10/11/2015

Rapport du projet de Cloud Computing

Prise en main de Docker, Hadoop et Storm

Membres du groupe (groupe Phœnix) :

- *MOUAFFO KENFACK Reine Vanessa*
- *TABUE Roméo*
- *TEGANTCHOUANG TEUKAP Boris Gaël*
- *TUENO FOTSO Steve Jeffrey (chef de groupe)*

Ecole Nationale Supérieure Polytechnique de Yaoundé (5GI)

GENIE INFORMATIQUE : ANNEE ACADEMIQUE 2015-2016

Table de matières

Présentation des Livrables

Nous avons envoyé par mail une version réduite des livrables (afin d'être dans la limite de 2 MB), qui contient le strict nécessaire pour comprendre ce que nous avons fait et pour effectuer les tests sur le projet concernant Docker (mise en place du cluster et test de l'Annuaire Web). Elle contient, en plus de ce rapport, un dossier Realisations_Logicielles constitué de :

- **Un dossier « BigDataHadoop »** contenant les réalisations logicielles et les résultats, sous forme de courbes, liés à l'implantation **sous Hadoop**, de l'application de comptage de mots dans un fichier (WordCount) **[Section Big Data sous-section Hadoop]**:
  - **Un dossier « Binaires »** contenant les scripts bash d'automatisation des simulations, avec une plage de paramètres, du WordCount sous Hadoop :
    - **« hadoop_tp.sh »** : qui reçoit des plages de paramètres et le fichier d'entrée et exécute l'implantation java/Hadoop du WordCount pour les différentes valeurs de paramètres prises chacune dans sa plage
    - **« hadoop_tp_suite.sh »**
    - **« simulate-hadoop.sh »** : le script principal qui reçoit les plages de paramètres et le fichier d'entrée, appelle hadoop_tp.sh avec ces plages de paramètres et le fichier d'entrée, récupère les résultats et trace les courbes
  - **Un dossier « Extrait_Codes_Sources »** contenant un extrait du code source java du WordCount sous Hadoop :
    - **« WordCount.java »**
  - **Un dossier « Resultats »** contenant les courbes des résultats obtenus à l'issue des simulations
- **Un dossier « BigDataSequentiel »** contenant les réalisations logicielles et les résultats, sous forme de courbes, liés à l'implantation **séquentielle, en java**, de l'application de comptage de mots dans un fichier (WordCount) **[Section Big Data sous-section Séquentiel]**:
  - **Un dossier « Binaires »** contenant le script bash d'automatisation des simulations, un certain nombre de fois pris en paramètre, du WordCount séquentiel:
    - **« simulate-sequentiel.sh »**
  - **Un dossier « Extrait_Codes_Sources »** contenant un extrait du code source java du WordCount séquentiel:
    - **« WordCountSequencial.java »**
  - **Un dossier « Resultats »** contenant la courbe des résultats obtenus à l'issue des simulations : une courbe représentant la répartition, par rapport à la moyenne, des durées d'un certain nombre d'exécution du WordCount séquentiel sur un certain fichier

- **Un dossier « BigDataStorm »** contenant les réalisations logicielles et les résultats, sous forme de courbes, liés à l'implantation **sous Apache Storm**, de l'application de comptage de mots dans un fichier (WordCount) **[Section Big Data sous-section Apache Storm]**:
    - **Un dossier « Binaires »** contenant le script bash d'automatisation des simulations, avec une plage de paramètres, du WordCount sous Storm :
        - **« simulate-storm.sh »** : script qui reçoit les plages de paramètres et le fichier d'entrée, exécute l'implantation java/storm du WordCount avec ces plages de paramètres et le fichier d'entrée, récupère les résultats et trace les courbes
    - **Un dossier « Extrait_Codes_Sources »** contenant un extrait du code source java du WordCount sous Storm :
        - **« FileSentenceSpout.java »** : l'implantation du Spout ayant pour source un fichier
        - **« WordSpliterBolt.java » :** l'implantation du Bolt Découpeur de phrase
        - **« WordCounterBolt.java » :** l'implantation du Bolt Compteur de mots
        - **« GlobalWordCounterBolt.java » :** l'implantation du Bolt Compteur Global en charge de la récupération des résultats des WordCounterBolt, de leur synthétisation et de l'écriture du décompte final dans le fichier de résultats
        - **« WordCountTopology.java » :** L'implantation de la classe principale en charge du découpage du fichier source selon le nombre de Spouts et de la création de la topologie
    - **Un dossier « Resultats »** contenant les courbes des résultats obtenus à l'issue des simulations

- **Un dossier « CloudComputing_Docker »** contenant les réalisations logicielles liées à l'implantation d'**un annuaire web J2EE distribué sous Docker [Section Cloud Computing] :**
    - **Un dossier « Apache »** contenant les outils de mise en place du container Apache :
        - **« Dockerfile »** : le script d'instanciation du container Apache
        - **« config_files » :** des fichiers supplémentaires utilisés lors de l'instanciation du container Apache
            - **« 000-default.conf »**
            - **« workers.properties »**
            - **« workers1.properties »**
            - **« apache.sh »** : le script contenant les instructions que la VM apache lancera à chaque démarrage, en particulier le démarrage du serveur Apache2
    - **Un dossier « MySQL »** contenant les outils de mise en place du container MySQL :
        - **« Dockerfile »** : le script d'instanciation du container MySQL

        - **« config_files » :** des fichiers supplémentaires utilisés lors de l'instanciation du container MySQL
            - **« annuaire.sql»** : le script SQL de création de la BD, des tables, des utilisateurs et des données de base
            - **« mysql.sh »** : le script contenant les instructions que la VM MySQL lancera à chaque démarrage, en particulier le démarrage du serveur mysql
    - **Un dossier « Tomcat »** contenant les outils de mise en place du container Tomcat :
        - **« Dockerfile »** : le script d'instanciation du container Tomcat
        - **« config_files » :** des fichiers supplémentaires utilisés lors de l'instanciation du container Apache
            - **« server.xml »**
            - **« tomcat7 »**
            - **« tomcat.sh »** : le script contenant les instructions que la VM Tomcat lancera à chaque démarrage, en particulier le démarrage du serveur Tomcat7
    - **Un dossier « Tomcat_clone »** contenant les outils de clonage d'une VM Tomcat, utilisés dans le cas où plusieurs VM Tomcat sont demandées
        - **« Dockerfile »** : le script de clonage d'une VM Tomcat à partir d'une VM Tomcat existante
    - **Un dossier « Extrait_Codes_Sources_Annuaire »** contenant un extrait du code source du projet J2EE Annuaire Web (uniquement les classes/servlets, les jsp, le web.xml et le script js principal) :
        - **Un dossier « Classes_Servlets »** contenant les classes/servlets du projet
        - **Un dossier « WEB-INF »** contenant les jsp, le web.xml et le script js principal
    - **« deploy.sh »** : script bash permettant de déployer et démarrer le cluster sous docker en lançant de façon intelligente, les dockerfiles de mise en place des différentes VMs, prenant en paramètre le nombre de tomcats
    - **« nbre_tomcats »** : fichier utilisé par deploy.sh pour stocker le nombre de tomcats pris en paramètres pour les prochains lancements du cluster
    - **« start.sh » :** script bash en charge du démarrage du cluster sous docker
    - **« stop.sh » :** script bash en charge de l'arrêt du cluster
- **Un dossier « TraceurDeCourbes »** contenant les codes sources et les binaires « .jar » de l'application prenant en entrée un fichier de données et générant en sortie les captures au format « .png » des courbes correspondantes, pour Hadoop et Apache Storm.

Vous pouvez retrouver, en vous rendant sur https://github.com/stuenofotso/Cloud-Computing-Project-Ressources La version complète des livrables. Cette version complète est

obligatoirement nécessaire pour reproduire les simulations concernant les implantations du WordCount en séquentiel, sous Hadoop et sous Apache Storm car contient, en plus, le code source complet de chacune des implantations, les binaires « .jar » des implantations incluant les librairies utilisées, les fichiers de résultats, bref, tout l'attirail que nous avons mis en place durant ces deux mois sur le projet.

Partie1: Cloud Computing

I- Contexte

Le Cloud Computing (informatique en nuage) est une infrastructure dans laquelle la puissance de calcul et le stockage sont gérés par des serveurs distants auxquels les usagers se connectent via une liaison Internet sécurisée. L'ordinateur de bureau ou portable, le téléphone mobile, la tablette tactile et autres objets connectés deviennent des points d'accès pour exécuter des applications ou consulter des données qui sont hébergées sur les serveurs. Le Cloud se caractérise par sa souplesse qui permet aux fournisseurs d'adapter automatiquement la capacité de stockage et la puissance de calcul aux besoins des utilisateurs [1].

Il apparaît donc clairement que le Cloud permet de stocker dans "le nuage" nos applications, données et tout autre genre d'informations. Mais avant de conserver nos applications dans ces serveurs, nous désirons être certains que cette dernière qui fonctionne parfaitement en local sur nos machines de développement, fonctionnera tout aussi bien sur le serveur du Cloud qui a une architecture et une configuration très différentes de celles de notre environnement de développement. L'architecture est différente dans le sens où elle offre :

- **L'abstraction sur la localisation :** en effet l'application est « quelque part » sur l'une des machines constitutives de la plateforme de virtualisation. Si cette plateforme utilise des mécanismes de réplication sur des Datacenters distants, les risques de désastre (incendies, inondations) sont couverts.
- **L'élasticité :** il est possible d'allouer des ressources supplémentaires à une application proche de la saturation, dans les limites physiques de la plateforme.
- **Le "Pay As You Go":** on ne paye que pour les ressources utilisées.
- **Le "self-service" :** les ressources sont disponibles dans des délais relativement courts

Etant limités en ressources, il faudrait trouver une solution qui nous permettrait une utilisation optimale de l'espace sur nos machines de développement c'est là qu'intervient Docker.

Docker offre la possibilité de faire des tests qu'on n'oserait pas faire sur sa propre machine, simuler, mutualiser un même serveur en isolant des utilisateurs sur une même machine physique. Il le fait à travers le mécanisme de containers dont il facilite la gestion. Ces derniers pouvant être vu comme des machines virtuelles très basiques, qui vont encapsuler les applications au sein d'un espace isolé. Chaque container va partager le kernel du système d'exploitation qui l'héberge. Docker va donc nous permettre de créer des containers applicatifs, de les packager, et de les déployer facilement.

II- Problématique

Nous souhaitons dans cette partie prendre en main docker et pour ce faire nous nous sommes reposés sur un petit exemple pour application du concept. L'exemple proposé est divisé en 3 segments:

- un serveur de base de données MySQL pour le stockage des données ;
- un module Tomcat qui va se charger du traitement des requêtes ;
- un module Apache qui s'occupera du load balancing.

Cette divisions en 3 blocs nous permettra d'avoir une bonne visibilité des parties de notre application, facilitera la maintenance et la réutilisation des composants.

Comment mettre en œuvre un outil de déploiement d'une architecture JEE (Apache + Tomcat + BD) sur un ensemble de conteneurs Docker? Telle est la problématique dans cette partie.

III- Solution

1) Description de la solution

Nous avons mis sur pied un ensemble de scripts permettant de déployer le cluster décrit précédemment sur tout terminal linux. Pour tester le bon fonctionnement de ce cluster une fois déployé, nous avons conçu et développé en J2EE un annuaire web exploitant les 3 étages du cluster et permettant à des utilisateurs de pouvoir consulter, ajouter, modifier et supprimer des contacts dans une base de données MySQL.

Nous rappelons que notre cluster est constitué de:

Une machine contenant essentiellement Apache et plus particulièrement son module de répartition de charge Mod_JK et permettant de répartir les requêtes des utilisateurs entre les différents serveurs d'application Tomcat

- Une machine contenant essentiellement MySQL, en charge du stockage des données
- Une ou plusieurs machines contenant chacune essentiellement Tomcat7, en charge du traitement des requêtes et de la génération des réponses HTTP

NB : Pour des raisons de rapidité et pour assurer la montée en charge, le nombre de Tomcat peut être paramétré.

Les scripts de déploiement du cluster peuvent se regrouper en deux catégories:

a) *Les Dockerfiles*

Il s'agit de scripts qui, pour chaque conteneurs, permettent de décrire la séquence d'instructions à exécuter par docker pour mettre en place la VM, de la récupération de l'image de base au script de démarrage en passant par l'installation et la configuration des paquets applicatifs.

A chaque VM (Virtual Machine) du cluster correspond donc un fichier de script dockerfile qui se retrouve dans un dossier portant le nom de la VM ainsi que les fichiers supplémentaires nécessaires à la mise en place de cette dernière:

- VM MySQL:
  - Le script de création de la BD, de création de l'utilisateur, d'allocation des droits, de création des tables et d'insertion des données de base
  - Le script de démarrage de la VM
- VM Tomcat:
  - Fichier servers.xml de configuration des ports d'écoute et des workers
  - Le script de démarrage de la VM
- VM Apache:
  - Fichier workers.properties de configuration des workers, mis à jour dynamiquement en fonction du nombre paramétré de VMs Tomcat
  - Fichier de mapping des urls aux workers
  - Script de démarrage de la VM

b) *Les Scripts BASH*

Nous avons mis sur pied:

Un script global de déploiement de l'architecture (Apache/Mod_JK + Tomcat + MySQL) en charge du séquencement des actions de création du cluster. Ce script s'appelle deploy.sh et se charge d'appeler les Dockerfiles de création des VMs et de créer les liens entre ces conteneurs ; liens qui sont utilisés au moment du démarrage des conteneurs, afin de procéder à la configuration de ces derniers. Finalement, ce script appelle le script de démarrage du cluster.

NB: Ce script installe et configure docker dans le cas où ce n'est pas encore fait.

- Un script en charge du démarrage du cluster qui se termine en affichant l'adresse URL permettant d'accéder à l'annuaire Web. Nous nous sommes assurés de rendre la prise en main et l'utilisation de cette application vraiment aisées: listing, édition, insertion, suppression de contacts.

- Un script en charge de l'arrêt du cluster

2) Déploiement

Notre projet se trouve dans le dossier CloudComputing_Docker du répertoire Réalisations_Logicielles dans lequel on retrouve les dossiers

- MySQL: qui a les fichiers de configuration de MySQL et un Dockerfile qui crée la machine virtuelle MySQL

- Tomcat: qui a les fichiers de configuration de Tomcat et un Dockerfile qui crée la machine virtuelle Tomcat

- Tomcat_clone: qui gère la multiplicité des Tomcat

- Apache: qui a les fichiers de configuration d'Apache et un Dockerfile qui crée la machine virtuelle Apache

Et les fichiers deploy.sh, start.sh et stop.sh.

Etapes :

a) Commencez par vous positionner dans le dossier du projet;

b) Pour le déploiement du cluster:

sudo ./deploy.sh nbre_de_parametres

Contraintes: nbre_de_parametres >=1

Ceci fait, la création de toutes les machines virtuelles a été faite et les liens entre elles ont été établis. A la fin deploy.sh appelle start.sh qui lance le cluster, vous devez donc copier l'url qui apparaît et le coller dans votre navigateur, ceci vous redirigera directement vers la page de notre application ou vous aurez donc la possibilité d'insérer, éditer, supprimer ou de consulter les contacts de l'annuaire.

Une fois le déploiement terminé

b) si vous souhaitez relancer le cluster:

sudo ./start.sh

c) si vous souhaitez stopper le cluster:

sudo ./stop.sh

IV- Résumé et perspectives

Il était question dans cette partie de prendre la main avec Docker, pour se faire nous avons implémenté un annuaire web distribué. Cette petite application a été subdivisée en 3 parties : une pour la gestion des données, une autre pour le traitement des requêtes et une dernière pour le load balancing des requêtes. Pour chaque étage de notre annuaire, nous avons eu à créer des containers (machines virtuelles) sous lesquelles il s'exécuterait. Ces machines virtuelles sont instanciées à travers l'exécution de docker files et scripts bash.

Il ressort de notre travail, la méthodologie à utiliser pour une mise en place claire, simple et facilement maintenable sous Docker d'une application avec une architecture multicouche.

Nous avons retenu plusieurs avantages dans l'utilisation de Docker et pouvons le recommander pour :

- Le partage d'un environnement de travail entre plusieurs personnes,
- Garder son système hôte propre, en installant tout ce dont on n'est pas sûr sous Docker,
- Avoir des versions spécifiques d'une librairie, d'un serveur, d'une base de données,
- La création, le paquetage et le déploiement facile des applications.

Partie2: Big Data

I- Contexte

Chaque jour, nous générons 2,5 trillions d'octets de données [2]. A tel point que 90% des données dans le monde ont été créées au cours des deux dernières années seulement. Ces données proviennent de partout : de capteurs utilisés pour collecter les informations climatiques, de messages sur les sites de médias sociaux, d'images numériques et de vidéos publiées en ligne, d'enregistrements transactionnels d'achats en ligne et de signaux GPS de téléphones mobiles. Le Big Data couvre quatre dimensions: volume, vélocité, variété et véracité [2].

Volume : les entreprises sont submergées de volumes de données croissants de tous types, qui se comptent en téraoctets, voire en pétaoctets.

- Transformer les 12 téraoctets de Tweets créés quotidiennement en analyse poussée des opinions sur un produit
- Convertir les 350 milliards de relevés annuels de compteurs afin de mieux prédire la consommation d'énergie

Vélocité : parfois, 2 minutes c'est trop. Pour les processus chrono sensibles tels que la détection de fraudes, le Big Data doit être utilisé au fil de l'eau, à mesure que les données sont collectées par l'entreprise afin d'en tirer le maximum de valeur.

- Scruter 5 millions d'événements commerciaux par jour afin d'identifier les fraudes potentielles
- Analyser en temps réel 500 millions d'enregistrements détaillés d'appels quotidiens

Variété : le Big Data se présente sous la forme de données structurées ou non structurées (texte, données de capteurs, son, vidéo, données sur le parcours, fichiers journaux, etc.). De nouvelles connaissances sont issues de l'analyse collective de ces données.

- Utiliser les centaines de flux vidéo des caméras de surveillance pour contrôler les points d'intérêt
- Tirer parti de la croissance de 80 % du volume de données image, vidéo et documentaires pour améliorer la satisfaction client

Véracité : 1 décideur sur 3 ne fait pas confiance aux données sur lesquelles il se base pour prendre ses décisions [2]. Comment s'appuyer sur de l'information en laquelle on n'a pas confiance? Etablir la confiance dans les Big Data représente un défi d'autant plus important que la variété et le nombre de sources augmentent.

II- Problématique

Il est question ici d'étudier les performances de deux implantations d'une application de comptage de mots avec deux solutions du Big Data: Hadoop et Apache Storm.

L'application doit prendre en entrée un grand fichier texte et délivrer en sortie un fichier contenant une liste de tuples <mot, décompte>. Pour chaque implantation, le travail principal consiste à chercher à obtenir les performances maximales avec le même nombre de processeurs; ainsi, il s'agit de rechercher les paramètres permettant d'atteindre les meilleures performances.

Pour Hadoop, nous allons varier la taille des blocs, de 1MB à 1GB par puissance de 2, et mesurer les temps d'exécution pour un fichier texte de 900MB.

Pour Apache Storm, nous allons varier le nombre de Spouts et le nombre de Bolts Découpeurs de 1 à 7 par pas de 2 et le nombre de Bolts Compteurs de mots de 1 à 101 par pas de 10 et mesurer les temps d'exécution pour un fichier texte de 30 MB. Réaliser cette tâche nécessitant de mettre en place une implantation de l'application de comptage de mots où les spouts lisent les phrases dans un fichier passé en paramètres et surtout qui est capable de s'arrêter à la fin du fichier (le fichier étant de taille dynamique) et afficher la durée de son exécution.

Vu l'immensité du nombre de paramètres, nous avons aussi cru nécessaire de mettre en place des scripts d'automatisation de la collecte des données, du tracée des courbes et de la sauvegarde de celles-ci sous forme d'images exploitables à des fin d'analyse.

III- Solution : cas de Hadoop

1 Description de la solution

Nous avons utilisé l'application Java WordCount telle que donnée en cours.

L'objectif étant de trouver la taille des blocs manipulés dans les datanodes par Hadoop permettant de minimiser le temps de traitement, plusieurs scripts bash ont été développés, permettant, du moment où Hadoop est correctement installé et configuré, à partir de la plage des valeurs de la taille des blocs, du pas de variation dans la plage et surtout du nom du fichier source, d'automatiquement :

- Stopper les daemons d'Hadoop s'ils étaient déjà lancés ;
- formater le système de fichiers hdfs ;

- Démarrer les daemons ;
- Créer les répertoires dans le système de fichiers hdfs nécessaires à l'exécution (*/user/$USER/input) ;*
- Exécuter l'application Java WordCount pour chaque valeur de la taille des blocs dans la plage ;
- De récupérer les différents temps d'exécution et de tracer les courbes correspondantes

Il s'agit de hadoop_tp.sh, hadoop_tp_suite.sh et simulate-hadoop.sh présents dans le sous répertoire Binaires du répertoire BigDataHadoop.

Pour refaire la simulation pour une plage et un fichier texte donnés, il faut télécharger la version complète des livrables(voir Annexe), se rendre dans le sous dossier Binaires du répertoire BigDataHadoop, *contenant launch_hadoop_perf.jar (servant au tracé des courbes à partie du fichier de résultats), wc.jar, hadoop_tp.sh, hadoop_tp_suite.sh et simulate-hadoop.sh,* et lancer la simulation en saisissant la commande :

chmod 755 * #pour donner les droits d'exécution sur les scripts
./simulate-hadoop.sh taille_min pas taille_max nom_fichier

Où taille_min est un entier, puissance de 2, supérieur ou égal à 1048576 représentant la valeur minimale de la taille des blocs, pas est un entier, puissance de 2, représentant le quotient de la division entre deux valeurs successives de la taille des blocs et taille_max est un entier, puissance de 2, supérieur à taille_min, représentant la valeur maximale de la taille des blocs.

L'on obtient en sortie un fichier de données donnant pour chaque taille de blocs le temps mis pour l'exécution ainsi qu'un fichier image, capture de la courbe de variation du temps d'exécution en fonction de la taille de blocs.

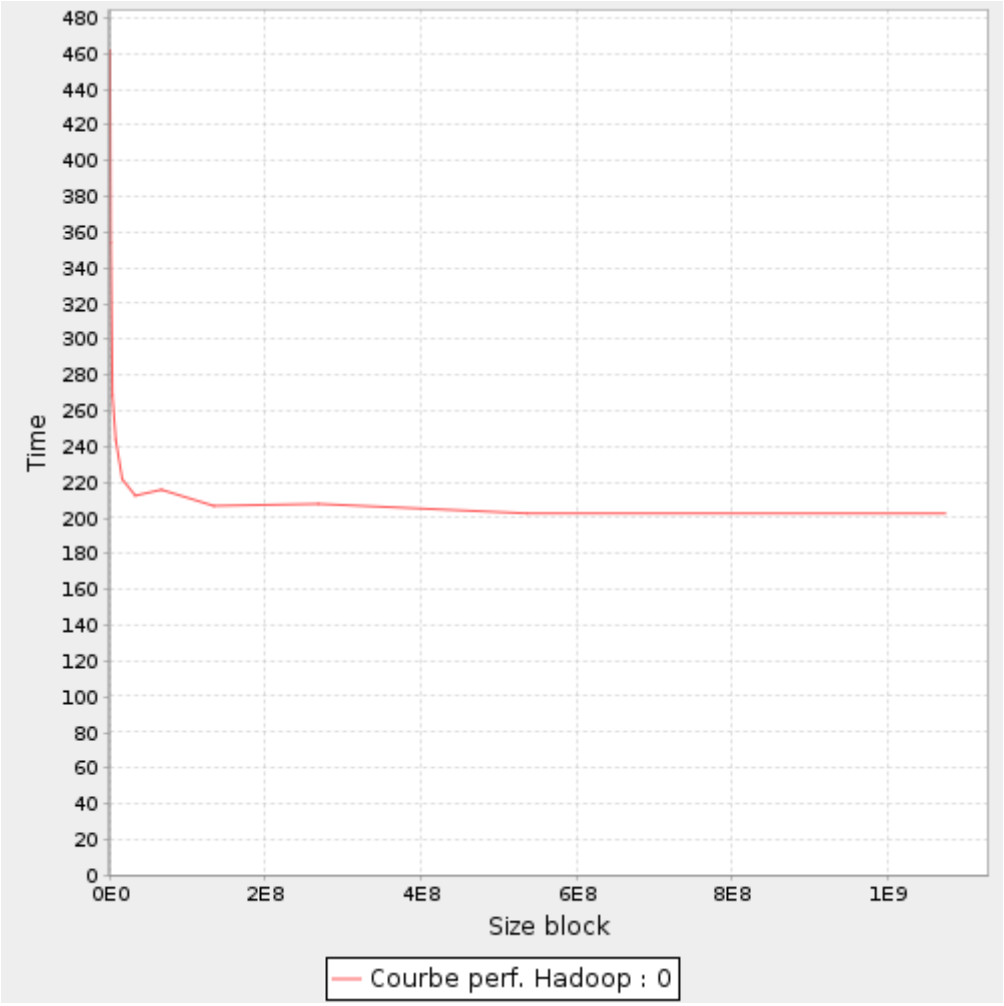
2- Analyse des résultats et conclusion

Nous avons fait varier la taille des blocs, de 1MB à 1GB par puissance de 2, et nous avons mesuré les temps d'exécution pour un fichier texte de 900MB. Chaque mesure s'est faite 10 fois, pour réduire les risques d'erreurs et les résultats finaux sont la médiane des résultats obtenus. Les simulations se sont effectuées sur Linux Ubuntu 14.4 LTE tournant sur un PC core duo 2.56 GHz *2, 2 Go de RAM.

Vous pouvez retrouver ces données dans le sous dossiers Résultats du dossier BigDataHadoop présent dans le répertoire Realisations_Logicielles.

a- Résultats de la première prise de mesure

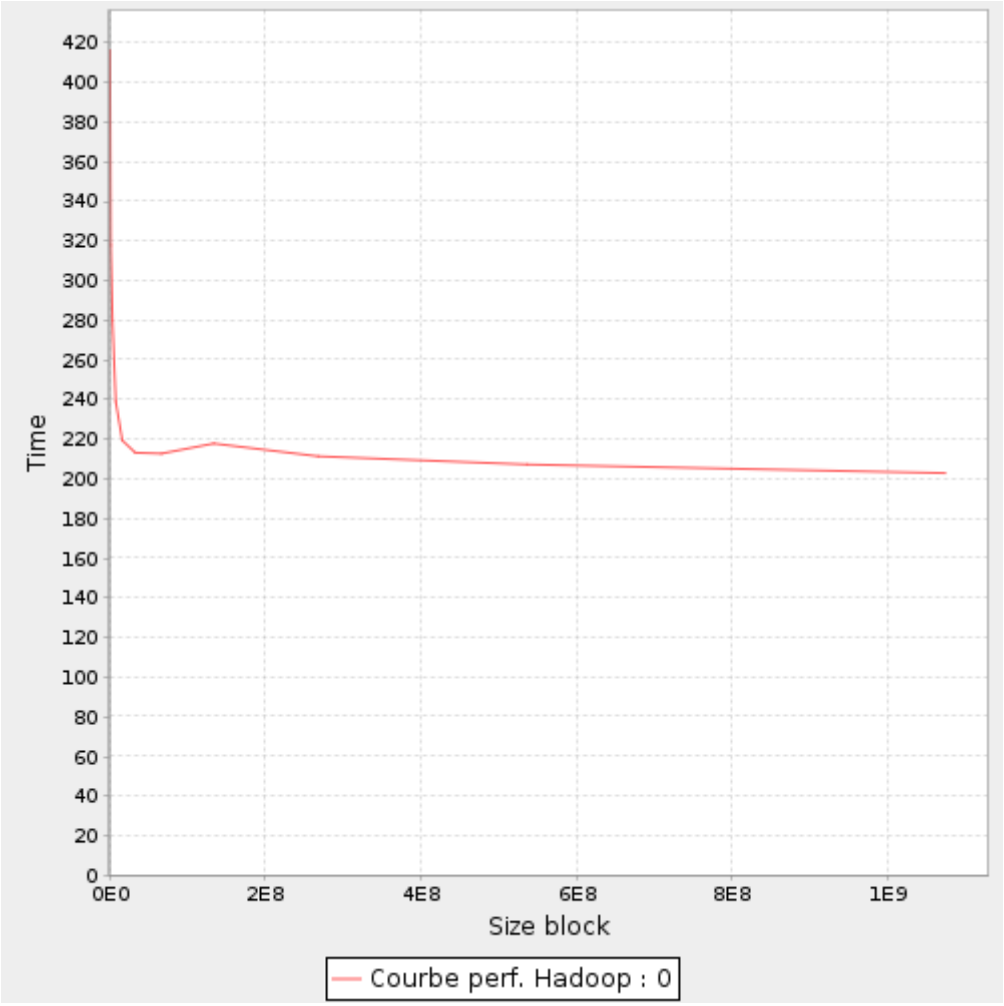
taille	1048576	2097152	4194304	8388608	16777216	33554432	67108864	134217728	268435456	536870912	1073741824
Durée(s)	461.88	354.03	268.70	243.64	221.52	212.38	215.67	206.65	207.73	202.43	202.39



Ici, on remarque que le temps optimum est 202.39 Secondes, obtenu avec une taille de blocs de 1073741824 Octets

b- Résultats de la deuxième prise de mesure

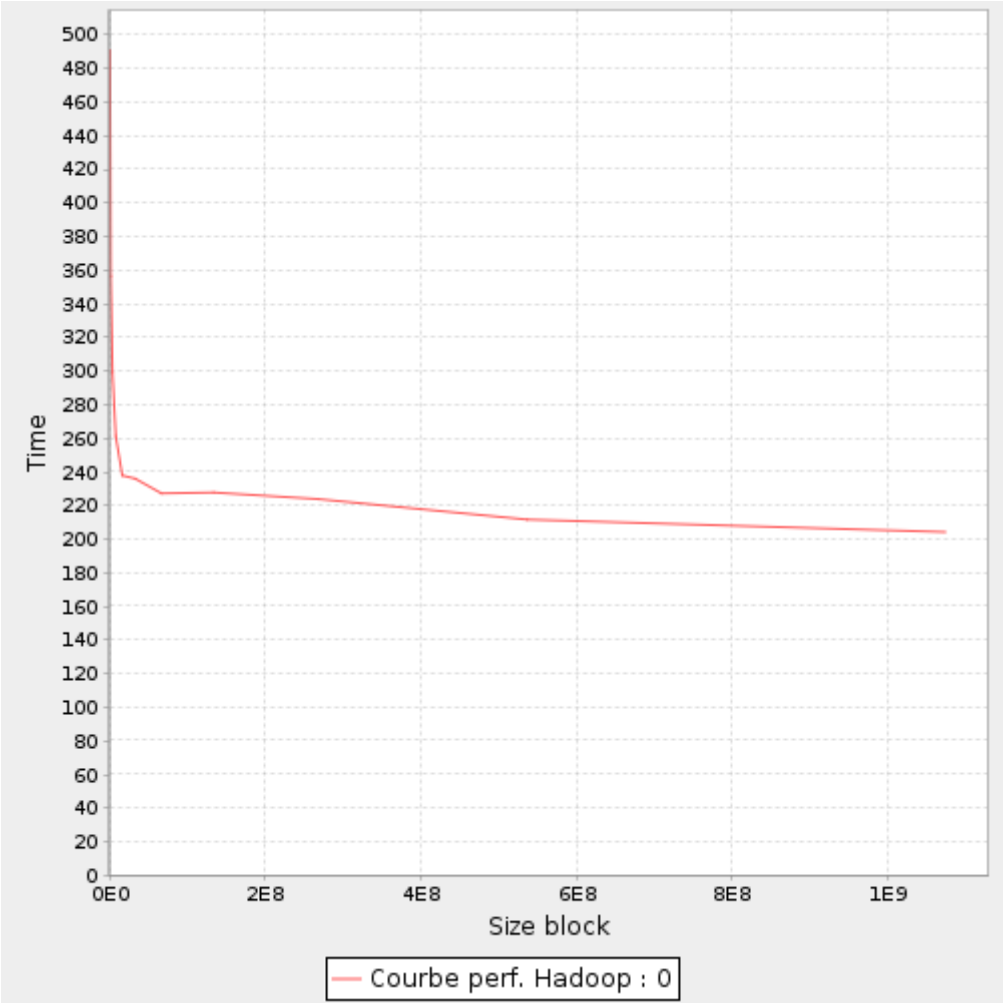
Taille(Octets)	1048 576	2097 152	4194 304	8388 608	1677 7216	3355 4432	6710 8864	13421 7728	26843 5456	53687 0912	10737 41824
Durée (S)	416. 22	317. 73	279. 83	238. 85	219.2 5	213.0 3	212.6 0	217.6 8	211.2 3	207.1 0	202.75



Ici, on remarque que le temps optimum est 202.75 Secondes, obtenu avec une taille de blocs de 1073741824 Octets

c- Résultats de la troisième prise de mesure

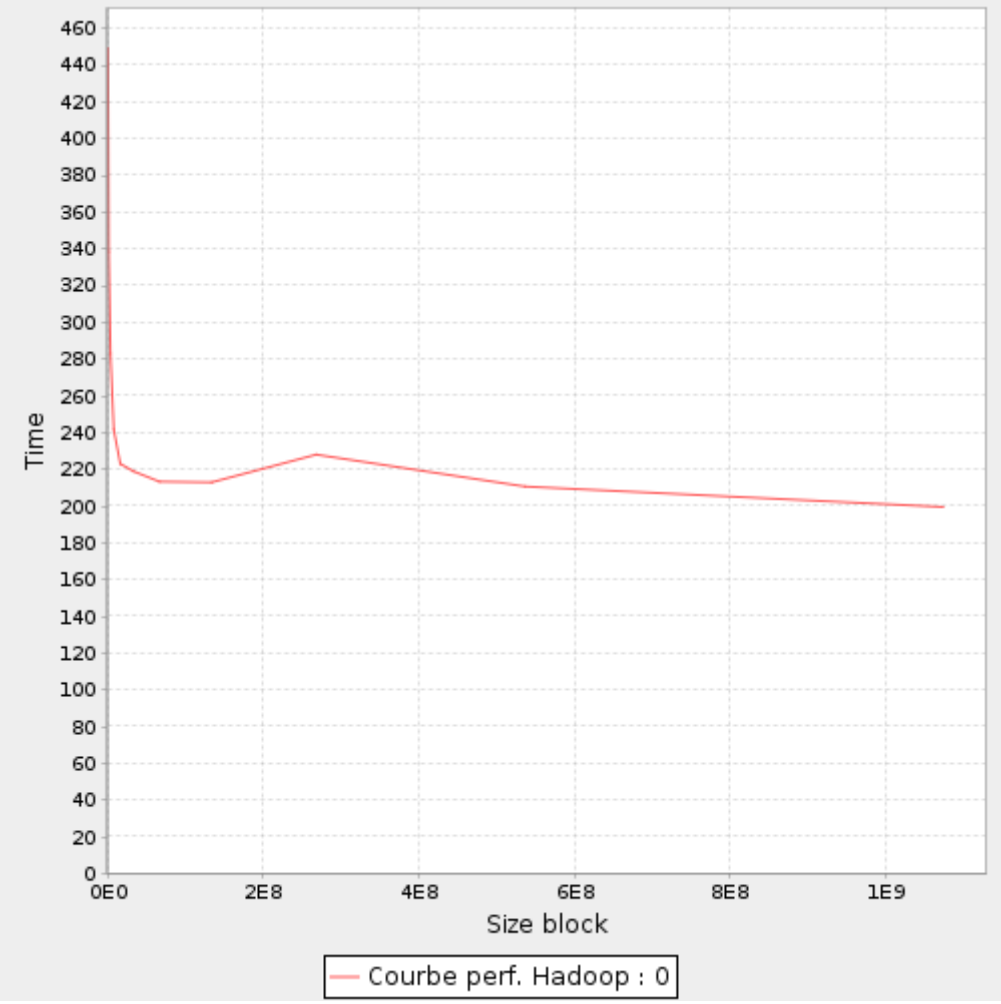
Taille(Octets)	1048 576	2097 152	4194 304	8388 608	1677 7216	3355 4432	6710 8864	13421 7728	26843 5456	53687 0912	10737 41824
Durée (S)	490. 83	356. 35	299. 35	261. 13	237.6 9	235.7 6	227.0 3	227.5 4	223.6 3	211.4 4	204.02



Ici, on remarque que le temps optimum est 204.02 Secondes, obtenu avec une taille de blocs de 1073741824 Octets

d- Résultats de la quatrième prise de mesure

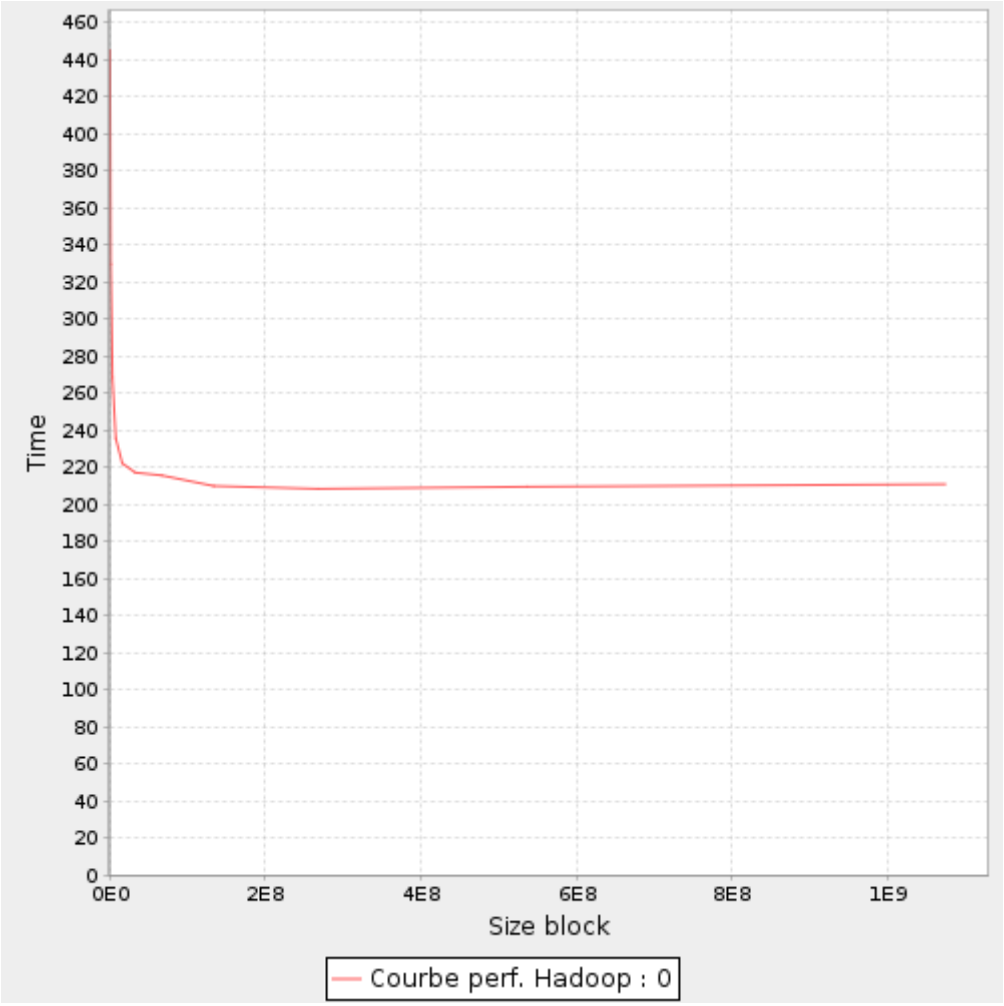
Taille(Octets)	1048 576	2097 152	4194 304	8388 608	1677 7216	3355 4432	6710 8864	13421 7728	26843 5456	53687 0912	10737 41824
Durée (S)	449. 18	338. 33	282. 86	241. 47	222.8 0	218.7 2	213.0 6	212.7 0	227.7 9	210.3 9	199.27



Ici, on remarque que le temps optimum est 199.27 Secondes, obtenu avec une taille de blocs de 1073741824 Octets

e- Résultats de la cinquième prise de mesure

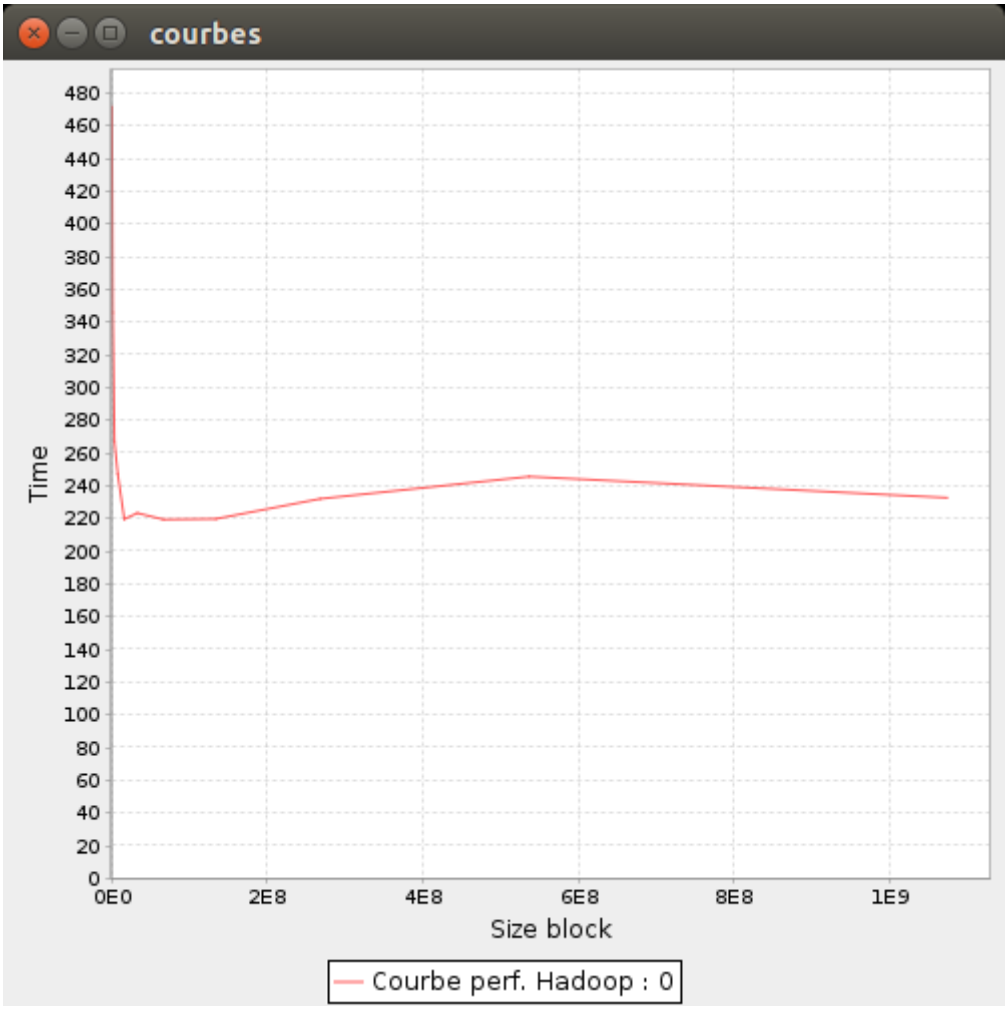
Taille(Octets)	1048 576	2097 152	4194 304	8388 608	1677 7216	3355 4432	6710 8864	13421 7728	26843 5456	53687 0912	10737 41824
Durée (S)	444. 87	329. 47	268. 32	235. 51	221.9 7	216.9 6	215.4 9	209.7 6	208.2 1	209.3 3	210.72



Ici, on remarque que le temps optimum est 208.21 Secondes, obtenu avec une taille de blocs de 268435456 Octets

f- Résultats de la sixième prise de mesure

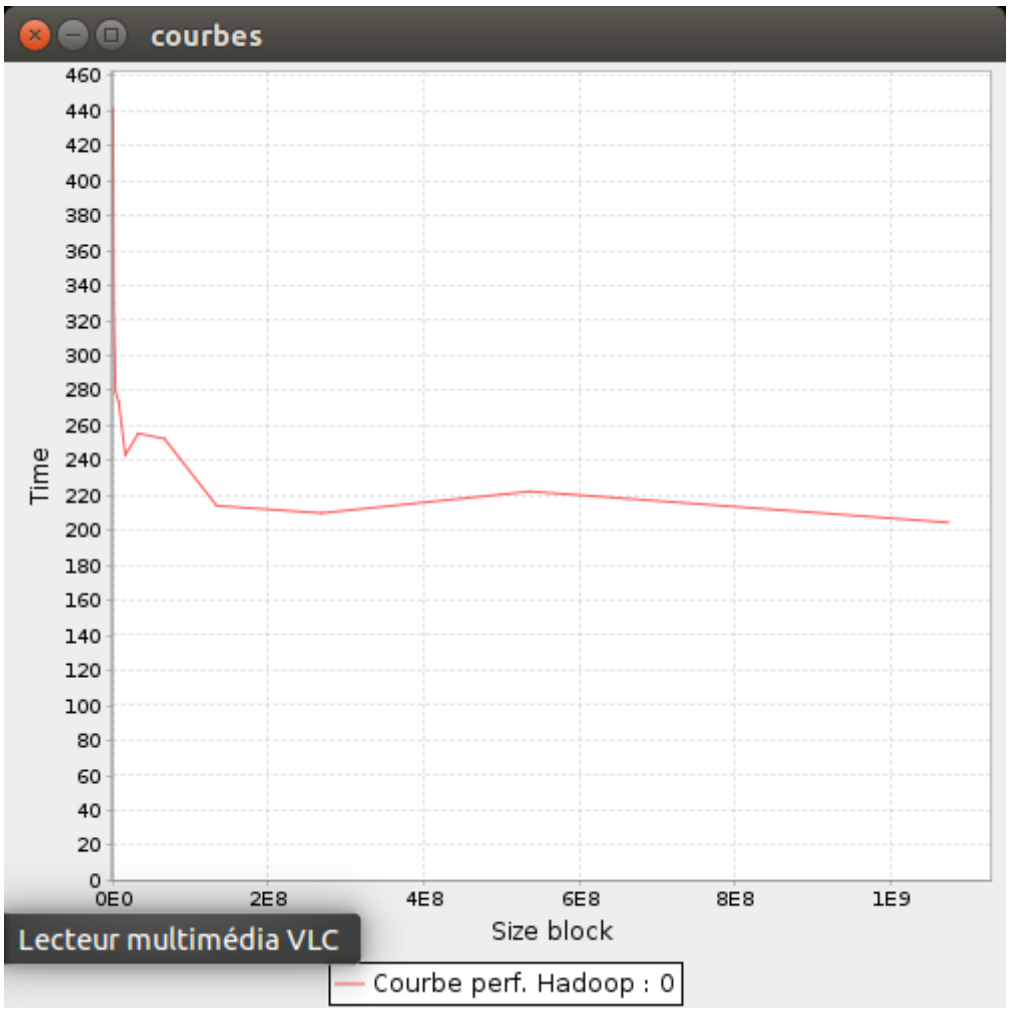
Taille(Octets)	1048 576	2097 152	4194 304	8388 608	1677 7216	3355 4432	6710 8864	13421 7728	26843 5456	53687 0912	10737 41824
Durée (S)	471. 69	345. 68	266. 90	247. 03	219.3 6	223.0 7	219.1 7	219.5 1	231.8 0	245.4 0	232.46



Ici, on remarque que le temps optimum est 219.17 Secondes, obtenu avec une taille de blocs de 67108864 Octets

g- Résultats de la septième prise de mesure

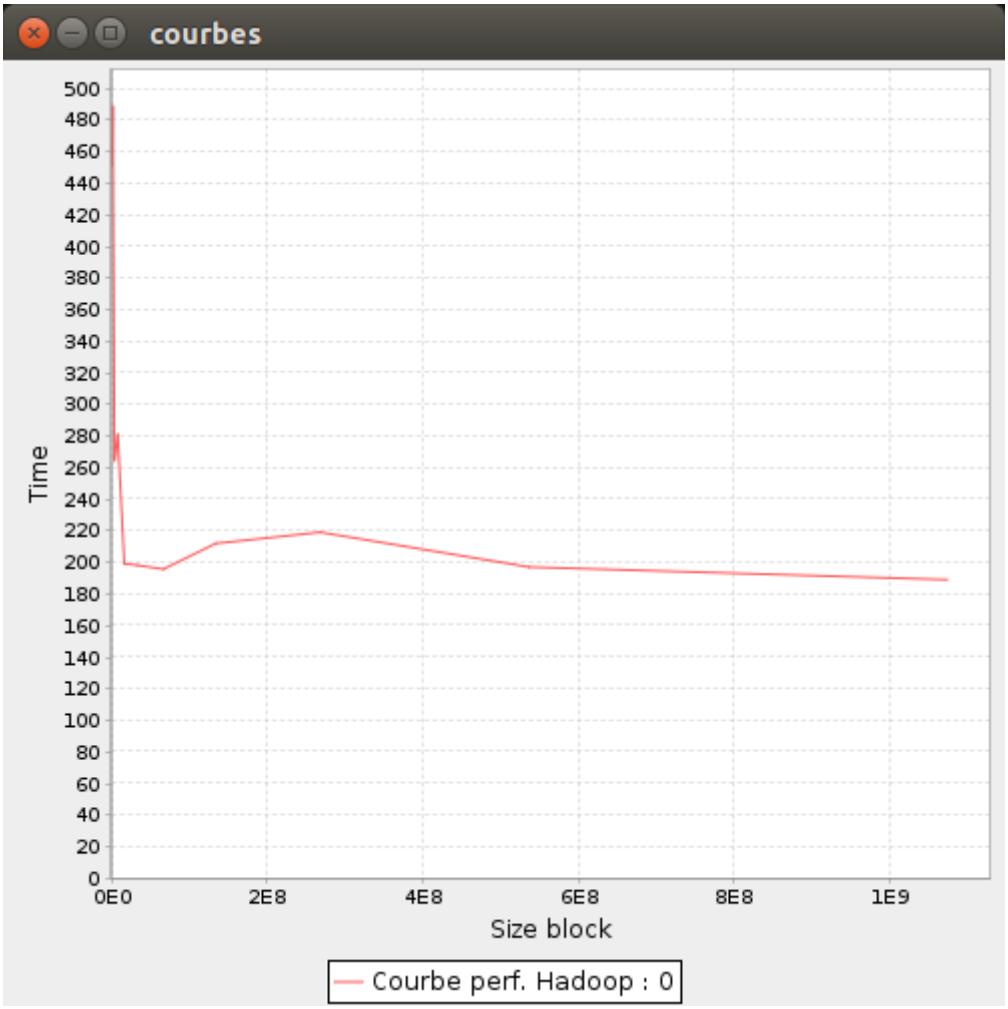
Taille(Octets)	1048 576	2097 152	4194 304	8388 608	1677 7216	3355 4432	6710 8864	13421 7728	26843 5456	53687 0912	10737 41824
Durée (S)	441. 17	329. 94	279. 61	273. 46	242.9 0	255.3 0	252.3 4	214.0 0	209.7 9	222.1 6	204.39



Ici, on remarque que le temps optimum est 204.39 Secondes, obtenu avec une taille de blocs de 1073741824 Octets

h- Résultats de la huitième prise de mesure

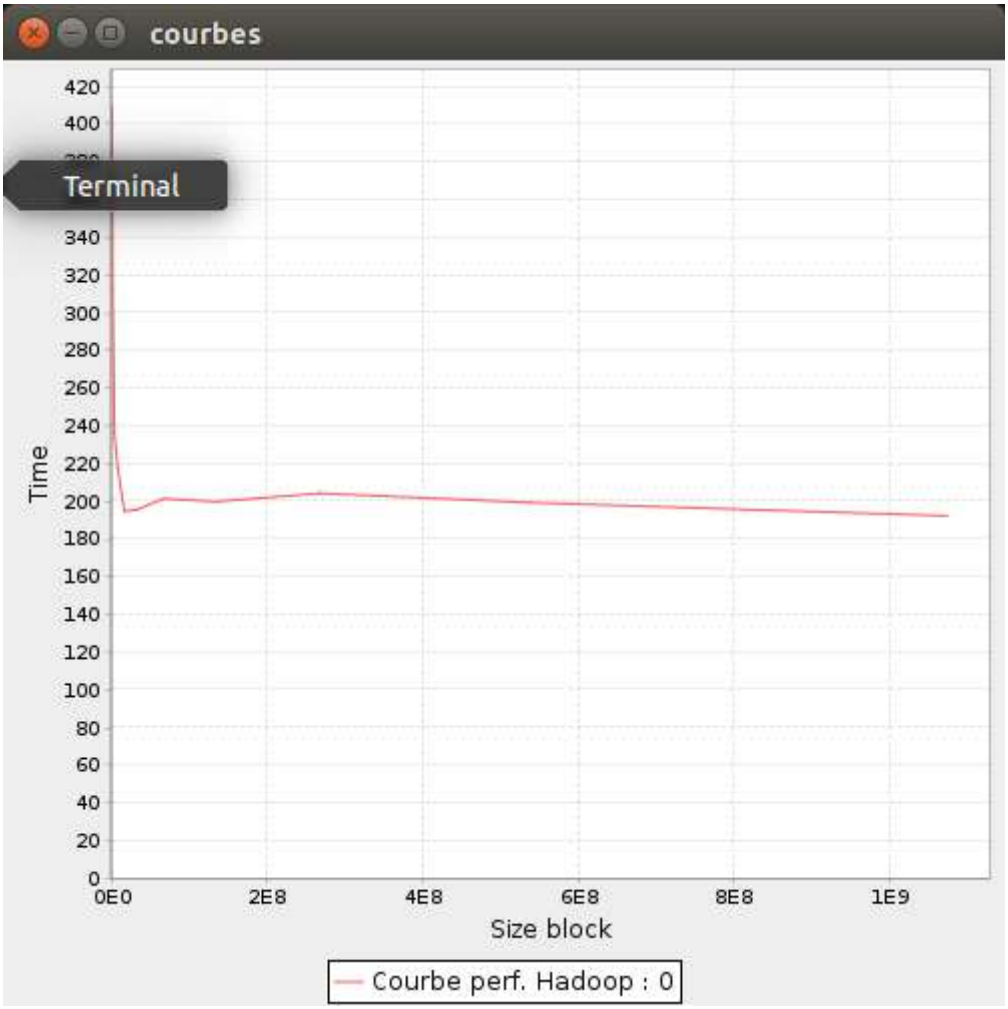
Taille(Octets)	1048 576	2097 152	4194 304	8388 608	1677 7216	3355 4432	6710 8864	13421 7728	26843 5456	53687 0912	10737 41824
Durée (S)	451. 35	488. 55	264. 46	280. 64	198.9 8	197.9 9	195.5 4	211.7 8	218.9 0	196.8 6	188.86



Ici, on remarque que le temps optimum est 188.86 Secondes, obtenu avec une taille de blocs de 1073741824 Octets

i- Résultats de la neuvième prise de mesure

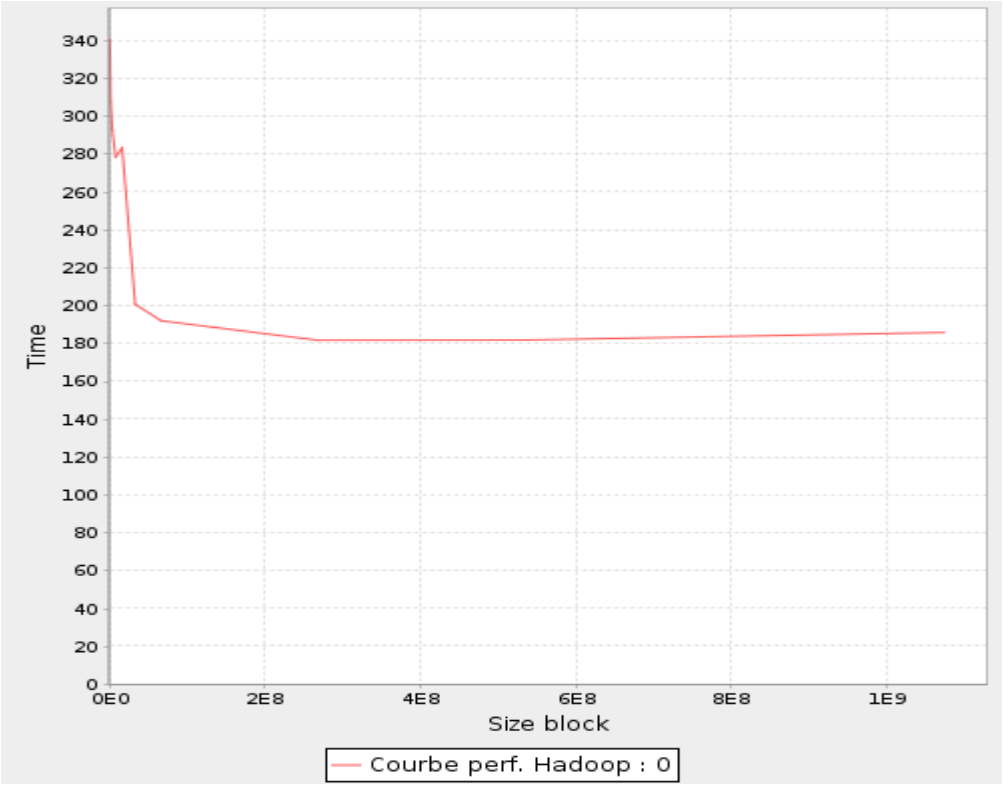
Taille(Octets)	1048 576	2097 152	4194 304	8388 608	1677 7216	3355 4432	6710 8864	13421 7728	26843 5456	53687 0912	10737 41824
Durée (S)	409. 22	312. 45	235. 64	217. 53	194.5 1	195.4 1	201.1 7	199.5 8	204.0 3	199.0 9	192.07



Ici, on remarque que le temps optimum est 192.07 Secondes, obtenu avec une taille de blocs de 1073741824 Octets

j- Résultats de la dixième prise de mesure

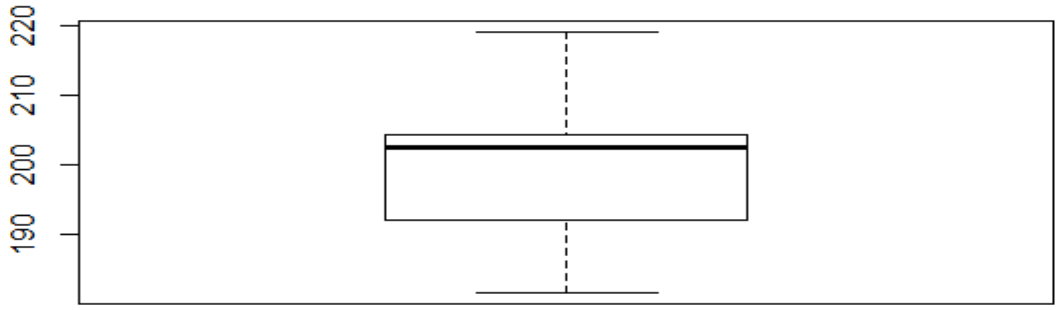
Taille(Octets)	1048 576	2097 152	4194 304	8388 608	1677 7216	3355 4432	6710 8864	13421 7728	26843 5456	53687 0912	10737 41824
Durée (S)	340. 67	307. 70	292. 63	278. 42	283.4 4	200.6 3	191.9 1	188.4 9	181.6 3	181.7 5	185.73



Ici, on remarque que le temps optimum est 181.63 Secondes, obtenu avec une taille de blocs de 268435456 Octets

k- Conclusion

Nous représentons ci-dessous la répartition des différents temps d'exécution obtenus autour de la moyenne :



Le temps optimal global, obtenu comme étant la médiane des différents temps optimaux partiels est 202.39 Secondes, obtenu avec une taille de blocs de 1073741824 Octets.

IV- Solution : cas séquentiel

1- Description de la solution

Nous avons développé en Java une version séquentielle du WordCount, dans le but d'effectuer une comparaison des performances avec les implantations sous Hadoop et Storm. Le code source de cette implantation se retrouve dans le sous dossier Codes_Sources du dossier BigDataSequentiel se trouvant dans le répertoire Realisations_Logicielles. Il comprend en somme une classe « WordCountSequencial » possédant :

- Un constructeur qui prend en entrée un nom de fichier, ouvre un flux en lecture sur ce fichier, appelle la fonction nextTuple de façon séquentielle sur le flux et appelle la fonction execute sur chaque retour de la fonction nextTuple, jusqu'à ce que ce retour soit nul. A la fin, écrit le contenu du compteur dans le fichier de sortie.
- Une fonction « nextTuple » qui renvoi de façon successive chaque ligne du fichier en utilisant le flux en lecture ouvert dans le constructeur ou null si la fin du fichier est atteinte.
- Une fonction « execute » qui prend en entrée une chaine de caractères, la découpe en mots et met à jour le compteur de mots avec le nombre d'occurrences de chaque mot apparaissant dans la chaine de caractères.
- Une fonction « main » qui prend en paramètre une chaine de caractères représentant le nom du fichier texte à traiter et instancie un objet WordCountSequencial avec comme paramètre cette chaine de caractères.

Dans le souci d'effectuer plusieurs simulations de cette application sur un même fichier afin d'obtenir le temps optimum avec un minimum d'erreurs, nous avons mis en place un script bash nommé « simulate-sequentiel.sh » qui se retrouve dans le sous dossier Binaires du dossier BigDataSequentiel se trouvant dans le répertoire Realisations_Logicielles et qui prend en entrée un nombre d'itérations et un nom de fichier et exécute autant de fois le WordCount séquentiel, sur ce fichier, en stockant les différentes durées d'exécution dans un fichier horodaté.

Pour refaire la simulation pour un nombre d'itérations et un fichier texte donnés, il faut télécharger la version complète des livrables (voir Annexe), se rendre dans le sous dossier Binaires du répertoire BigDataSequentiel, contenant wordCountSequencial.jar et simulate-sequentiel.sh, et lancer la simulation en saisissant la commande :

chmod 755 * #pour donner les droits d'exécution sur les scripts
./simulate-sequentiel.sh nbre_occurrences nom_fichier

Où nbre_occurrences, entier positif, représente le nombre d'itérations de l'exécution du WordCount séquentiel sur le fichier de nom nom_fichier.

L'on obtient en sortie un fichier de données donnant les différents temps d'exécution.

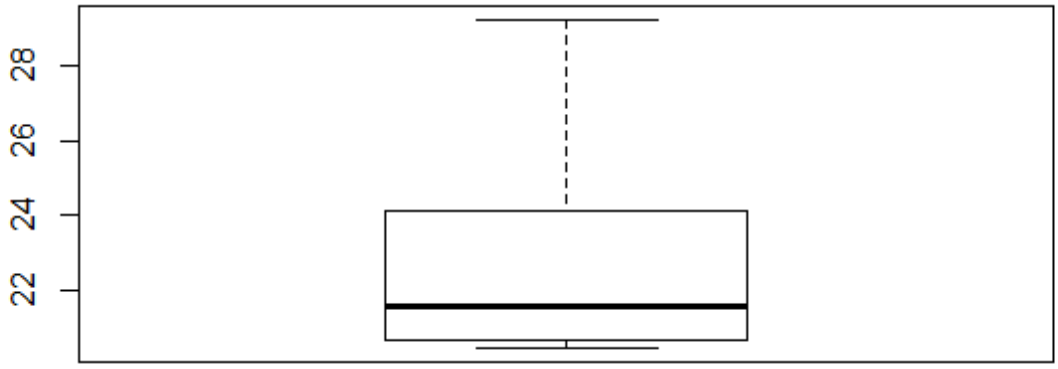
2- Analyse des résultats et conclusion

Nous avons relevé 10 fois les durées d'exécution du WordCount séquentiel sur un fichier de 900 MB, le résultat final est la médiane des résultats obtenus. Les simulations se sont

effectuées sur Linux Ubuntu 14.4 LTE tournant sur un PC core duo 2.56 GHz *2, 2 Go de RAM.

Vous pouvez retrouver ces données dans le sous dossiers Résultats du dossier BigDataSequentiel présent dans le répertoire Realisations_Logicielles.

Durée (S)	27.89	23.07	29.24	24.14	20.55	20.71	21.03	20.69	20.46	22.15



Ci-dessus, la courbe de répartition des durées d'exécution obtenues autour de la moyenne.

La durée d'exécution finale, pour un fichier de 900 MB est donc de 21.59 Secondes, obtenue comme étant la valeur médiane des durées d'exécution réalisées.

V- Solution : cas de Apache Storm

1- Description de la solution

Nous avons développé en Java une version du WordCount utilisant Apache Storm. Le code source de cette implantation se retrouve dans le sous dossier Codes_Sources du dossier BigDataStorm se trouvant dans le répertoire Realisations_Logicielles. Il comprend en somme :

- Une classe, FileSentenceSpout, présente dans le fichier **« FileSentenceSpout.java »** et contenant l'implantation du Spout ayant pour source un fichier
- Une classe, WordSpliterBolt, présente dans le fichier **« WordSpliterBolt.java »** et contenant l'implantation du Bolt Découpeur de phrase
- Une classe, WordCounterBolt, présente dans le fichier **« WordCounterBolt.java »** et contenant l'implantation du Bolt Compteur de mots
- Une classe, GlobalWordCounterBolt, présente dans le fichier **« GlobalWordCounterBolt.java »** et contenant l'implantation du Bolt Compteur

Global en charge de la récupération des résultats des WordCounterBolt, de leur synthétisation et de l'écriture du décompte final dans le fichier de résultats

- Une classe, WordCountTopology, présente dans le fichier **« WordCountTopology.java »,** classe principale en charge du découpage du fichier source selon le nombre de Spouts et de la création de la topologie

Pour résoudre la problématique de lecture simultanée d'un même fichier par plusieurs spouts, nous avons mis en place, dans la classe principale, le découpage du fichier d'entrée en un certain nombre de blocs correspondant, la plus part du temps, au nombre de spouts passé en paramètre. Chaque Spout, à son instanciation, s'occupe de la lecture d'un bloc. A la fin de la lecture de son bloc, il vérifie s'il y'a encore des blocs non lus ; dans l'affirmative, il prend un de ces blocs et effectue la lecture de ce dernier ; sinon, il passe à l'état zoombie.

Lorsque le nombre de Spouts à l'état zoombie est égal au nombre de Spouts passé en paramètre et déclaré au moment de la création de la Topologie du cluster dans la classe principale, tous les spouts envoient, à chaque appel à la méthode NextTuple, une chaine de caractères représentant la fin du fichier.

Chaque Bolt découpeur effectue son travail normalement, c'est-à-dire découper les phrases reçues en mots et envoyer chaque mot à un Bolt Compteur, jusqu'à ce qu'il reçoive la chaine de fin du fichier, auquel cas, il passe à l'état zoombie. Lorsque le nombre de Bolt Découpeurs à l'état zoombie est égal au nombre de Bolts Découpeurs passé en paramètre et déclaré au moment de la création de la Topologie du cluster dans la classe principale, tous les Bolts Découpeurs envoient désormais la chaine de fin de fichier aux Bolts Compteurs.

Dans le souci de faire en sorte que tous les Bolts Compteurs puissent recevoir la chaine de fin de fichier et aussi dans le souci de repartir de façon quasiment équitable la charge à ces derniers, nous avons liés les Bolts Compteurs aux Bolts Découpeurs avec une liaison **shuffleGrouping**, ce qui veut dire que les mots sont envoyés de façon aléatoire des Bolts Découpeurs aux Bolts Compteurs.

Chaque Bolt Compteur effectue son travail normalement, c'est-à-dire compter chaque mot reçu et garder le décompte dans une map, jusqu'à ce qu'il reçoive la chaine de fin du fichier. La réception de celle-ci veut dire que les Spouts ont atteint la fin du fichier et que les Bolts Découpeurs ont tous fini de découper les phrases reçues.

A la réception de cette chaine de fin de fichier par un Bolt Compteur, celui-ci envoi le contenu de son map au Bolt Compteur Global (qui n'existe qu'en une seule instance et qui est lié aux Bolts Compteurs par une liaison GlobalGrouping) et passe à l'état zoombie.

Dès lors que tous les Bolts Compteurs sont à l'état zoombie, ces derniers n'envoient désormais, au Bolt Compteur Global, que la chaine de fin de fichier. La réception de cette dernière par celui-ci veut dire pour lui que le cluster a terminé son travail. Il écrit le contenu de son map dans le fichier de sortie et envoie un signal de fin à la Classe Principale qui se charge d'arrêter le cluster.

Pour que le Bolt Compteur Global puisse envoyer un signal à la Classe principale, nous avons implémenté le pattern Observable/Observer entre ces derniers : La Classe Principale est un Observateur (Observer) pour le Bolt Compteur Global qui est Observable. A l'instanciation

du Bolt Compteur Global, celui-ci ajoute comme observateur la Classe Principale et donc peut notifier cette dernière, en particulier la notifie de la fin de l'exécution du traitement par le cluster afin que celui-ci arrête le cluster.

L'application prend en paramètres :

- Le nom du fichier d'entrée
- Le nom du fichier de sortie
- Un entier qui représente le nombre maximal de groupes de mots qu'un Spout doit envoyer sans recevoir de confirmation de réception et de traitement de la part des Bolts. Cet entier permet de varier le nombre de mots en attente de traitements et ainsi de varier l'écart de temps entre la fin de lecture du fichier par les Spouts et la fin des traitements par les Bolts
- Un entier représentant le nombre de Spouts
- Un entier représentant le nombre de Bolts Découpeurs
- Un entier représentant le nombre de Bolts Compteurs

Dans le souci d'effectuer plusieurs simulations de cette application sur un même fichier, avec des valeurs de paramètres différents, afin d'obtenir le temps et les paramètres optimums avec un minimum d'erreurs, nous avons mis en place un script bash nommé « simulate-storm.sh » qui se retrouve dans le sous dossier Binaires du dossier BigDataStorm se trouvant dans le répertoire Realisations_Logicielles et qui prend en entrée :

- La valeur min, le pas et la valeur max pour le nombre de Spouts
- La valeur min, le pas et la valeur max pour le nombre de Bolts Découpeurs
- La valeur min, le pas et la valeur max pour le nombre de Bolts Compteurs
- Le nom du fichier d'entrée

Appelle, pour chaque série de valeurs de paramètres, le WordCount sous Storm sur ce fichier, en stockant les différentes durées d'exécution dans un fichier horodaté, puis trace les courbes correspondantes donnant les variations du temps et sauvegarde ces dernières dans des fichiers image au format « .png ».

Pour refaire la simulation pour des plages de paramètres et un fichier texte donnés, il faut télécharger la version complète des livrables (voir Annexe), se rendre dans le sous dossier Binaires du répertoire BigDataStorm, contenant launch_storm_perf.jar (servant au tracé des courbes à partie des fichiers de résultats), launch_wordcount_topology.jar (le binaire du WordCount sous java/storm) et simulate-storm.sh, et lancer la simulation en saisissant la commande :

chmod 755 * #pour donner les droits d'exécution sur les scripts
./simulate-storm.sh min_1 pas_1 max_1 min_2 pas_2 max_2 min_3 pas_3 max_3 nom_fichier

Où min_i, pas_i et max_i sont des entiers représentant respectivement la valeur mimale, le pas d'incrément et la valeur maximale pour le paramètre d'ordre i et, nom_fichier, le nom du fichier d'entrée.

L'on obtient en sortie un dossier pour chaque valeur du nombre de spouts, dans ce dossier, un fichier de données et une courbe pour chaque valeur du nombre de Bolts Découpeurs,

qui représentent les variations de la durée d'exécution suivant les valeurs du nombre de Bolts Compteurs. Les fichiers de données sont stockés dans un dossier horodaté avec pour préfixe « doc-» et les courbes dans un dossier horodaté avec pour préfixe « img-». Un récapitulatif des données obtenus présentant pour chaque valeur du nombre de spouts, du nombre de Bolts découpeurs et du nombre de Bolts compteurs, le temps mis, est stocké dans un fichier horodaté avec pour préfixe « results_excel- ».

Remarque : Pour faire fonctionner Storm de façon distribuée, il est nécessaire d'utiliser le service ZOOKEEPER qui est un service centralisé fournissant une synchronisation distribuées lorsque les équipements devant exécuter les différentes tâches sont distribués. Etant donné que nous avons réalisé toutes nos simulations sur un seul PC isolé, nous n'avons pas vu l'intérêt d'utiliser ZOOKEEPER. Néanmoins, le lecteur le désirant trouvera un excellent tutoriel pour l'installation de ZOOKEEPER et la liaison avec apache Storm en suivant le lien http://10jumps.com/blog/storm-installation-single-machine .

2- Analyse des résultats et conclusion

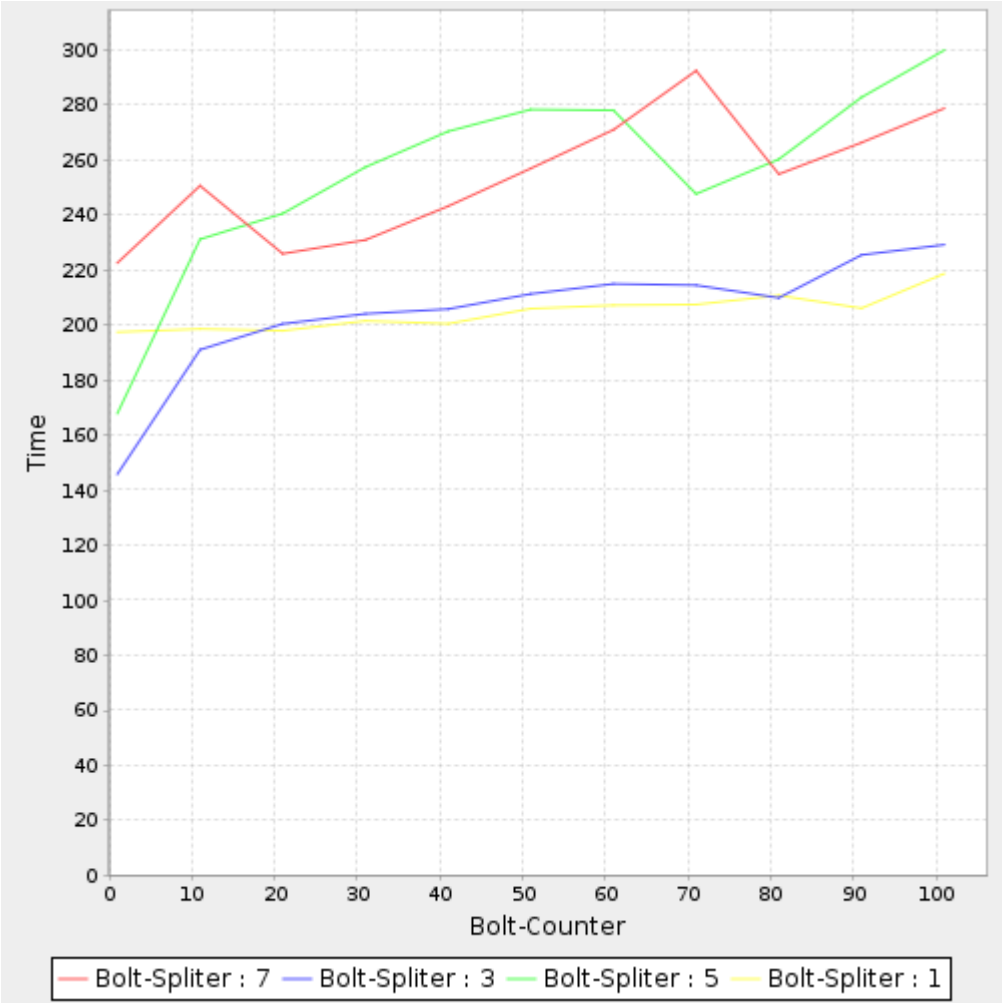
Pour Apache Storm, nous avons fait varier le nombre de Spouts et le nombre de Bolts Découpeurs de 1 à 7 par pas de 2 et le nombre de Bolts Compteurs de mots de 1 à 101 par pas de 10 et nous avons mesuré les temps d'exécution pour un fichier texte de 30 MB. Les simulations se sont effectuées sur Linux Ubuntu 14.4 LTE tournant sur un PC core duo 2.56 GHz *2, 2 Go de RAM.

Vous pouvez retrouver ces données dans le sous dossiers Résultats du dossier BigDataStorm présent dans le répertoire Realisations_Logicielles.

a- Cas de l'utilisation de 1 Spout :

Nombre de SPLITER	Nombre de COUNTER	Temps
1	1	197.40
1	11	198.53
1	21	197.87
1	31	201.46
1	41	200.42
1	51	205.92
1	61	207.14
1	71	207.41
1	81	210.71
1	91	206.07
1	101	218.64
3	1	145.72
3	11	190.97

3	21	200.39
3	31	204.04
3	41	205.76
3	51	211.27
3	61	214.93
3	71	214.44
3	81	209.84
3	91	225.45
3	101	229.15
5	1	167.79
5	11	231.15
5	21	240.52
5	31	257.41
5	41	270.39
5	51	278.37
5	61	278.06
5	71	247.66
5	81	260.28
5	91	282.79
5	101	299.92
7	1	222.55
7	11	250.63
7	21	225.89
7	31	230.90
7	41	243.18
7	51	256.93
7	61	270.99
7	71	292.47
7	81	254.85
7	91	266.32
7	101	278.76

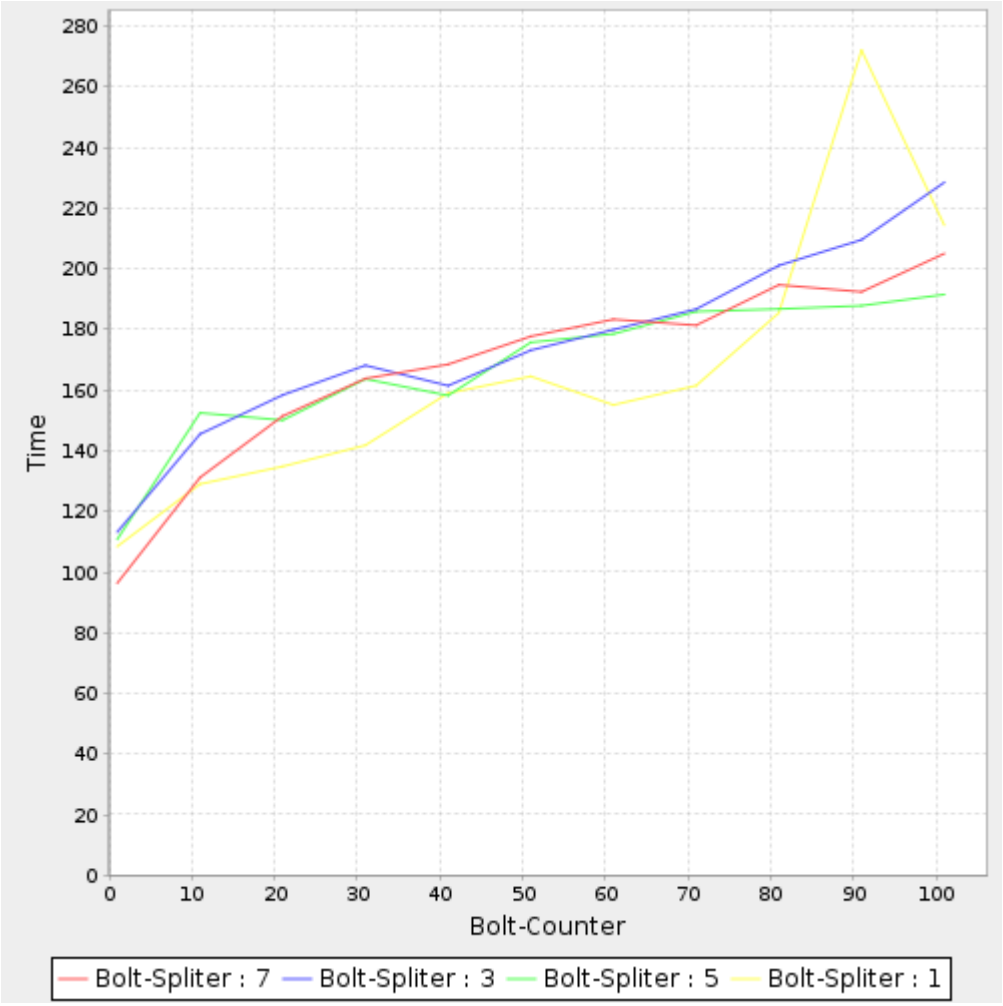


De ce graphe, il ressort que pour ces données, la meilleure configuration correspond à celle qui minimise le temps – dans le cas présent environ 145 s – de traitement. Après lecture graphique, nous obtenons la configuration suivante : 1 SPOUT – 3 BOLT SPLITER – 1 BOLT COUNTER.

b- Cas de l'utilisation de 3 Spouts :

Nombre de SPLITER	Nombre de COUNTER	Temps
1	1	108.35
1	11	128.90
1	21	134.71
1	31	141.71
1	41	158.88
1	51	164.43
1	61	155.00
1	71	161.39
1	81	185.38

1	91	272.05
1	101	214.37
3	1	113.12
3	11	145.40
3	21	158.29
3	31	168.05
3	41	161.36
3	51	173.04
3	61	179.86
3	71	186.62
3	81	201.02
3	91	209.51
3	101	228.42
5	1	110.72
5	11	152.39
5	21	149.98
5	31	163.58
5	41	158.12
5	51	175.65
5	61	178.44
5	71	185.82
5	81	186.67
5	91	187.76
5	101	191.44
7	1	393.82
7	11	131.07
7	21	151.34
7	31	163.71
7	41	168.39
7	51	177.63
7	61	183.25
7	71	181.29
7	81	194.63
7	91	192.33
7	101	204.94

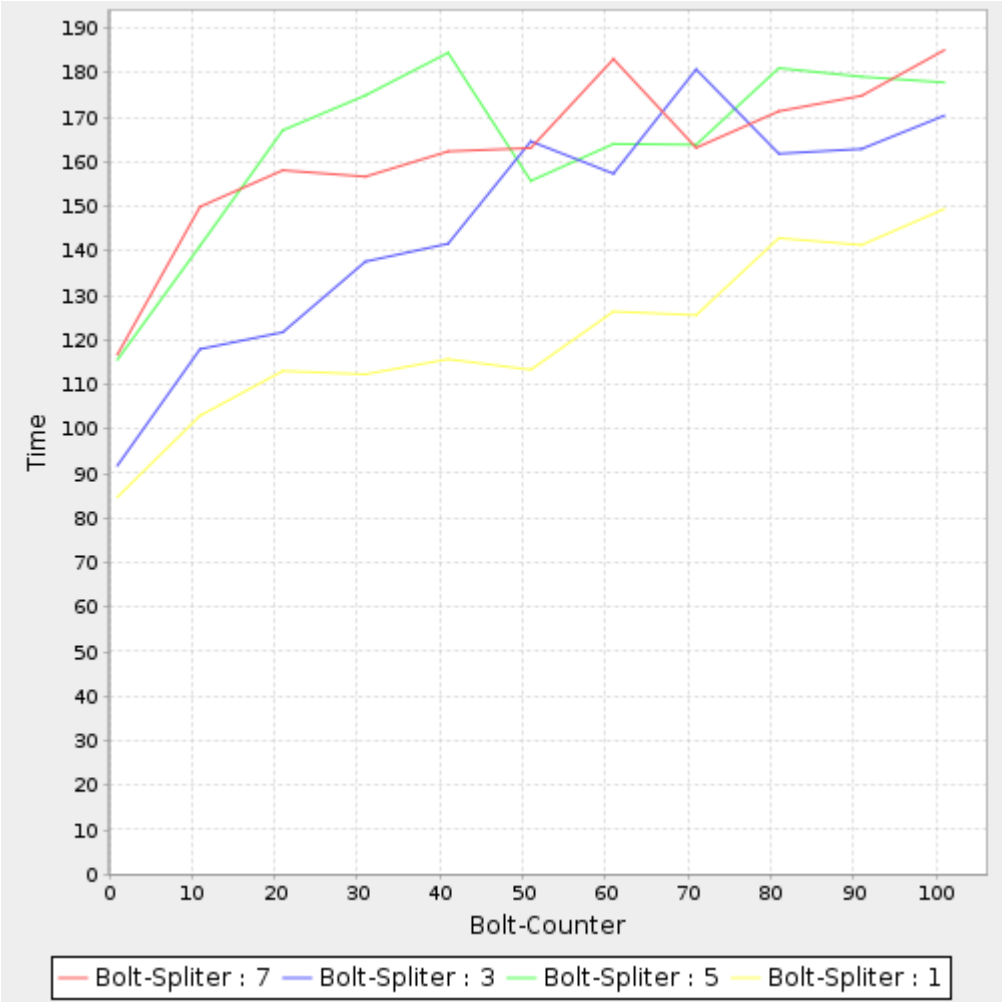


Nous constatons que le temps croit avec le nombre de Bolt counter. La meilleure configuration converge vous celle où l'on retrouve 3 SPOUTS, 1 SPLITERS et 11 BOLTS COUNTER.

c- Cas de l'utilisation de 5 Spouts :

Nombre de SPLITER	Nombre de COUNTER	Temps
1	1	84.65
1	11	102.96
1	21	113.00
1	31	112.26
1	41	115.65
1	51	113.28
1	61	126.37
1	71	125.55
1	81	142.86

1	91	141.31
1	101	149.39
3	1	91.73
3	11	117.91
3	21	121.70
3	31	137.58
3	41	141.60
3	51	164.60
3	61	157.35
3	71	180.77
3	81	161.82
3	91	162.89
3	101	170.37
5	1	115.52
5	11	141.13
5	21	167.08
5	31	174.92
5	41	184.50
5	51	155.72
5	61	164.02
5	71	163.84
5	81	181.05
5	91	179.12
5	101	177.83
7	1	116.67
7	11	149.91
7	21	158.09
7	31	156.67
7	41	162.34
7	51	163.09
7	61	183.08
7	71	163.16
7	81	171.36
7	91	174.83
7	101	185.08

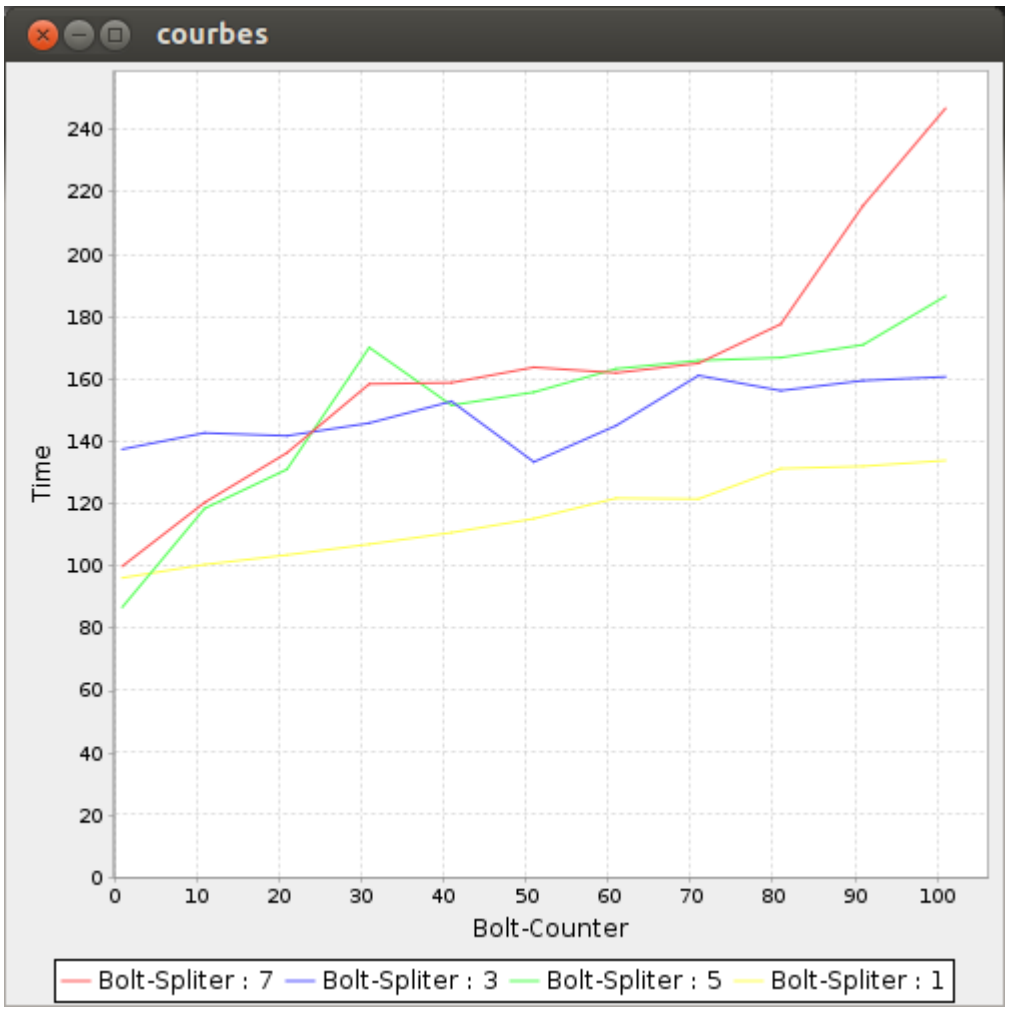


Il est visible sur ce graphe que lorsque le nombre de BOLT-COUNTER augmente, le temps augmente ; aussi, la meilleure configuration dans ce cas est celle où l'on retrouve 5 SPOUTS, 1 BOLTS SPLITER et 1 ou 10 BOLTS COUNTER.

d- Cas de l'utilisation de 7 Spouts :

Nombre de SPLITER	Nombre de COUNTER	Temps
1	1	96.05
1	11	100.32
1	21	103.40
1	31	106.85
1	41	110.58
1	51	115.07
1	61	121.64
1	71	121.39
1	81	131.15

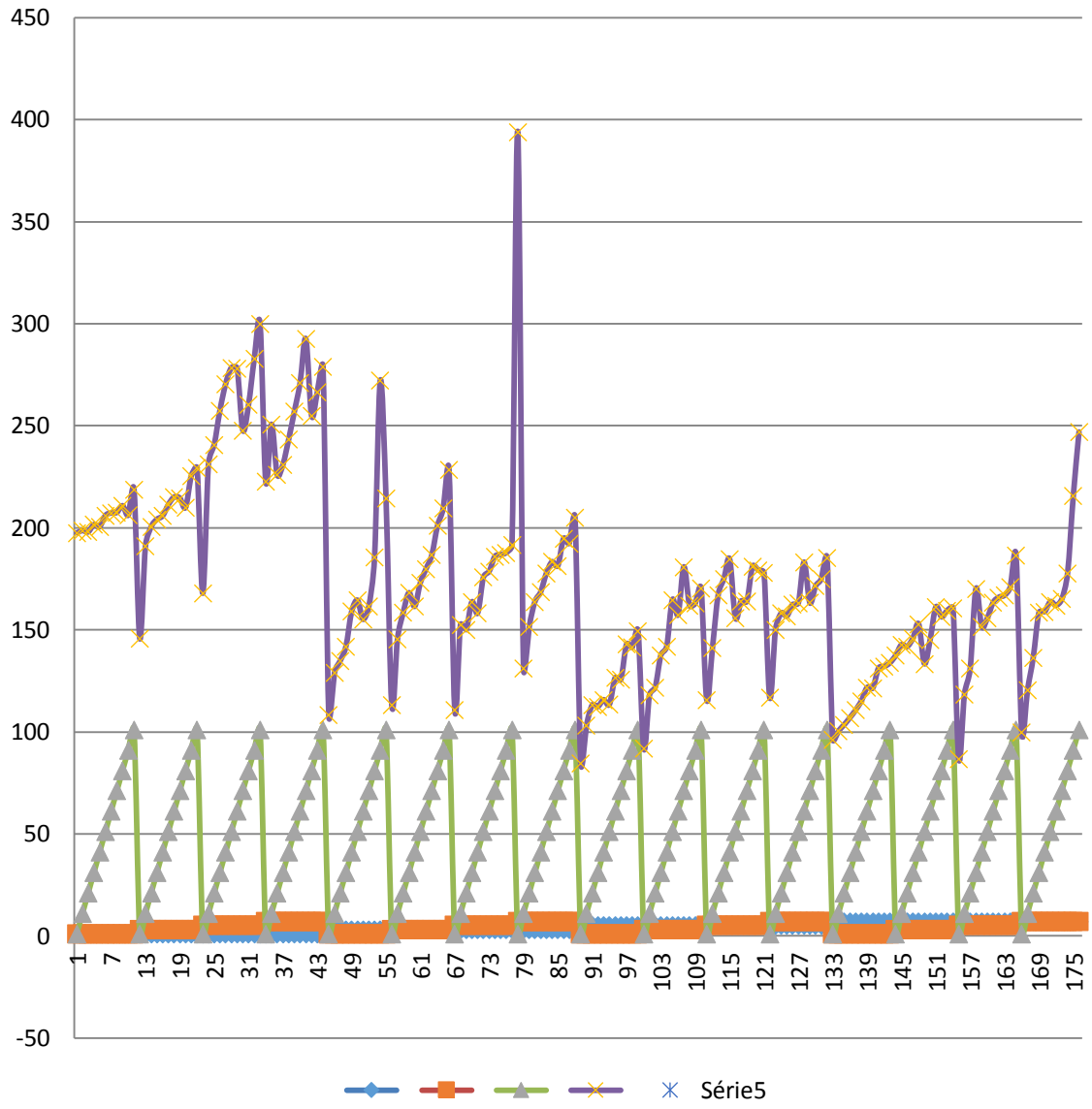
1	91	131.90
1	101	133.76
3	1	137.38
3	11	142.61
3	21	141.68
3	31	145.78
3	41	152.74
3	51	133.30
3	61	144.97
3	71	161.06
3	81	156.20
3	91	159.42
3	101	160.64
5	1	86.65
5	11	118.34
5	21	130.91
5	31	170.06
5	41	151.50
5	51	155.70
5	61	163.23
5	71	165.86
5	81	166.81
5	91	170.93
5	101	186.54
7	1	99.80
7	11	120.26
7	21	136.21
7	31	158.35
7	41	158.75
7	51	163.70
7	61	161.91
7	71	164.98
7	81	177.57
7	91	215.63
7	101	246.84



On remarque à travers ce graphe que plus on augmente le nombre de Bolt counter, plus le temps augmente. Après lecture graphique, nous déduisons que la configuration optimale ici est celle où l'on a 7 SPOUTS, 1 BOLTS SPLITERS et 1 OU 11 BOLTS COUNTERS.

e- Conclusion

Pour résumer, nous avons tracé la courbe globale regroupant tous les données précédentes et nous avons obtenus le graphe suivant :



La courbe en violette est celle de la variation du temps. Celle en verte est celle de la variation du nombre de BOLT COUNTE, celle en rouge est la courbe de la variation de BOLT SPLITER et la courbe en bleu est celle de la variation des SPOUT.

Nous constatons que le temps minimum vaut **84.65 secondes** et est atteint lorsqu'on a la configuration suivante : 5 SPOUT - 1 SPLITER - 90 COUNTER.

VI- Résumé et perspectives

Nous avons effectué une comparaison entre les trois implantations de l'application de comptage de mots : Hadoop, Séquentiel et Apache Storm. Il s'agissait plus précisément, dans le cas de Hadoop et Apache Storm, de prendre en entrée un grand fichier et de délivrer en sortie une liste de tuples « mot, comptage », et de modifier les paramètres de telle sorte qu'on obtienne les performances maximales.

Il ressort de ces différentes analyses qu'en exécution sur un même PC en local, dans le cas d'un grand fichier, l'implantation séquentielle du WordCount se révèle plus intéressante que l'implantation utilisant Hadoop qui, à son tour, se révèle beaucoup plus intéressante que l'implantation utilisant Storm. N'ayant pas eu l'occasion d'effectuer les simulations en environnement distribué, nous ne pouvons pas nous prononcer de ce côté-là.

Webographie, Bibliographie et Références

- [1] http://www.futura-sciences.com/magazines/high-tech/infos/dico/d/informatique-cloud-computing-11573/
- The Docker Book by James Turnbull, 16/08/2015, version 1.8.0
- Getting Started With Storm by Jonathan Leibiusky, Gabriel Eisbruch & Dario Simonassi, 1276
- Storm, Distributed and fault-tolerant realtime computation by Nathan Marz, Twitter
- Installation de Apache STORM by Philippe Lacomme et Raksmey Phan, version 1.2 du 15/092015
- Introduction à Apache STORM, Premier progamme… by Philippe Lacomme, version 1.0 du 23/08/2015
- [2] https://www-01.ibm.com/software/fr/data/bigdata/ du 22/11/2015
- http://10jumps.com/blog/storm-installation-single-machine du 22/11/2015

Annexe

Nous allons présenter dans cette annexe la procédure à suivre pour récupérer la version complète des livrables sur https://github.com/stuenofotso/Cloud-Computing-Project-Ressources .

1- Si vous possédez une installation de git, il suffit d'ouvrir la ligne de commande et de saisir ***git https://github.com/stuenofotso/Cloud-Computing-Project-Ressources.git*** ; cette commande clonera le dépôt dans le répertoire courant.
2- Vous pouvez vous rendre sur la page correspondante à ce lien, cliquez sur le bouton « Download ZIP » ; vous pourrez de ce fait télécharger une sauvegarde du dépôt compressée au format « .zip ».