Déploiement d'une architecture Hadoop pour analyse de flux

François-Xavier Andreu
GIP RENATER
GIP RENATER c/o DSI Campus de Beaulieu, Bat 12 D
263, Avenue du Gal Leclerc CS 74205
35042 RENNES Cedex

Résumé

Le GIP RENATER exploite les exports de flux NetFlow générés par les équipements de niveau 3 du « backbone » RENATER. Cette information permet de calculer la consommation des utilisateurs mais aussi de voir les incidents de sécurité et de les analyser. Jusqu'en 2012 ces deux traitements étaient réalisés en temps réel par un collecteur NetFlow qui ne conservait aucune donnée brute. Le besoin de pouvoir travailler sur un historique des flux nous a amené -après une étude des produits libres et commerciaux- à déployer une architecture Hadoop (framework pour le big data) afin de garder une trace des flux et de les analyser à posteriori.

Hadoop est une plate-forme libre de la fondation Apache conçue pour réaliser des traitements sur des volumes de données massifs (dans notre cas environ 1,4To par jour). Il s'appuie sur son propre système de fichiers : HDFS (Hadoop Distributed File System) et implémente au-dessus une architecture type Map/Reduce. Il est aujourd'hui accompagné d'un véritable écosystème d'outils variés. L'architecture logicielle à mettre en place est très simple, mais nous nous sommes vite aperçu que l'optimisation de la plate-forme était primordiale (utilisation des disques et/ou de la mémoire, nombre de tâches Map/Reduce à paramétrer, réplication des données, spéculation…). Toutefois, l'installation et l'utilisation de Hadoop sont très simples. Les applications utilisatrices peuvent être réalisées en Java, C Python, Ruby et le parcours des données et les traitements distribués sont cachés à l'utilisateur. Dans notre cas l'architecture retenue privilégie la recherche rapide sur les dernières 24 heures mais un traitement d'agrégation en continu est mis en place pour la détection d'anomalies.

Mots-clefs

Hadoop, analyse de flux, NetFlow

1 Introduction

Depuis plus de 10 ans le GIP RENATER exploite les flux NetFlow générés par les équipements de niveau 3 du *backbone*. Cette information est utile à des fins d'études statistiques et d'analyse d'incidents de sécurité. L'outil utilisé est un collecteur développé par le GIP. Toutefois ces flux ne sont pas conservés en raison du volume important que constitue ce type de données (plusieurs centaines de giga-octets par jour). Le besoin de conserver un historique des flux sur quelques jours pour le travail du CERT RENATER était grandissant. Nous avons donc réalisé une étude en 2012 sur les produits libres et commerciaux qui pouvaient répondre à nos exigences. Ces solutions sont soit confrontées dans notre contexte à un problème de performance (les exports NetFlow représentent jusqu'à 400 000 flux par seconde en pleine journée), soit excessivement chères et majoritairement fermées pour un usage prédéfini des données (base de données non accessibles pour d'autres analyses). Notre choix s'est finalement porté sur la possibilité d'étendre au niveau fonctionnel notre outil de collecte des flux puisque seule la fonctionnalité d'un historique nous faisait défaut. Cette solution nous permet de garder les fonctionnalités déjà existantes qui ont été spécifiquement développées pour nos besoins. À cette fin nous devons pouvoir stocker et analyser plus de 8 milliards de flux par jour… une architecture distribuée semblait s'imposer. Depuis quelques années une solution existe, utilisée dans le contexte du *big data*, il s'agit du *framework opensource* Hadoop de la fondation Apache. C'est cette solution que nous avons retenue et que nous vous présentons dans cet article. Ce déploiement a été possible grâce à l'aide de l'équipe KerData de l'INRIA en particulier du chercheur Shadi Ibrahim.

2 Les contextes

2.1 Pré-étude

En 2012, nous avons mené une étude sur des outils de collecte et d'analyse de flux NetFlow. Cette étude a montré la difficulté de trouver un outil approprié à nos usages, à la fois « entreprise » et « opérateur ». Aucun des outils étudiés n'a cette capacité aujourd'hui. De plus le prix des solutions, en général supérieur à 200 000 euros dans notre configuration, en fait des outils haut de gamme pour lesquels un manque de fonctionnalités est difficilement envisageable. Un autre problème est la non-disponibilité des flux stockés pour un autre usage que celui de l'outil. Un travail d'analyse à posteriori de flux pour des études de sécurité ou de classification de flux n'est pas possible car les données stockées ne sont pas accessibles dans la plupart de ces solutions.

2.2 Les exports NetFlow sur le *backbone* RENATER

Un flux NetFlow est construit par l'équipement réseau à partir des paquets ayant la même clé. Cette clé est identifiée par les champs suivant : adresse IP source, adresse IP destination, protocole de transport, port source, port destination, interface d'entrée dans l'équipement et le champ type de service. Un flux est unidirectionnel et peut représenter un ensemble de paquets. D'autres champs – ne composant pas la clé – sont disponibles : le nombre de paquets, la taille total du flux, le *next-hop* bgp, les numéros d'AS, les masques de réseaux des adresses IP…

La Figure 1 - représente 16 flux qui ont été générés par le téléchargement d'une page web (www.jres.org). Le premier flux est la connexion au port 80 vers le serveur, qui lui répond par le port 80 qu'il faut utilisé le port 443 (https). Le reste des flux représente le chargement complet de la page, avec à chaque fois le port client qui est incrémenté par le système (50694 jusqu'à 50703), le port au niveau du serveur reste le même.

Nb Octets	Nb Paquets	Index In	Index Out	IP Source	IP Dest	Prot	Port Sce	Port Dst
625	6	5	13	10.0.0.53	192.168.0.176	6	50694	80
666	5	13	5	192.168.0.176	10.0.0.53	6	80	50694
8791	97	5	13	10.0.0.53	192.168.0.176	6	50695	443
265758	219	13	5	192.168.0.176	10.0.0.53	6	443	50695
5456	37	5	13	10.0.0.53	192.168.0.176	6	50698	443
81686	73	13	5	192.168.0.176	10.0.0.53	6	443	50698
3879	35	5	13	10.0.0.53	192.168.0.176	6	50699	443
69363	63	13	5	192.168.0.176	10.0.0.53	6	443	50699
13268	197	5	13	10.0.0.53	192.168.0.176	6	50700	443
559836	445	13	5	192.168.0.176	10.0.0.53	6	443	50700
4908	37	5	13	10.0.0.53	192.168.0.176	6	50701	443
71157	64	13	5	192.168.0.176	10.0.0.53	6	443	50701
4480	27	5	13	10.0.0.53	192.168.0.176	6	50702	443
51704	49	13	5	192.168.0.176	10.0.0.53	6	443	50702
1636	9	5	13	10.0.0.53	192.168.0.176	6	50703	443
2054	9	13	5	192.168.0.176	10.0.0.53	6	443	50703

Figure 1 - Flux engendrés par le téléchargement de l'URL www.jres.org

Un évènement sur Internet (clic sur un lien, ouverture d'un page…) ne produit pas qu'un seul flux en général. Sur RENATER, toutes les interfaces IP sont activées avec NetFlow (le NetFlow n'est pas activée sur les interfaces *Backbone* MPLS). Cela représente au plus fort de la journée jusqu'à 400 000 flux par seconde (Figure 2 -). Au total sur 24 heures nous obtenons plus de 17 milliards de flux. Certain de nos équipements sont configurés avec un échantillonnage par paquets ce qui réduit le nombre de flux ainsi obtenu. Ce dernier pourrait être bien supérieur. À notre avantage une partie des flux sont dupliqués car observés en plusieurs endroits sur le réseau. Ce n'est donc pas 17 milliards de flux que nous devons transférer vers Hadoop mais environ 8-9 milliards. Cela représente à peu près 1,4 To par jour (7Go toutes les 5 minutes). Ceux sont ces données que nous souhaitions pouvoir étudier aisément et si possible rapidement.

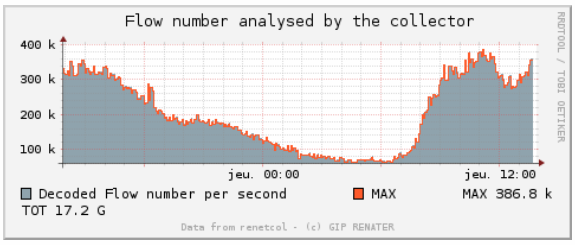


Figure 2 - Nombre de flux par seconde reçu par le collecteur

3 HADOOP

3.1 Présentation

Hadoop est un *framework* libre, géré par la fondation Apache, conçu pour analyser de très grande quantités de données. Il supporte le passage à l'échelle et est très performent en termes de tolérance aux pannes. Il est composé d'HDFS (*Hadoop Distributed File System*), son propre système de fichiers, et de MapReduce son « moteur » de calcul distribué.

Hadoop est accompagné de nombreuses briques s'appuyant sur le socle HDFS et/ou MapReduce. Comme l'illustre la Figure 3 - différents projets ont vu le jour, soit pour améliorer son utilisation avec des logiciels d'administration (ZooKeeper, Ambari, Hue…), de *scheduling* (Oozie, Azkaban…), de traitement temps réel (HBase, Impala…), soit avec des outils d'intégration de données (Flume, Dumbo…) ou encore d'apprentissage automatique (Mahout).

Les briques de base les plus importantes sont :

- HBase une base de données distribuée temps réel.
- Hive une interface pour exécuter des requêtes SQL.
- Pig une plateforme pour l'analyse des données via le langage Pig Latin.
- ZooKeeper un outil pour la coordination.

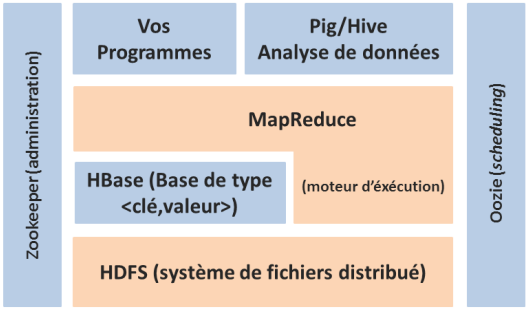


Figure 3 - Exemple d'architecture autour de Hadoop (en orange)

Ce grand nombre de logiciels nous indique la création d'une véritable communauté de recherche, développement et d'utilisateurs autour du *big data* que l'on aurait pu considérer comme une mode ou un sujet restreint à quelques entreprises spécifiques[1]. Hadoop est aujourd'hui l'outil majoritairement déployé là où les quantités de données à traiter ressortent du *big data* (au-delà de centaines de Gigaoctets). Il n'est toutefois pas nécessaire d'avoir des centaines de serveurs pour pouvoir l'utiliser (il peut être déployé sur un seul serveur, permettant de tester son déploiement aisément). Sur la page web qui liste les entreprises qui ont déployé Hadoop[2], on remarque même que les petits clusters sont bien plus nombreux que les grands. Ces derniers démontrent quand même que Hadoop peut être utilisé sur des parcs de plusieurs milliers de serveurs.

Nous n'avons pour l'instant utilisé que le *framework* de base : HDFS + MapReduce (cadre orange dans la Figure 3 -).

Pour installer une image virtuelle contenant Hadoop et une liste de briques préinstallés on peut se tourner vers les distributions Cloudera[3], Hortonworks[4] ou MapR[5]. Chacune d'elle a fait un choix dans les différents outils satellites d'Hadoop. MapR a même remplacé HDFS par son propre système de fichier.

3.2 Architecture d'un cluster Hadoop

Hadoop peut être installé sur un parc de serveurs hétérogènes qui peuvent être physiques ou virtuels. Hadoop est constitué d'un nœud maître le NameNode et de nœuds de traitement ; les DataNodes. Le NameNode gère les Jobs et possède la table des fichiers d'HDFS. Les DataNodes sont les nœuds sur lesquels les données et les Jobs sont distribués.

[1] voici une page web recensant la plupart des logiciels http://semanticommunity.info/Big_Data_at_NIST#Appendix_B._Solutions_Glossary
[2] http://wiki.apache.org/hadoop/PoweredBy
[3] http://www.cloudera.com/content/cloudera/en/home.html
[4] http://hortonworks.com/
[5] http://www.mapr.com/

Le NameNode n'est pas vraiment sollicité comparé aux autres nœuds, il est même possible de configurer un DataNode sur ce nœud qui aura une double fonction dans le cluster. Un second nœud NameNode peut être installé en cas de panne du premier, mais ce n'est pas indispensable pour les petits clusters.

3.3 Fonctionnement d'HDFS et MapReduce

HDFS s'appuie sur le système de fichier du système d'exploitation. Sous Linux c'est EXT3 qui est le plus utilisé, entre autre pour sa fiabilité. Il n'est pas recommandé d'utiliser de configuration RAID ou du LVM qui rajoute une couche non nécessaire à l'édifice. En effet HDFS gère sa propre réplication de données.

HDFS découpe chaque fichier en blocs de 64Mo par défaut. Ces blocs sont répartis sur les nœuds de données. Une tâche Map traite un seul bloc. Ce mécanisme est optimal pour des fichiers de grosses tailles. Pour traiter de nombreux fichiers de taille inférieure à celle d'un bloc, il n'est pas recommandé de baisser cette valeur, mais plutôt de concaténer les fichiers. Hadoop est plus performent avec des milliers de gros fichiers plutôt que des millions de petits. La taille par défaut 64Mo peut être augmentée (128Mo, 256Mo…) en fonction de vos données et de votre cluster.

Un des points fort d'Hadoop est sa tolérance aux pannes. Son fonctionnement repose essentiellement sur la réplication des données. Le choix est laissé à l'utilisateur, il est possible de désactiver la réplication ou de répliquer x fois les données. Par défaut le facteur de réplication est de 3. Notons que la distribution des blocs n'est pas vraiment homogène dans HDFS. La Figure 4 - montre un fichier réparti sur 10 nœuds qui ne l'est en fait que sur 8, certains ayant beaucoup plus de parties que d'autres. Même la réplication des données (ici 3) ne permet pas une distribution homogène. Comme les nœuds peuvent avoir à traiter des blocs de données qui ne sont pas sur leurs disques, ils sont envoyées par le réseau et cela induit de la latence dans les tâches.

Figure 4 - Répartition des blocs d'un fichier en fonction du facteur de réplication

Le patron MapReduce permet de traiter les données. Les tâches Map parcourent les données pour en extraire des informations de type <clé, valeur> qu'elles passent aux tâches Reduce qui se chargent de les agréger.

Pour comprendre facilement ce que peut faire Hadoop, on peut prendre comme exemple la commande *shell* suivante :

```
cat <mes données> |  <mon programme de parcours> | sort | <mon programme
d'agrégation> > <mon résultat>.txt
```

Avec Hadoop il est possible d'exécuter cette commande très simplement sur un cluster :

```
hadoop jar contrib/streaming/hadoop-*streaming*.jar \
-file <mon programme de parcours> -mapper <mon programme de parcours> \
-file <mon programme d'agrégation> -reducer <mon programme d'agrégation> \
-input <mes données> -output <mon résultats>
```

Elle est un peu plus longue mais les seuls paramètres à modifier sont les noms des programmes et leur emplacement.

Hadoop peut être aussi simple que cela… Bien sûr cette commande n'est qu'un aperçu de son utilisation où nous utilisons la fonctionnalité offerte par Hadoop de lire les données en streaming et de les analyser avec du code extérieur à Hadoop (autre que du Java). Les programmes peuvent être réalisés en C, Python ou autre, à condition de lire sur l'entrée standard type *stdin* et d'écrire sur *stdout*. Ce point permet de conserver son code existant lors d'un passage à Hadoop.

Notons que la version que nous avons employée jusqu'ici ne permet pas d'exploiter des données binaires. En effet Hadoop a été conçu pour traiter des données textes (logs, pages web…) dont les ensembles de valeurs sont séparés par des fins de lignes. Lorsque l'on donne à HDFS un fichier, celui-ci est découpé en bloc de 64Mo au niveau des retours à la ligne. Si vous n'avez pas de retours, le fichier sera découpé au beau milieu de vos champs. Certaines applications permettent d'exploiter des données binaires, comme Dumbo, mais nous n'avons pas eu le temps de la tester.

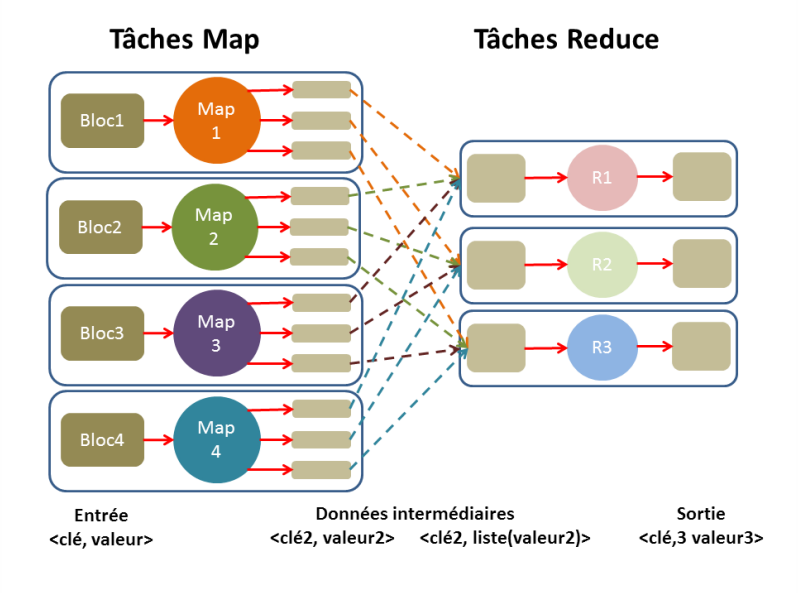


Figure 5 - Exécution d'un Job MapReduce

Les données sont lues par les tâches Map, chacune en charge d'un bloc HDFS. Une tâche Map prend une entrée de type <clé, valeur> et génère pour chaque tâche Reduce un fichier de ligne <clé2, valeur2>. Il est important de noter ce point, un reducer doit traiter X fichiers où X est le nombre de Map. Les tâches Reduce n'attendent pas que les séries de Map soient complètement finies pour commencer leur travail (essentiellement de copie des fichiers générés, flèches de couleurs sur la Figure 5 -). Les tâches Reduces agrègent ensuite les résultats des Maps. Nous avons donc en parallèle les deux tâches différentes. La Figure 6 - est l'analyse complète des différents temps d'exécution d'un Job. En vert les tâches Map composées de trois séries. En bleu le début des tâches Reduce qui commencent toutes par récupérer les fichiers résultats sur les différents nœuds sur lesquels les Map ont été lancées. En orange la phase propre de réduction.

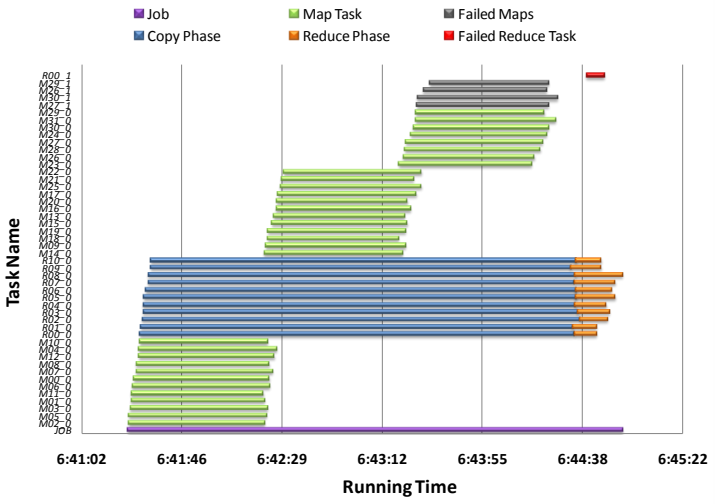


Figure 6 - *Visualisation du temps d'exécution des différentes tâches d'un Job – Shadi Ibrahim, INRIA*

Un autre point au sujet des tâches : celles-ci sont exécutées dans une JVM (*Java Virtual Machine*). Une JVM par tâche, et 2 JVM maximum par cœur. Donc sur un serveur biprocesseur multithreads ayant 32 cœurs vous avez à votre disposition 64 tâches possibles. Ce nombre est à répartir entre les tâches Map et Reduce. Par exemple 48 pour les Maps et 16 pour les Reduces. Les 48 tâches Maps peuvent traiter en simultané 48*64Mo de données soit 3Go. Si votre Job nécessite de parcourir 30Go vous aurez 10 séries de tâches Map. Si vous avez 10 serveurs cela ne fera qu'une série.

3.4 Les principales options

Hadoop a près de 200 paramètres configurables. Heureusement seulement 14 d'entre eux ont un impact significatif sur les performances d'un Job. La plus importante est le nombre de tâches Reduce que nous allons employer pour un Job. Mais nous pouvons également paramétrer la mémoire *buffer* allouée à une tâche Map ou Reduce, la proportion de tâches Map finies avant de lancer les tâches Reduce, réemployer les JVM et non les lancer pour une seule tâche… Il existe également une tâche *Combiner* qui peut être activée. Celle-ci s'exécute juste après les Maps en pré-agrégeant les données intermédiaires avant de les envoyer pour les tâches Reduce.

Une autre manière d'optimiser un Job est d'activer le mode *speculation*. Le JobTracker – démon qui gère les tâches du Job – supervise la performance de chaque tâche Map. Si l'une d'elle est trop lente, il va en lancer une autre en parallèle (sur un autre nœud) avec les mêmes données à analyser. Toutefois cette optimisation ne fut pas réellement avantageuse dans notre cas, surement dû au fait que nous n'ayons que des serveurs identiques.

Même si Hadoop est facilement utilisable à partir de tutoriaux nous recommandons de lire les livres [1] et [2] afin d'avoir une vue exhaustive de son utilisation.

Un point important avant d'utiliser un cluster Hadoop est de savoir si on en a réellement besoin. Cela dépend de vos données. Si celle-ci sont inférieures à quelques centaines de Giga-octets il peut être financièrement plus judicieux d'utiliser un seul serveur avec beaucoup de mémoire (plus que la taille de vos données), voir [3].

4 L'architecture déployée au sein du GIP RENATER

4.1 Choix et planification

Comme écrit précédemment, nous aurions pu choisir de louer une infrastructure dans un Cloud. Mais nous n'étions pas très à l'aise à l'idée de mettre nos flux (c'est-à-dire les données caractérisant le trafic de la communauté) quelque part sur Internet. Nous avons donc choisi d'acheter des serveurs physiques qui sont hébergés sur un nœud RENATER à côté du collecteur NetFlow. Nous avions la possibilité d'opter pour une infrastructure de virtualisation, mais dans ce cas il nous aurait fallu plus de compétences pour ce projet. En février 2013 nous avons étudié quelques devis, puis nous avons décidé d'attendre le mois de juillet, date à laquelle le GIP ferait parti de l'accord-cadre Matinfo3 de l'AMUE. Nous avons pu ainsi économiser près de 5k euros sur 40k euros. Mais notre projet a pris quelques mois de retard. Nous avons néanmoins pu installer Hadoop en *single-node* sur un serveur, mais nous étions loin d'un cluster.

La commande fut passée début juillet, les serveurs (11) reçus 15 jours plus tard. Nous avons déballé et « racké » ceux-ci juste avant nos congés d'été. Nous partîmes donc en congés sans avoir pu tester l'architecture. De retour mi-août nous avons installé les systèmes d'exploitation, le cluster fut opérationnel à la fin du mois.

4.2 *Hardware* et *software* utilisés

Les serveurs sont des DELL R420 (11 unités, taille 1U), biprocesseurs Xeon E5-2440 (2,4GHz, 6coeurs), 32Go de RAM, 1 disque système 15krpm de 146Go et 2 disques de données 15krpm de 300Go. Nous avons une capacité de 5,36To dans HDFS. En janvier cela nous paraissait suffisant pour stocker quelques jours de données (4 en l'occurrence). Mais nous ne savions pas que la répartition des blocs dans HDFS était aussi mauvaise, ce qui fait décroître les performances d'un Job et oblige fortement de répliquer les données. Aujourd'hui avec un facteur de réplication de 3, nous avons pour 30 heures de flux 4,5To à stocker. Les serveurs sont connectés à 1Gbit/s via un switch Cisco 3750.

Le système d'exploitation utilisé est Debian (64bits). Les principaux paquets utilisés autres que Hadoop sont Java, Munin[6] et dstat (pour le monitoring du cluster) ainsi que NetFilter pour la sécurité. Nous avons également utilisé pdsh[7] pour lancer une commande shell en parallèle sur chaque nœud.

La version d'Hadoop installée est la 0.20.0 qui n'est pas la plus récente mais c'était la version la plus connue de notre contact à l'INRIA. Comme il nous a apporté une aide précieuse, nous avons suivi son conseil. Notons qu'il est très facile de changer de version d'Hadoop, il suffit d'installer une nouvelle version dans un nouveau répertoire, de copier les fichiers de configuration vers ce nouveau répertoire (voir en Annexe les différents fichiers de configuration) et de changer le nom en prenant l'ancien. Nous avons suivi le tutorial de Michael G. Noll (http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/) qui est le tutorial de référence. Hadoop s'installe en moins d'une demi-heure si vous suivez à la lettre ses directives.

4.3 Expérimentations

Une fois qu'Hadoop est opérationnel vous avez à votre disposition de nombreux benchmarks pour le tester et vous faire la main sur sa configuration. Les 3 principaux programmes sont du type « *word count* » (très gourmand en mémoire et disque), « *sort* » (pour tester la mémoire) et « *π estimation* » (benchmark pour la CPU).

Pendant cette phase nous avons à chaque fois testé ces applications avec les données sur disque puis dans un répertoire monté en mémoire. Nous nous sommes rendus compte que les résultats étaient les mêmes, aucun gain à mettre les données en mémoire. Or la mémoire est bien plus rapide que les disques, même pour traiter de grandes quantités de données séquentiellement (nous avons testé nos serveurs avec iostat et dd). En fait comme nos serveurs possèdent 32Go de mémoire chacun, lors de tous nos tests nos données étaient dans la mémoire, soit montées directement soit dans le cache du système. Nous avons donc abandonné cette piste pour nos applications, les données ne pouvant pas être toutes stockées en mémoire. D'autant plus qu'il faut en garder pour les tâches Map et Reduce.

Nous avons également comparé la suite de commandes shell avec la commande Hadoop (voir 3.3). Sur un serveur solo, les deux commandes sont équivalentes, Hadoop n'est pas plus lent malgré le fait d'avoir à lancer des JVM. Et dès les premiers tests sur l'ensemble des nœuds les performances sont là (voir Annexe). Notons que la différence apparaît nettement lors de l'emploi de plusieurs Reducer.

Nous avons également testé l'intégration d'un nœud dans le cluster déjà opérationnel, cette action est très simple. De même nous avons changé en cours de route le facteur de réplication des données, cela s'effectue sans relancer Hadoop. Petit bémol quand même car si vous ne saviez pas qu'il existe un paramètre indiquant le taux de transfert maximum entre les nœuds pour ce cas, vous pouvez attendre longtemps avant que vos données soient répliquées. Par défaut ce paramètre est à 1Mb/s et si vous le changez, il faut redémarrer Hadoop.

Le redémarrage d'Hadoop est simple, mais avant d'être opérationnel cela peut prendre quelques minutes. Dans notre cas, avec 2,3 To de données sur 10 serveurs Hadoop est prêt en 2 minutes. Par contre il n'est pas nécessaire que tous les nœuds soient reconnus : nous avions toutes les données au bout de la reconnaissance de 8 serveurs sur les 10 (facteur de réplication à 3).

4.4 Nos applications

Nous avons actuellement deux types d'applications. Une application de recherche de flux et une application d'agrégation d'informations quantitatives. Ces deux applications sont constituées chacune de 2 programmes Python (Map et Reduce). Pour chacune d'elle nous avons suivi les conseils de Michael G. Noll et utilisé des générateurs, le gain est de 5 à 15% en temps d'exécution.

L'application de recherche est une application à la demande, elle est surtout utilisée par le CERT RENATER lors de problèmes de sécurité. La performance de cette application était primordiale lors du commencement du projet. En effet il n'est pas concevable d'attendre 30 min les flux d'une adresse IP. Nous avons aujourd'hui 22 secondes pour parcourir 5Go (5 minutes de flux), 1m30 pour 1h (50Go) et 5m pour 5h (250Go). Ces temps peuvent sembler longs, mais ils sont acceptables pour cette première architecture (sur un seul serveur, un *grep* met 22 secondes pour… 700Mo).

[6] Nous avons suivi ce tutorial : http://www.fotozik.fr/tuto-installation-de-munin-outil-de-monitoring, Munin fût installé et opérationnel sur le cluster en 15 minutes.
[7] pdsh (parallel distributed shell), il existe également pssh

Nous souhaitions également avoir des statistiques agrégées sur les préfixes, numéro d'AS, etc plus fines que celles de notre collecteur. Cette application, plus gourmande en ressources que la recherche, prend plus de temps à s'exécuter. Elle est activée toutes les 5 minutes sur les dernières 5 minutes reçues. Heureusement elle est finie au bout de 3 minutes. Les résultats sont mis dans une BD MySQL. Ces données agrégées sont archivées et permettent de détecter des anomalies (essentiellement sur la base de dépassement de seuil). Travailler sur un historique nous permet également d'avoir des TOP X ce que le collecteur ne pouvait pas faire à la volée.

5 Et Après ?

Notre architecture peut être améliorée sur plusieurs axes :

- Les données ne sont pas envoyées sur le cluster en temps réel, or certains logiciel peuvent le permettre (Flume, Kafka)
- Les données pourraient être compressées avant d'être mises dans HDFS et décompressées à la volée par les tâches Map.
- Installation d'Hadoop 2 qui a de nombreuses améliorations (dont l'intégration d'un ressource manager : YARN)
- Mettre un nœud data supplémentaire sur le nœud maître, voir déplacer le nœud maître sur le collecteur pour avoir un nœud data seul sur ce serveur.
- Tester Spark (http://spark.incubator.apache.org/) une implémentation théoriquement meilleure que le MapReduce d'Hadoop. Spark s'appuie sur HDFS mais optimise le traitement des données en mémoire.
- Utiliser un service Cloud RENATER pour avoir des dizaines de serveurs et un temps de réponse bien inférieur ?

6 Conclusion

Nous aurions pu appeler cet article « *Big Data analytics ? Take it easy with Hadoop* » tellement le *framework* est abordable. Bien entendu nos applications sont simples, mais elles s'appliquent désormais sur un cluster sans que l'on ait eu besoin d'écrire des programmes spécifiques à une grappe de calcul. L'aide que l'on trouve sur Internet ainsi que les recherches en cours autour du Big Data démontrent que cette solution est aujourd'hui mûre et surtout accessible au plus grand nombre. Mais le chemin n'est pas fini pour nous, il va nous falloir optimiser certaine partie de notre architecture…

Bibliographie

[1] t. white, Hadoop The Definitive Guide, O'REILLY Yahoo! Press, 2012.

[2] T. G. Srinath Perera, Hadoop MapReduce Cookbook, PACKT PUBLISHING, 2013.

[3] N. D. O. D. Rowstron, «Nobody ever got fired for using Hadoop on a cluster,» Microsoft Research, Cambridge.

Annexe

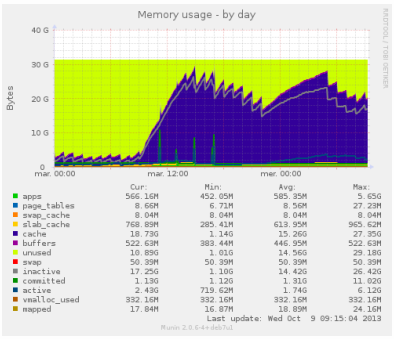


Figure 7 - Utilisation de la mémoire cache d'un nœud lors d'ajout de fichiers dans HDFS

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100,00%	255	0	0	255	0	0 / 0
reduce	100,00%	32	0	0	32	0	0 / 0

	Counter	Map	Reduce	Total
Job Counters	Launched reduce tasks	0	0	32
	Rack-local map tasks	0	0	57
	Launched map tasks	0	0	255
	Data-local map tasks	0	0	198
FileSystemCounters	FILE_BYTES_READ	3 364 672 348	3 316 057 077	6 680 729 425
	HDFS_BYTES_READ	17 094 054 398	0	17 094 054 398
	FILE_BYTES_WRITTEN	6 632 506 998	3 316 057 077	9 948 564 075
	HDFS_BYTES_WRITTEN	0	153 760 169	153 760 169
Map-Reduce Framework	Reduce input groups	0	9 441 321	9 441 321
	Combine output records	0	0	0
	Map input records	183 701 389	0	183 701 389
	Reduce shuffle bytes	0	3 303 078 584	3 303 078 584
	Reduce output records	0	9 441 321	9 441 321
	Spilled Records	367 402 778	183 701 389	551 104 167
	Map output bytes	2 948 653 957	0	2 948 653 957
	Map input bytes	17 093 026 048	0	17 093 026 048
	Map output records	183 701 389	0	183 701 389
	Combine input records	0	0	0
	Reduce input records	0	183 701 389	183 701 389

Nombre de tâches *Map* exécutées avec des blocs qui ne sont pas sur le nœud

Nombre total de tâches *Map* 255*64Mo=16Go

Taille des données à traiter ~16Go

Taille des résultats des tâches *Reduces*

Taille des résultats intermédiaires : sorties des *Maps*

Figure 8 - Résumé d'un Job

Comparaison du temps d'exécution des programmes mapper.py et reducer.py et de leur version optimisée avec des générateurs (commandes *shell*), taille des données 16Go :

```
hduser@H1:/usr/local/hadoop$ time cat /mnt/data2/RENATER DATA 20130912.ano | python
/home/hduser/mapper_rc.py | sort -k1,1 | python /home/hduser/reducer_rc.py > /tmp/tmpres.txt
real    16m52.850s
user    17m28.546s
sys     0m14.961s
hduser@H1:/usr/local/hadoop$ time cat /mnt/data2/RENATER DATA 20130912.ano | python
/home/hduser/mapper_rc_opt.py | sort -k1,1 | python /home/hduser/reducer_rc_opt.py >
/tmp/tmpres.txt
real    15m18.917s
user    15m52.056s
sys     0m15.153s
```

Utilisation de Hadoop avec les mêmes programmes Python:

- Avec 1 reducer :

```
hduser@H1:/usr/local/hadoop$ time bin/hadoop jar contrib/streaming/hadoop-*streaming*.jar -
file /home/hduser/mapper rc.py -mapper /home/hduser/mapper rc.py -file
/home/hduser/reducer rc.py -reducer /home/hduser/reducer rc.py -input
RENATER_DATA_20130912.ano -output mapper_reducer_python_on_16GB
packageJobJar: [/home/hduser/mapper_rc.py, /home/hduser/reducer_rc.py,
/mnt/data2/hadoop/tmp/hadoop-unjar4760798304806219021/] []
/tmp/streamjob8740989761538130444.jar tmpDir=null
13/10/03 13:53:58 INFO mapred.FileInputFormat: Total input paths to process : 1
13/10/03 13:53:58 INFO streaming.StreamJob: getLocalDirs(): [/mnt/data2/hadoop/mapreduce]
13/10/03 13:53:58 INFO streaming.StreamJob: Running job: job_201310031323_0003
13/10/03 14:03:20 INFO streaming.StreamJob: Job complete: job_201310031323_0003
13/10/03 14:03:20 INFO streaming.StreamJob: Output: mapper reducer python on 16GB
real    9m22.948s
user    0m1.856s
sys     0m0.324s
```

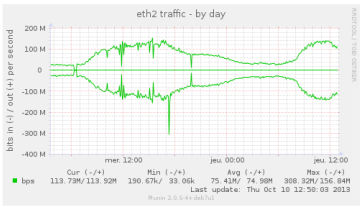
- Avec 16 reducers :

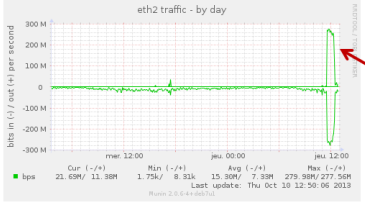
```
hduser@H1:/usr/local/hadoop$ time bin/hadoop jar contrib/streaming/hadoop-*streaming*.jar -D
mapred.reduce.tasks=16 -file /home/hduser/mapper_rc.py -mapper /home/hduser/mapper_rc.py -file
/home/hduser/reducer rc.py -reducer /home/hduser/reducer rc.py -input
RENATER DATA 20130912.ano -output mapper reducer python on 16GB16Red
packageJobJar: [/home/hduser/mapper rc.py, /home/hduser/reducer rc.py,
/mnt/data2/hadoop/tmp/hadoop-unjar8556470053887078301/] []
/tmp/streamjob6358409090578982157.jar tmpDir=null
13/10/03 14:14:55 INFO mapred.FileInputFormat: Total input paths to process : 1
13/10/03 14:14:55 INFO streaming.StreamJob: getLocalDirs(): [/mnt/data2/hadoop/mapreduce]
13/10/03 14:14:55 INFO streaming.StreamJob: Running job: job 201310031323 0005
13/10/03 14:16:04 INFO streaming.StreamJob: Job complete: job 201310031323 0005
13/10/03 14:16:04 INFO streaming.StreamJob: Output: mapper_reducer_python_on_16GB16Red
real    1m9.991s
```

- Avec 32 reducers :

```
hduser@H1:/usr/local/hadoop$ time bin/hadoop jar contrib/streaming/hadoop-*streaming*.jar -D
mapred.reduce.tasks=32 -file /home/hduser/mapper_rc.py -mapper /home/hduser/mapper_rc.py -file
/home/hduser/reducer rc.py -reducer /home/hduser/reducer rc.py -input
RENATER DATA 20130912.ano -output mapper reducer python on 16GB32Red
packageJobJar: [/home/hduser/mapper rc.py, /home/hduser/reducer rc.py,
/mnt/data2/hadoop/tmp/hadoop-unjar4463473973417112387/] []
/tmp/streamjob1858155122856828992.jar tmpDir=null
13/10/03 14:18:22 INFO mapred.FileInputFormat: Total input paths to process : 1
13/10/03 14:18:22 INFO streaming.StreamJob: getLocalDirs(): [/mnt/data2/hadoop/mapreduce]
13/10/03 14:18:22 INFO streaming.StreamJob: Running job: job 201310031323 0007
13/10/03 14:19:21 INFO streaming.StreamJob: Job complete: job 201310031323 0007
13/10/03 14:19:21 INFO streaming.StreamJob: Output: mapper_reducer_python_on_16GB32Red
real    0m59.957s
```

Statistiques réseau du nœud maître :
Actuellement sur notre architecture, le *Namenode* reçoit les données et les distribue sur l'ensemble des *Datanodes*, il y a donc autant de trafic reçu que transmit





Statistiques réseau d'un nœud de données lors du changement du facteur de réplication (+1)

Figure 9 - Changement du facteur de réplication

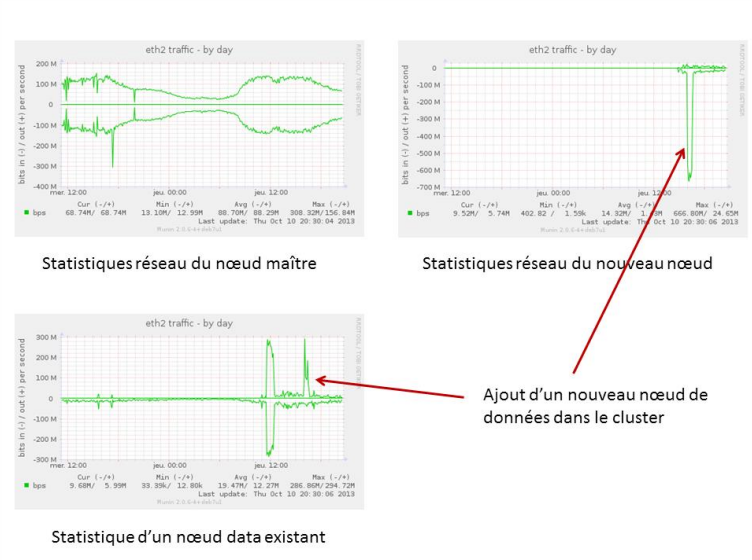


Figure 10 - Ajout d'un nouveau nœud dans Hadoop

Quelques liens :

- Running a typical ROOT HEP analysis on Hadoop/MapReduce , CHEP 2013 : http://indico.cern.ch/getFile.py/access?contribId=401&sessionId=4&resId=0&materialId=slides&confId=214784
- 7 Tips for Improving MapReduce Performance : http://blog.cloudera.com/blog/2009/12/7-tips-for-improving-mapreduce-performance/
- Yahoo! Hadoop Tutorial : http://developer.yahoo.com/hadoop/tutorial/
- Running Hadoop on Ubuntu Linux (Single-Node Cluster) : http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/
- Running Hadoop on Ubuntu Linux (Multi-Node Cluster) : http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/
- Hadoop wiki : http://wiki.apache.org/hadoop/FrontPage
- Monitoring Hadoop Clusters using free tools http://www.rundeconsult.no/?p=57&lang=en
- Big Data : La jungle des différentes distributions open source Hadoop : http://blog.ippon.fr/2013/05/14/big-data-la-jungle-des-differentes-distributions-open-source-hadoop/
- Hadoop dans ma DSI : comment dimensionner un cluster ? : http://blog.octo.com/hadoop-dans-ma-dsi-comment-dimensionner-un-cluster/
- Best Practices: Linux File Systems for HDFS : http://hortonworks.com/kb/linux-file-systems-for-hdfs/