ medianCalculator(tmpReference.valueLow)
53
54         reference.valueHi = newReferenceValueHi1
55         reference.valueLow = newReferenceValueLow1
56         reference.valueMedian = newReferenceValueMedian1
57       }
58     }
59   }

```

```

45
46         val newReferenceValueMedian = reference.valueMedian
47             .map(f => reference.valueMedian.last)
48         reference.valueMedian = newReferenceValueMedian
49         val newReferenceValueHi = reference.valueHi.map(f
50             => reference.valueHi.last)
51         reference.valueHi = newReferenceValueHi
52         val newReferenceValueLow = reference.valueLow.map(f
53             => reference.valueLow.last)
54         reference.valueLow = newReferenceValueLow
55     } // Case : 3
56     else {
57
58         val newReferenceValueMedian2 = tmpReference.
59             valueMedian :+ (0.99 * tmpReference.valueMedian.
60                 last + 0.01 * current.valueMedian.last)
61         val newReferenceValueHi2 = tmpReference.valueHi :+
62             (0.99 * tmpReference.valueHi.last + 0.01 *
63                 newDist(javatools.JavaTools.getIntegerPart(
64                     wilsonCi(1))))
65         val newReferenceValueLow2 = tmpReference.valueLow
66             :+ (0.99 * tmpReference.valueLow.last + 0.01 *
67                 newDist(javatools.JavaTools.getIntegerPart(
68                     wilsonCi(0))))
69         reference.valueHi = newReferenceValueHi2
70         reference.valueLow = newReferenceValueLow2
71         reference.valueMedian = newReferenceValueMedian2
72
73         // Anomalies dection : compare the current with the
74         // reference
75         if ((BigDecimal(current.valueMedian.last) -
76             BigDecimal(current.valueLow.last) > reference.
77                 valueHi.last - current.valueMedian.last + current
78                 .valueHi.last < reference.valueLow.last) &&
79             scala.math.abs(current.valueMedian.last -
80                 reference.valueMedian.last) > 1) {
81
82             val updateAlarmsDates = alarmsDates.dates
83                 :+ date
84             alarmsDates.dates = updateAlarmsDates
85
86             val updateAlarmsValues = alarmsValues.
87                 medians :+ current.valueMedian.last
88             alarmsValues.medians = updateAlarmsValues
89         }
90     }
91 }

```

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 expliqués dans [!];
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.

V.2.3 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 V.19.

Listing V.19 – Exemple de la soumissions d'un traitement sur Spark

```
1 ~$ bin/spark-submit --class repeatlasanalysis.AnalyseTraceroute
  --master local --driver-memory 30G --conf "spark.network.timeout
    =10000000" SparkExample-lowprints-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. Par exemple la valeur *local* indique que Spark est exécuté localement avec un seul worker thread (aucun parallélisme), alors que *local[K]* lance le traitement sur Spark en utilisant *K* worker sur *K* threads, idéalement *K* est le nombre des cœurs de la machine ;
- *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.

V.3 Conclusion

Chapitre VI

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

VI.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 **IV**. Les technologies que nous présentons couvrent les besoins d'une ou de plusieurs étapes d'un processus d'analyse de données.

VI.2 Critères d'évaluation des technologies Big Data

Les critères d'évaluation d'une technologie Big Data par rapport à une autre varient suivant plusieurs entrées. En générale, la liste des critères que l'on peut considérer dans la comparaison des technologies Big Data entre elles est très longue. En ce qui concerne les critères sur lesquels nous évaluons les différentes technologies Big Data expérimentées sont les suivants :

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

En pratique, nous n'avons pas pris en compte d'autres critères dans notre cas, 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 sur les frais d'utilisation de deux technologies dédiées au stockage de données massives : Amazon S3 (voir le Tableau **IV.1**) et MongoDB Atlas dont les frais d'utilisation dépendent de plusieurs paramètres ¹.

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.

VI.3 Caractéristiques de l'environnement de test

Les différents tests ont été réalisés sur un conteneur de type OpenVZ ayant les caractéristiques suivantes : système Debian GNU/Linux 7.11 (wheezy), 32768 MB de RAM, CPU modèle 6, CPU MHz 2294.331.

Il existe différentes catégories de virtualisation, **OpenVZ** s'inscrit dans la catégorie Isolateur. Un isolateur est un logiciel permettant d'isoler l'exécution des applications dans des contextes ou zones d'exécution. Un conteneur OpenVZ s'agit d'un partitionnement logique au niveau des ressources systèmes : processus, réseau et système de fichier^a.

^a. Source : http://cesar.resinfo.org/IMG/pdf/jtsiars-openvz_1_.pdf, consultée le 29/12/2018.

En ce qui concerne les paramètres de détection, nous avons utilisé les valeurs par défaut : *timeWindow* est de 3600 secondes, l'intervalle de confiance est de 0,05, *alpha* est de 0,01 et *minSeen* est de 3. Pour la date de début et la date de fin sont variées suivant les données utilisées.

Les différents tests effectués, présentés dans la suite de ce document, ont été effectués au sein de la machine dont les caractéristiques sont décrites ci-dessous. De plus, un seul test est lancé à un moment donné.

VI.4 Application 1 : MongoDB

MongoDB est la technologie Big Data utilisé par Fontugne dans l'implémentation de l'outil de détection [?]. 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*. Dans le cas de l'adressage IPv4, *V* ne prend aucune valeur, or, *V* prend la valeur 6 s'il s'agit de l'adressage IPv6. La nomination structurée 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 de données récupérées dépend 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 le contenu d'un traceroute est différent de celui réussi.

Généralement l'analyse de données à grande échelle se limite qu'au mode lecture de données pour en tirer les connaissances. MongoDB est adapté non seulement aux projets visant la lecture de données massives mais aussi aux projets où on envisage la mise à jour d'un objet dans une collection (modification ou suppression).

En cas de la mise à jour de la structure de nouveaux objets traceroutes par RIPE Atlas, cela n'affecte pas les données précédemment stockées dans MongoDB.

Performances de la base de données MongoDB dans l'analyse des délais

Nous évaluons les temps d'exécution lors de l'analyse des délais des liens présents dans un ensemble de traceroutes, en utilisant MongoDB comme technologie de stockage de données massives.

Dans le Tableau VI.1, nous varions l'ensemble de traceroutes en terme de stockage pour mesurer le temps nécessaire pour avoir l'évolution de tous les liens présents dans les traceroutes analysés, nous ne présentons que les premier trois essais.

Période	Taille (GB)	Essai 1 (secondes)	Essai 2 (secondes)	Essai 3 (secondes)
07/02/18	1	752,03	750	755,24
07/02/2018~08/02/2018	2	1499,28	1504,98	1503,62
07/02/2018~09/02/2018	3	2275,89	2265,96	2284,98
07/02/2018~10/02/2018	4	3035,19	3043,21	3057,45
07/02/2018~11/02/2018	5	3871	3889,57	3894,5

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

Tandis que la Figure VI.1 reprend la médiane des temps d'exécution de la détection en fonction de la taille de données analysées en utilisant MongoDB. La Figure VI.2 illustre la variation de la distribution des temps d'exécution. La Figure VI.2 ainsi que la Figure VI.1 ont été obtenues en considérant les temps d'exécution de 10 essais pour chaque taille de données.

Temps d'exécution de la détection et la taille de données analysées avec MongoDB

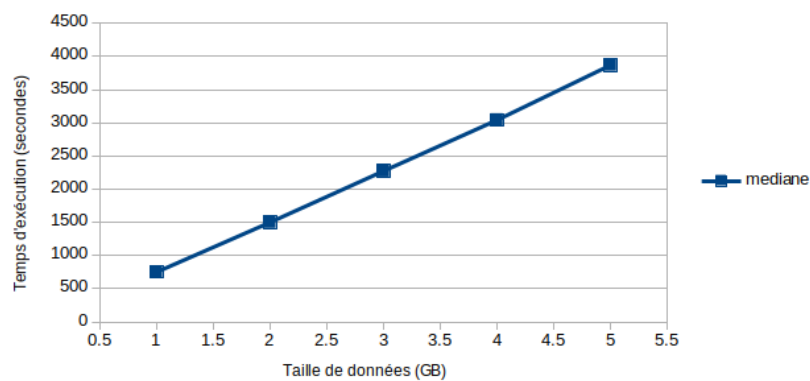


FIGURE VI.1 – La médiane des temps d'exécution en fonction de la taille de traceroutes analysés (MongoDB)

Les temps d'exécution en fonction de la taille de l'ensemble de données (MongoDB)

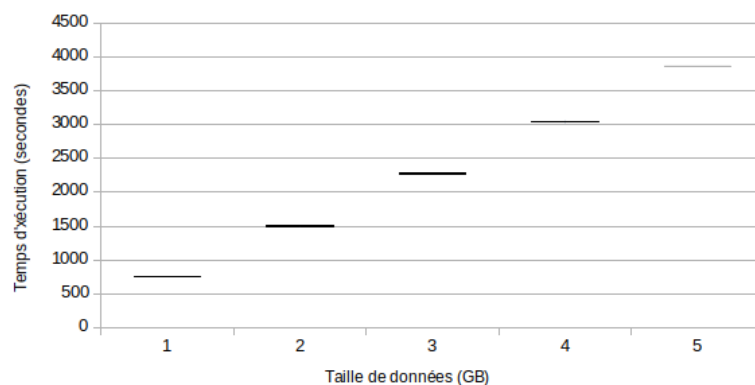


FIGURE VI.2 – Les temps d'exécution en fonction de la taille de traceroutes analysés (MongoDB)

VI.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 DynamoDB 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 nécessite l'ajustement des ressources de la machine qui reçoive 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 fait une taille moyenne de 620 MB en format compressé, ce que représente une quantité d'environ 9 GB en format texte.

VI.6 Application 3 : Amazon S3, Amazon Glue et Amazon Athena

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. Une vue globale du processus de l'analyse est illustré dans la Figure VI.3².

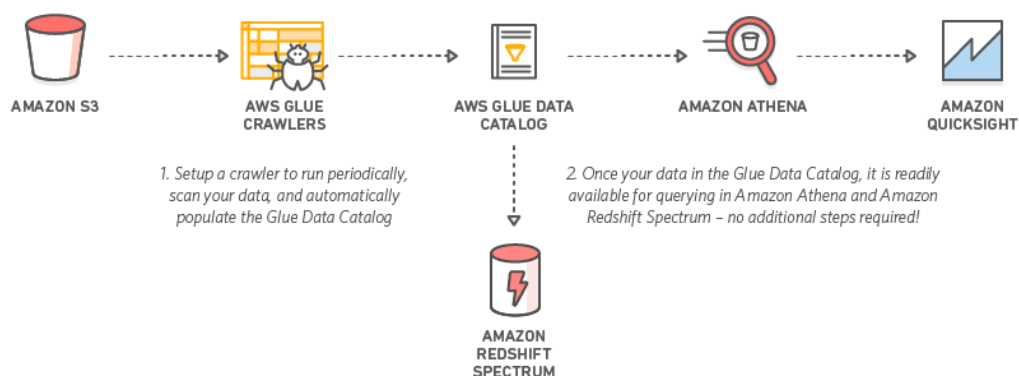


FIGURE VI.3 – Une combinaison des services web d'Amazon : Amazon S3, Amazon Glue, Amazon Athena, Amazon QuickSight et Amazon Redshift

Source: https://docs.aws.amazon.com/fr_fr/athena/latest/ug/glue-best-practices.html, consultée le 16/12/2018.

Afin d'utiliser Amazon Athena, nous avons besoin du schéma de données. Il s'agit d'une

2. Amazon Redshift est un entrepôt de données et Amazon QuickSight est un service cloud d'informatique décisionnelle.

table comme les tables dans un SGBDR. Pour ce faire, nous avons lancé avec Amazon Glue la détection automatique du schéma 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 du firmware de la sonde ayant effectué le traceroute. Pour finir, le schéma des traceroutes a été créé manuellement (voir la section A.1 dans l'annexe A). La table a été créée en utilisant le partitionnement des données dans un compartiment S3. Plus de détails sur le partitionnement sont données dans la section A.2 dans l'annexe A.

Une fois les fichiers de données sont synchronisés vers le compartiment AWS S3 et le schéma de données est créé, on passe à l'interrogation de données en utilisant les requêtes SQL basées sur Presto.

Pour intégrer Amazon Athena dans l'outil de détection [?], on distingue deux possibilités. La première possibilité n'utilise Athena que pour récupérer, dans la machine locale, les traceroutes vérifiés en terme de validité; c'est l'objectif des étapes 1 et 2 dans le processus de la création de l'évolution des RTTs différentiels des liens (voir la section II.4.4). Les traitements qui suivent (étapes à partir de 3) sont effectués dans la machine locale. Dans ce cas, l'utilisation des technologies Big Data est limité qu'au niveau stockage de données massives.

Tandis que la deuxième possibilité vise la maximisation des traitements 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 relatives à toutes les étapes décrites dans la section II.4.4.

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. Vue la complexité des étapes 1 à 9, on ne peut pas trouver une seule requête SQL assurant toutes les étapes. Supposons qu'il existe une requête SQL capable de trouver les liens possibles avec leurs RTTs différentiels : à l'étape 4 dans II.4.4, 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 $j \in [1, R]$ et R 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, sachant que les détails ne sont données, 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 peut inclure L liens. Dans ce cas, le nombre total, d'une période d_k , de mise à jour de la distribution des RTTs différentiels est de l'ordre $R \times L$ de fois. Cette estimation est à titre indicatif, de plus elle ne concerne que l'étape 4, le nombre de lectures et/ou d'écritures dépend des requêtes SQL créées pour les autres étapes.

En plus du nombre de lectures et d'écritures, la détection des anomalies repose sur une comparaison d'intervalles de confiances calculés par le score Wilson. 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.

Afin d'utiliser le service Amazon Athena à moindre coût, il est conseillé d'utiliser le partitionnement. Ce dernier permet d'analyser seulement les données concernées en ciblant ces dernières. De ce fait, moins de frais sont appliqués. Si un partitionnement particulier est adopté, le schéma de données est basé sur ce partitionnement ainsi que les requêtes SQL.

En ce qui concerne l'évolutivité d'une application basée sur ces trois services d'Amazon, on note que toute mise à jour de la structure de données des objets traceroutes peut affecter l'entière 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 effectuer le même traitement que celui fait avec MongoDB. Au lieu de récupérer les traceroutes depuis MongoDB, nous l'avons récupéré depuis Amazon S3 via Amazon Athena, c'est ce que nous l'avons appelé approche 1.

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, il est possible de personnaliser les traceroutes à récupérer depuis Amazon S3.

Le Tableau VI.2 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	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 VI.2 – 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 VI.4 présente un seul essai pour chacune des tailles utilisées auparavant avec MongoDB.

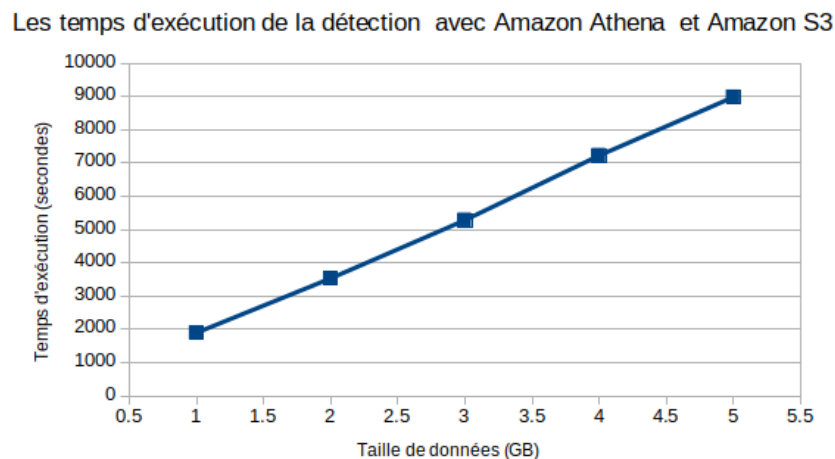


FIGURE VI.4 – Les temps d’exécution de la détection des anomalies en fonction de la taille de données (Amazon S3 et Amazon Athena)

VI.7 Application 4 : Spark Apache avec Scala

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.

Avec Spark, on travaille sur des collections d’objets, sur lesquelles on applique des traitements. Dans une application écrite en Spark, nous avons besoin des classes modélisant les objets tout au long de l’analyse de données, appelées *case class*. De plus, nous avons besoin de définir les fonctions à appliquer sur ces objets. Nous avons décrit les différentes *case class* conçues pour traiter les objets traceroutes, dans l’annexe V, en vue de tracer l’évolution des RTTs différentiels des liens au cours du temps.

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 Spark³, Spark réserve 1 GB pour le *driver* et 1 GB pour chaque *executor*.

Nous avons utilisé la version locale de Spark. Nous avons reproduit l'ensemble des étapes de détection en Scala. Ainsi, nous avons défini l'ensemble des traitements dans des fonctions ainsi que les classes permettant de modéliser les données tout au long de l'analyse. Ensuite, nous avons créé une archive (.jar) que nous soumettons à Spark, cette archive contient la fonction *main* qui s'agit du point d'entrée vers tous les traitements. De plus, nous avons paramétré la mémoire réservée au *driver* vu que la valeur par défaut (1 GB) ne convient pas à la quantité de données que nous souhaitons analyser.

Spark donne la possibilité de projeter les objets lus depuis des fichiers textes suivant le besoin de l'analyse, on parle du principe *Schema on Read* décrit dans la section IV.3.4. Ce que représente une flexibilité faisant face au problème de données manquantes. De plus, on ne lut que les données qui nous intéressent.

Nous avons utilisé Spark avec Scala comme API, il s'agit de la programmation fonctionnelle. Ce paradigme de programmation permet de décrire les traitements de façon élégante et claire, ce que facilite la mise à jour des traitements sur les données.

Le temps qu'on note est celui écoulé durant l'analyse des traceroutes donnés en entrée à travers les fichiers JSON. L'analyse inclut la lecture des fichiers et la détection des anomalies. Ces dernières sont stockées dans un fichier JSON pour une éventuelle réutilisation.

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. On note que nous avons aussi évalué la mémoire allouée au *driver*. Etant donné que la mémoire allouée par défaut au *driver* est de 1 GB, l'analyse d'un 1 GB de données a échoué pour l'insuffisance de mémoire pour traiter ce nombre important de traceroutes. C'est pourquoi nous avons paramétré l'analyse en précisant 30 GB comme taille de mémoire réservée au *driver*.

Variante la mémoire allouée au driver Nous avons varié la mémoire réservée au *driver* afin d'évaluer l'effet de la configuration de Spark en ce qui concerne le temps de l'analyse. Le Tableau VI.3 présente les temps d'exécution nécessaires pour analyser un nombre de traceroutes équivalent à une taille de 1 GB en utilisant différentes configurations.

driver-memory	période	durée	taille	Nombre traceroutes	Temps (secondes)
1 GB	07/02/2018	1 jour	1 GB	échec	échec
10 GB	07/02/2018	1 jour	1 GB	494158	2498
25 GB	07/02/2018	1 jour	1 GB	494158	2483
30 GB	07/02/2018	1 jour	1 GB	494158	2403

TABLE VI.3 – Les temps d'exécution en fonction de la mémoire allouée au driver

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

Il est possible d'ajuster la mémoire allouée aux *executors*, mais comme l'utilisation de Spark est en mode local, l'ajustement de la mémoire des *executors* n'a aucun effet.

Variante la taille de données Nous avons reproduit l'outil de détection en utilisant Spark avec l'API Scala comme c'est décrit dans l'annexe V. L'application Spark lit d'abord les traceroutes présents dans les fichiers donnés en entrée, ensuite, pour chaque période d_k , il faut parcourir tous les traceroutes afin de trouver les traceroutes effectués durant d_k , appelée approche 1. Le Tableau VI.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	1 jour	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 VI.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 aux 3 échantillons en une seule fois est de 20955, ce qui 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:  $D \leftarrow findAllPeriods(start, end, timeWindow)$ 
2: for all  $d_k \in D$  do
3:    $T \leftarrow loadTraceroutesFromInputFiles(dataPath)$ 
4:   for all  $t_j \in T$  do
5:      $checkTracerouteInPeriod(d_k, t_j)$ 
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 VI.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 VI.5 – Les temps d'exécution lors de la détection des anomalies en utilisant Spark (approche 2)

La Figure VI.5 permet de présenter les temps d'exécution pendant plusieurs essais et pour plusieurs ensembles de données.

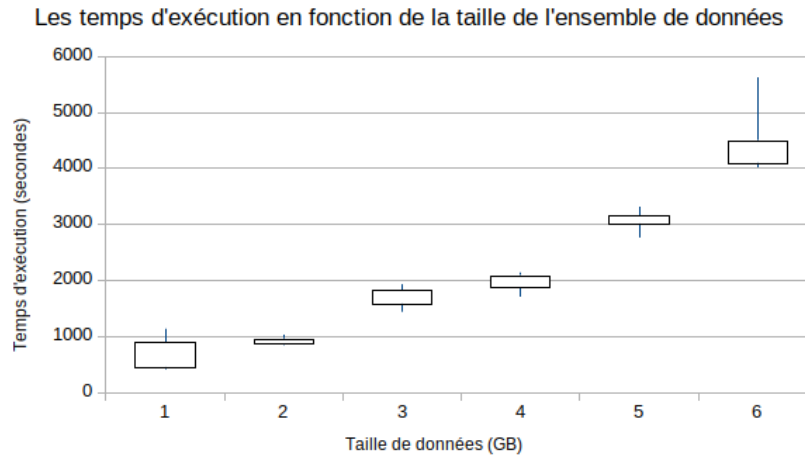


FIGURE VI.5

VI.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 VI.6⁴. 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.

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

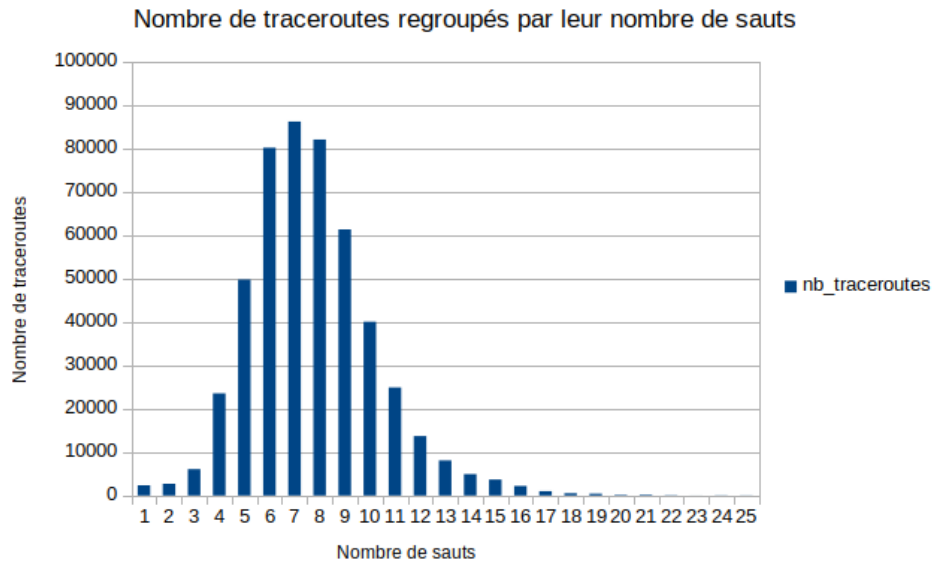


FIGURE VI.6 – Regroupement des traceroutes d’une journée par leurs nombre de sauts

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 VI.7 une idée sur le nombre de liens identifiés ainsi que les liens, parmi ceux identifiés, ceux ayant enregistré un moins une anomalie. La Figure VI.7 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.

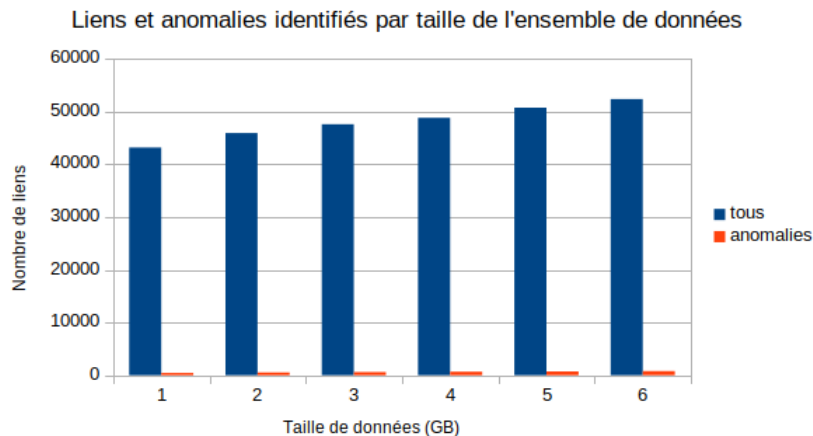


FIGURE VI.7 – Les liens et les anomalies identifiés par ensemble de traceroutes

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 VI.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).

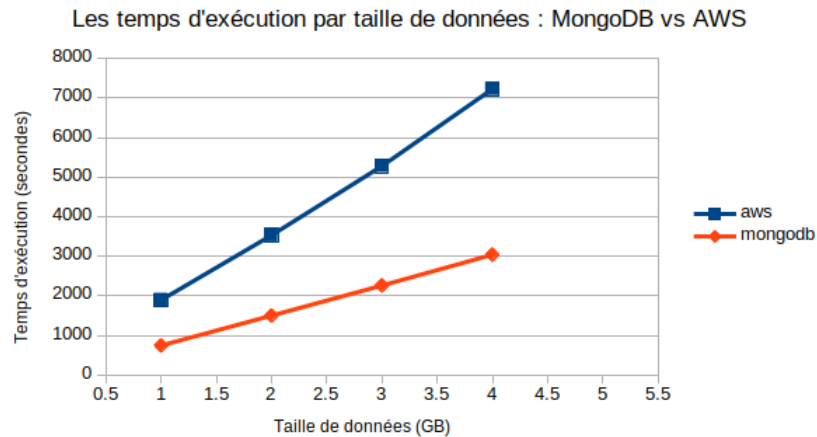


FIGURE VI.8 – Comparaison de l'évolution des temps d'exécution entre MongoDB et Amazon Athena avec Amazon S3

Apache Spark vs MongoDB

La Figure VI.9 illustre la moyenne des temps d'exécution pour MongoDB et Spark. Ces résultats concernent les mêmes ensembles de données.

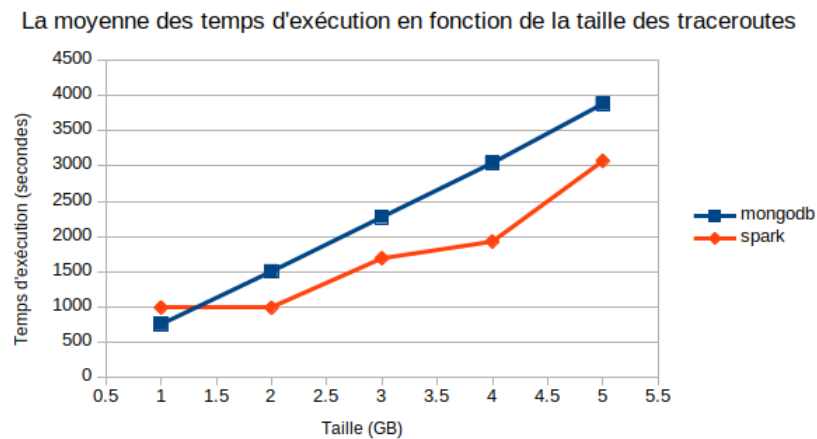


FIGURE VI.9 – La moyenne des temps d'exécution pour MongoDB et Spark

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 VI.10.

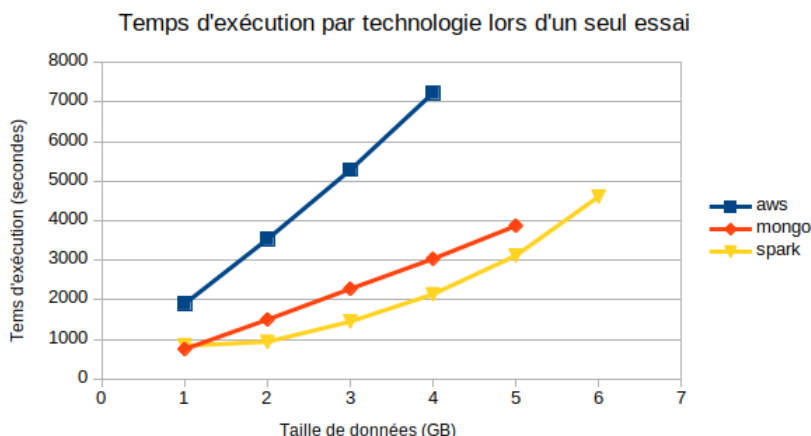


FIGURE VI.10 – Comparaison des temps d'exécution entre MongoDB, AWS et Apache Spark

VI.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 important. 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. [?] 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 :

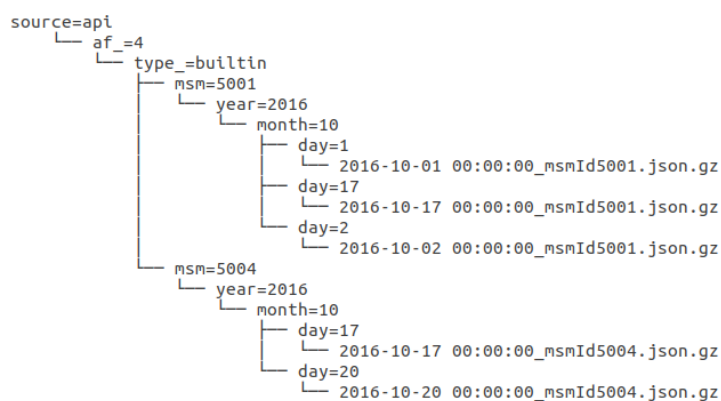


FIGURE A.1 – L'organisation des traceroutes dans un compartiment Amazon S3

A.2.3 L'interrogation de données sur Amazon Athena

L'interrogation de données présentes sur Amazon S3 via Amazon Athena est effectuée 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.

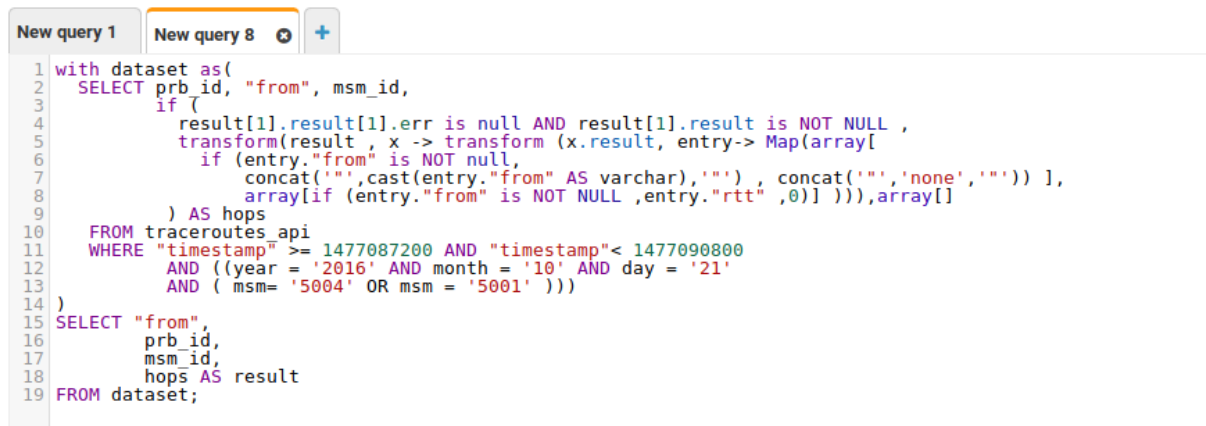
Pour les données sollicitées, ce sont les trois colonnes *prb_id*, *from*, *msm_id* (ligne 2) et la liste de saut (*hops*) obtenue après quelques vérifications (lignes 3 à 9). A la ligne 10, c'est la table créée pour les traceroutes (voir les détails de la table dans A.1). Les traceroutes à analyser 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
```

The image shows a screenshot of the Amazon Athena console's query editor. At the top, there are tabs for 'New query 1' and 'New query 8'. The query editor contains a SQL script with line numbers 1 through 19. The script defines a CTE named 'dataset' and then selects data from it, filtering by timestamp and msm values. The query uses complex nested functions like 'transform' and 'Map' to process the data.

```
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

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 B.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	Génération 1	8
I.2	Génération 2	8
I.3	Génération 3	8
I.4	Les trois générations des sondes Atlas	8
I.5	La connexion des sondes Atlas à l'infrastructure RIPE Atlas [?]	10
I.6	L'architecture du système RIPE Atlas	11
I.7	Les étapes d'établissement d'une connexion entre la sonde Atlas et l'architecture RIPE Atlas	13
I.8	La corrélation entre la moyenne des AS paths et la moyenne des RTTs [?]	25
I.9	Visualisation des changements des chemins traceroute [?]	26
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 : [?]	32
II.2	Illustration des périodes de l'analyse entre la date de début et la date de fin	33
II.3	Illustration des sauts d'un traceroute avec leurs informations	35
II.4	Inférence des liens possibles entre les routeurs des deux sauts consécutifs $R_{a,i}$ et $R_{b,j}$	36
II.5	La comparaison des deux intervalles de confiance : courant et référence	38
II.6	Le processus de la détection des anomalies dans les délais des liens	39
III.1	Phase I : préparation des données	45
III.2	Phase II : détection des changements	46
IV.1	Architecture standard du Big Data [?]	57
IV.2	Illustration d'une base de données NoSQL de type clé-valeur	59
IV.3	Illustration d'une base de données NoSQL de type document	59
IV.4	Illustration d'une base de données NoSQL de type colonne	59
IV.5	Illustration d'une base de données NoSQL de type graphe	60
IV.6	Bases de données NoSQL suivant le théorème de CAP	61
IV.7	Un classement des SGBDs sur <i>DB-Engines Ranking</i> du 1 août 2018	62
IV.8	Schema on Write (SGBDR)	63
IV.9	Schema on Read (Big Data)	63
IV.10	Interaction entre une application Spark et le cluster manager. Source : [?]	68
IV.11	Un exemple d'un cluster formé de trois executors. Source : [?]	68
IV.12	Spark Unified Stack. Source : [?]	69
IV.13	Exemple d'un flux de données avec Spark	70
V.1	Groupement des traceroutes avec MongoDB	74
VI.1	La médiane des temps d'exécution en fonction de la taille de traceroutes analysés (MongoDB)	85

VI.2 Les temps d'exécution en fonction de la taille de traceroutes analysés (MongoDB)	85
VI.3 Une combinaison des services web d'Amazon : Amazon S3, Amazon Glue, Amazon Athena, Amazon Quicksight et Amazon Redshift	86
VI.4 Les temps d'exécution de la détection des anomalies en fonction de la taille de données (Amazon S3 et Amazon Athena)	89
VI.5	93
VI.6 Regroupement des traceroutes d'une journée par leurs nombre de sauts	94
VI.7 Les liens et les anomalies identifiés par ensemble de traceroutes	94
VI.8 Comparaison de l'évolution des temps d'exécution entre MongoDB et Amazon Athena avec Amazon S3	95
VI.9 La moyenne des temps d'exécution pour MongoDB et Spark	95
VI.10 Comparaison des temps d'exécution entre MongoDB, AWS et Apache Spark . .	96
A.1 L'organisation des traceroutes dans un compartiment Amazon S3	100
A.2 Une exemple d'une requête SQL sur Amazon Athena	101

Liste des tableaux

I.1	Les caractéristiques du matériel des trois générations des sondes Atlas	8
I.2	Comparaison entre sondes et ancres RIPE Atlas	15
I.3	Comparaison entre sondes Atlas et ProbeAPI	22
I.4	Les détails des mesures effectuées dans le travail de C. Anderson [?]	24
II.1	Récapitulatif des traceroutes utilisés dans le travail de référence	30
IV.1	Les tarifs du AWS S3 (formule Stockage standard S3)	65
VI.1	Les temps d'exécution d'analyse de traceroutes en fonction de la taille de données avec MongoDB	85
VI.2	Les temps d'exécution par taille de l'ensemble de données (Amazon Athena et Amazon S3)	89
VI.3	Les temps d'exécution en fonction de la mémoire allouée au driver	90
VI.4	Les temps d'exécution des analyses lancées sur des traceroutes (approche 1) . .	91
VI.5	Les temps d'exécution lors de la détection des anomalies en utilisant Spark (approche 2)	92

Listings

III.1	Illustration	44
III.2	Les saut du traceroute T7 (sans agrégation)	47
III.3	Les saut du traceroute T7 (après l'agrégation)	47
III.4	Exemple des liens inférés du traceroute T7	48
III.5	Illustration de l'ordre des liens	48
III.6	Liste des liens possibles inférés via les traceroutes T1, T2, T3, T4 et T5	48
III.7	Caractérisation des liens identifiés lors de la période 1514769800 avec les tra- ceroutes T1, T2, T3, T4 et T5	48
III.8	Illustration de l'ordre des liens	49
V.1	Exemple de configuration avec SparkConf	72
V.2	Description du case class Traceroute	73
V.3	Description du case class Hop	73
V.4	Description du case class Signal	73
V.5	Le mapping entre données et cas class	73
V.6	Etape FindBins (I.1)	74
V.7	La classe TracerouteWithTimewindow	76
V.8	La classe TracerouteWithTimewindow	76
V.9	La classe TracerouteWithTimewindow	76
V.10	La classe TracerouteWithTimewindow	76
V.11	La classe TracerouteWithTimewindow	76
V.12	Deduction des liens par groupe de traceroutes	77
V.13	Définition de la classe ResumedLink	78
V.14	Deduction des liens par groupe de traceroutes	78
V.15	Deduction des liens par groupe de traceroutes	78
V.16	Définition de la classe ResumedLink	78
V.17	Définition de la méthode listAlarms	79
V.18	Définition de la méthode findAlarms	79
V.19	Exemple de la soumissions d'un traitement sur Spark	82
A.1	Création de la table des traceroutes dans Amazon Athena	98
B.1	Exemple d'une réponse traceroute	102