République Algérienne Démocratique et Populaire

Université Abou Bakr Belkaid– Tlemcen

Faculté des Sciences

Département d'Informatique

Mémoire de fin d'études

Pour l'obtention du diplôme de Master en Informatique

Option: *Système d'Information et de Connaissances (S.I.C)*

Thème

Etude comparative des bases de données NoSQL

Réalisé par :

- **AMARA Manel Warda**

Présenté le 21 Juin 2015 devant le jury composé de MM.

- **EL YEBDRI Zeyneb** *(Présidente)*
- **MATALLAH Houcine** *(Encadreur)*
- **MAHFOUD Houari** *(Examinateur)*
- **BEKARRA Chakib** *(Examinateur)*

Année universitaire : 2014-2015

Remerciements

Mes remerciements vont en premier lieu à ALLAH Aza wa jel le tout puissant de m'avoir donné la foi et de m'avoir permis d'en arriver là.

Je tiens à remercier particulièrement notre encadreur Mr MATALLAH Houcine pour son assistance permanente, pour ces conseils et son suivi durant la réalisation de ce projet.

J'exprime ma gratitude, mes remerciements à mon cher père Zinne dinne qui a fait de son mieux de m'avoir aidé par ses conseils

Je remercie très sincèrement, les membres de jury d'avoir bien voulu accepter de faire partie de la commission d'examination de ce sujet.

Je tien à remercier
Tous mes enseignants, qui ont contribué à notre formation tout au long de nos années d'études.

Spécial remerciements à Tounkara Ganda qui m'a accompagné durant la réalisation de ce projet.

Je tiens à remercier ma sœur et mon amie Belghitri Krima et toutes mes amies de la promotion.

Dédicaces

Je commence par rendre grâce à dieu et à sa bonté, pour la patience, la compétence et le courage qu'il m'a donné pour arriver à ce stade d'étude.

Je dédie ce modeste travail et ma profonde gratitude à celle qui m'a transmis la vie, l'amour, le courage, à toi chère maman que dieu t'accueillera dans son vaste Paradis, toutes mes joies, mon amour et ma reconnaissance, à mon très cher père.

A mes sœurs Liala , Imene, Aya Lina, à mon frère Houssem Dinne, Zinne Dinne et Zinne Abidine ainsi qu'à toute ma grande famille Amara et Benhaila.

A mes amis et frères (Karima, Ganda, Hadjira .H) et tous mes amis du SIC, MID et RSD

Je n'oublierai pas de dédier ce travail à mes précieux amis sur Facebook et à toute personne qui me connaisse et me considère comme une amie.

Amara Manel Warda

Table des matières

Introduction générale

Chapitre I : SGBDs relationnels

Chapitre II : Le NoSQL

Chapitre III : HADOOP, HBASE, MONGODB ET YSCB

Chapitre IV : Etude comparative

Contexte

Durant les trois dernières décennies, les systèmes de gestion de bases de données relationnelles régnaient sur le marché des bases de données. Cependant, ces dernières années, le Web a connu une révolution démesurée, avec l'avènement des sites web à fort trafic tels que Facebook, Google, Amazon, LinkedIn et autres. Ces grands acteurs du web ont été rapidement limités par les dits systèmes pour deux raisons majeures à savoir la volumétrie et la montée en charge.

N'ayant pas trouvé de solution sur le marché répondant à ces problèmes, ils décidèrent de développer chacun à l'interne leurs propres SGBD. Ces produits développés de manières distinctes, ont apparu très rapidement sous le nom de base de données NoSQL ou de base de données non relationnelle.

Les bases de données NoSQL, signifiant « **N**ot **o**nly **SQL** » ont commencé à émerger depuis 2009, pour répondre aux nouveaux besoins en performance lors des traitements des gros volumes de données. NoSQL ne vient pas remplacer les BD relationnelles mais proposer une alternative ou compléter les fonctionnalités des SGBDR pour donner des solutions plus performantes dans leurs contextes d'exploitation.

Problématique

Avec l'augmentation exponentielle des données, plusieurs utilisateurs des systèmes de bases de données relationnelles veulent migrer vers les bases de données NoSQL. Alors une question très pertinente s'impose : Quel est le meilleur système de base de données parmi ces nombreuses solutions NoSQL disponibles ?

Les utilisateurs de SGBD relationnels savent la puissance des bases NoSQL mais ils ignorent sans doute comment faire un bon choix. Ce choix de la solution à adopter est une vraie problématique à cause du manque des éléments d'informations sur ce sujet.

Contribution

Vu l'intérêt du sujet et la problématique posée, une étude comparative entre les bases NoSQL, s'avère très importante pour orienter les utilisateurs vers les bonnes solutions selon leurs besoins.

Dans cette optique et dans le cadre de notre PFE, nous allons développer une étude comparative entre deux modèles NoSQL très réussis dans le domaine, afin de fournir un ensemble de critères et indicateurs, aux acteurs intéressés, pour des prises de décisions éventuelles sur les solutions adoptées pour leurs entreprises.

Plan du travail

Ce mémoire est structuré comme suit :

- Le premier chapitre de ce manuscrit, est consacré à la présentation les concepts et les limites des bases de données relationnelles.

- Dans le deuxième chapitre, on va présenter les concepts de base de cette mouvance NoSQL ainsi que les différents modèles de bases de données NoSQL.

- Dans le troisième chapitre, on va étaler sur les bases de données et les outils sur lesquels l'étude va être menée, à savoir : Hadoop, HBASE, MongoDB et YCSB.

- Le quatrième chapitre fera l'objet des différents résultats expérimentaux obtenus après exécution d'un ensemble de tests ainsi qu'une interprétation des résultats d'évaluation des performances de chaque base de données.

- Enfin, nous clôturons notre manuscrit par une conclusion générale et quelques perspective

I. Introduction

Les systèmes de gestion de bases de données (SGBD) relationnels ont pris une place importante sur le marché des SGBD. Ils répondent de façon satisfaisante aux besoins des applications classiques de gestion (administration, comptabilité, banques, etc.).

Ce modèle bien que très puissant, a atteint ses limites pour les environnements distribués requis par les environnements de grandes échelles et par des trafics très élevés générés par les grands acteurs d'Internet tels que Google, Facebook, Yahoo…etc.

II. Définitions et concepts de base

II.1 Qu'est-ce qu'une base de données ?

Une base de données (BD) représente l'ensemble (cohérent, intégré, partagé) des informations nécessaires au fonctionnement d'une entreprise mémorisées sur un support permanent, dont la gestion est assurée par un logiciel appelé système de gestion de bases de données. [2]

II.2 Qu'est-ce qu'un SGBD ?

Un Système de Gestion de Base de Données peut être défini comme un ensemble de logiciels prenant en charge la structuration, le stockage, la mise à jour et la maintenance des données. Autrement dit, il permet de décrire, modifier, interroger et administrer les données. C'est, en fait, l'interface entre la base de données et les utilisateurs (qui ne sont pas forcément informaticiens). [2]

III. Les Bases de Données Relationnels

III.1 Propriétés d'un SGBD relationnel

A partir des objectifs cités ci-dessus, découlent les propriétés fondamentales d'un SGBDR :

- Base formelle : reposant sur des principes parfaitement définis.

- Organisation structurée des données dans des tables interconnectées (d'où le qualificatif relationnel), pour pouvoir détecter les dépendances et redondances des informations.
- Implémentation d'un langage relationnel ensembliste permettant à l'utilisateur de décrire aisément les interrogations et manipulation qu'il souhaite effectuer sur les données.
- Indépendance des données vis-à-vis des programmes applicatifs (dissociation entre la partie "stockage de données" et la partie "gestion" ou "manipulation"
- Gestion des opérations concurrentes pour permettre un accès multi-utilisateur sans conflit.
- Gestion de l'intégrité des données, de leur protection contre les pannes et les accès illicites. [2]

III.2 Principes fondamentaux du modèle relationnel

- **Table (ou Relation) et Attribut** : Une table est un tableau avec des colonnes et des lignes. Une table peut en contenir un nombre quelconque et leur ordre est indifférent. On trouve encore le vocable d'enregistrement qui donc un ensemble de valeurs, chacun renseignant un champ. [1]
- **Clé :** Une clé d'identification ou clé d'une table est un attribut ou une combinaison d'attributs qui satisfait : une contrainte d'unicité ou une contrainte minimale. [1]
- **Attribut** : Un attribut est un nom attribué à une colonne d'une relation. [1]
- **Domaine** : Un domaine est ensemble de valeurs que peut prendre un ou plusieurs attributs. [1]
- **Tuple** : Un tuple est une ligne d'une relation.
- **Degré :** Degré d'une relation est le nombre de ses attributs.
- **Schéma de relation :** Le schéma d'une relation R est défini par le nom de la relation et la liste des attributs avec pour chaque attribut son domaine.
- **Cardinalité :** La cardinalité d'une relation est le nombre de tuples de la relation.
- **BD relationnelle** : Une base de données relationnelle est une collection de relations normalisées.
- **Les contraintes d'intégrité** : Le modèle relationnel comporte un certain nombre de règles permettant de garantir la cohérence des données.

III.3 Les langages de manipulation de données relationnels

Le modèle relationnel a été à l'origine proposé avec deux langages de manipulation de données (LMD), l'algèbre relationnelle et le calcul des tuples, équivalents en puissance qui ont fixé l'ensemble des fonctions que tout LMD relationnel doit offrir.

III.3.1 L'algèbre relationnelle

L'algèbre relationnelle est un ensemble d'opérateurs qui, à partir d'une ou deux relations existantes, créent en résultat une nouvelle relation temporaire. La relation résultat a exactement les mêmes caractéristiques qu'une relation de la base de donnée et peut donc être manipulée de nouveau par les opérateurs de l'algèbre. Formellement l'algèbre relationnelle comprend 5 opérateurs de base : sélection, projection, union, différence et produit. [3].

III.3.2 Le calcul des tuples (ou calcul relationnel)

Le calcul des tuples est constitué des langages issus du calcul des prédicats de la logique du premier ordre. Deux adaptations de ce calcul au modèle relationnel ont été proposées, qui toutes deux ont conduit à des langages utilisateurs :

- Le calcul des tuples : qui a donné naissance au LMD QUEL du SGBD relationnel.
- Le calcul des domaines : qui a donné naissance à des LMD de type graphique comme QBE, proposé par IBM. [1]

L'algèbre relationnelle et le calcul relationnel sont équivalents dans la mesure où ils ont la même puissance d'expression, c.-à-d. toute expression de l'algèbre relationnelle possède une expression du calcul relationnel qui lui est équivalente.

IV. Le langage SQL

SQL est le LMD relationnel le plus répandu du fait qu'il est la seule norme existante pour les SGBD relationnels. Le langage "Structured Query Language "(SQL) est un langage déclaratif, non procédural dans sa formulation, masquant assez bien le caractère algébrique des expressions. Il a une syntaxe relativement simple.

SQL présente trois facettes fortement intégrées peu importe le SGBD relationnel :

- LID (Langage d'Interrogation des Données) : pour la définition des requêtes d'interrogation sur les données contenues dans la base.
- LDD (Langage de Définition de Données) : pour la définition des tables, des contraintes diverses et des vues relationnelles stockées dans le dictionnaire du SGBD.
- LMD (Langage de Manipulation de Données) : pour la manipulation des tables et plus précisément les manipulations des tuples de relations.
- LCD (Langage de Contrôle de Données) : pour gérer la définition physique des accès (index), la spécification des fichiers physiques et la validation des opérations exécutées dans un contexte multiposte. [1]

V. Les propriétés ACID

Une base de données relationnelle est construite sur les propriétés ACID (atomicité, cohérence, isolation, durabilité) qui sont un ensemble de propriétés garantissant la fiabilité d'une transaction informatique :

- **Atomicité :** L'ensemble des opérations d'une transaction sont réalisées ou toutes sont abandonnées pour pouvoir revenir à l'état initial, qui veut dire que les mises à jour de la BD doivent être atomiques, binaires : réalisées ou pas.

  Exemple **:** sur 5000 lignes devant être modifiées au sein d'une même transaction, s la modification d'une seule échoue, alors la transaction entière doit être annulée.

- **Consistance :** Les modifications apportées à la base doivent être valides, en accord avec l'ensemble de la base et de ses contraintes d'intégrité.

  Si un changement enfreint l'intégrité des données, alors soit le système doit modifier les données dépendantes, comme dans le cas classique d'une suppression en cascade, soit la transaction doit être interdite.

- **Isolation :** Les transactions lancées au même moment ne doivent jamais interférer entre elles. Si une requête est lancée alors qu'une transaction est en cours, le résultat de celle- ci ne peut montrer que l'état original ou final d'une donnée, mais pas l'état intermédiaire.

- **Durabilité :** Toutes les transactions sont lancées de manière définitive. Une base ne doit pas afficher le succès d'une transaction, pour ensuite remettre les données modifiées dans leur état initial. [6]

VI. Limites de SGBD Relationnels

Le modèle relationnel est fondé sur un modèle mathématique solide s'appuyant sur la logique des prédicats du premier ordre. Il s'appuie sur des concepts simples qui font sa force en même temps que sa faiblesse. Nous expliquerons ici les principales limites des SGBDR.

VI.1 Modélisation des entités réelles

Les relations ne correspondent pas toujours à des entités du monde réel (notamment pour les objets complexes). Les processus de normalisation et de décomposition des schémas relationnels entraînent la multiplication des relations, et par conséquent des opérations de jointure lors du traitement des requêtes. [7]

VI.2 Surcharge sémantique

Le modèle relationnel s'appuie sur un seul concept (la relation) pour modéliser à la fois les entités et les associations / relations entre ces entités. Il existe donc un décalage entre la réalité et sa représentation abstraite. [7]

VI.3 Contraintes d'intégrité

Les SGBDs relationnels sont limités à des contrôles de cohérence simples. Il est donc difficile de modéliser des contraintes réelles correspondant aux données d'une entreprise. Ce problème n'est que partiellement résolu avec SQL-2 qui permet d'exprimer des contraintes dans la partie langage de définition. [7]

VI.4 Langage de manipulation

Il est limité aux opérateurs de base du langage SQL, qu'il n'est pas possible d'étendre à de nouveaux opérateurs.

VI.5 Scalabilité limitée

Atteinte par les groupes comme Yahoo, Google, Face book...etc. Cette limite est la plus gênante pour les entreprises. Dans le cas de traitements de données en masse, le constat est simple : les SGBD relationnels ne sont pas adaptés aux environnements distribués requis par les volumes gigantesques de données et par les trafics tout aussi gigantesques générés par ces opérateurs. [4]

VI.6 Des propriétés ACID en milieu distribué

Les propriétés ACID, bien que nécessaires à la logique du relationnel, nuisent fortement aux performances surtout la propriété de cohérence.

En effet, la cohérence est très difficile à mettre en place dans le cadre de plusieurs serveurs (environnement distribué), car pour que celle-ci, pour pouvoir satisfaire cette cohérence, il faut que tous les serveurs doivent être des miroirs les uns des autres, de ce fait deux problèmes apparaissent :

- Le coût en stockage est énorme car chaque donnée est présente sur chaque serveur.
- Le coût d'insertion/modification/suppression est très grand. [4]

VI.7 Les limites face aux usages

Les structures de données manipulées par les systèmes sont devenues de plus en plus complexes avec en contrepartie des moteurs de stockage peu évoluant. Le principal point faible des modèles relationnels est l'absence de gestion d'objets hétérogènes ainsi que le besoin de déclarer au préalable l'ensemble des champs représentant un objet. [5]

VII. Conclusion

Les Systèmes de Gestion de Bases de Données relationnels (SGBDR) sont devenus le noyau de tout système informatique et est très majoritairement répandue pour la gestion des données, cependant, la diversification des applications des bases de données a dévoilé les limites des SGBDR notamment sur le plan de modélisation des données imprécises et de l'interrogation flexible.

I. Introduction

Les bases de données NoSQL, signifiant « **N**ot **o**nly **SQL** » ont commencé à émerger depuis 2009, pour répondre aux nouveaux besoins en performance lors de traitement de gros volumes de données.

Le NoSQL ne vient pas remplacer les BD relationnelles mais proposer une alternative ou compléter les fonctionnalités des SGBDs relationnels pour donner des solutions plus intéressantes dans certains contextes.

II. Big Data

Le terme «Big Data » est apparu il y'a quelques années pour expliquer un domaine issu d'une révolution dans la manière de traitement des données.

Big Data voudrait dire « grosses données ». Or, la problématique des « grosses données », ou données ayant un volume important, n'est pas nouvelle. Depuis plus de 30 ans, nous sommes confrontés à des volumes importants de données. Cela fait presque dix ans que la problématique de gestion des gros volumes de données se pose dans les métiers de la finance, de l'indexation Web et de la recherche scientifique. Le Big Data est défini donc par rapport à la manière avec laquelle les grandes masses de données peuvent être traitées et exploitées de façon optimale. Le concept de Big Data se caractérise par plusieurs aspects. De nombreux responsables informatiques et autorités du secteur tendent à définir le Big Data selon trois grandes caractéristiques : Volume, Vitesse et Variété, soit les trois « V ». [11]

a. **Volume :** Le Big Data est associé à un volume de données vertigineux, se situant actuellement entre quelques dizaines de téraoctets (1 téraoctet=212 octets) et plusieurs péta-octets (1 péta-octets = 215 octets) en un seul jeu de données. Les entreprises, tous secteurs d'activité confondus, devront trouver des moyens pour gérer le volume de données en constante augmentation qui est créé quotidiennement.

b. **Vitesse :** La vitesse décrit la fréquence à laquelle les données sont générées, capturées et partagées. Les entreprises doivent appréhender la vitesse non

seulement en termes de création de données, mais aussi sur le plan de leur traitement, de leur analyse et de leur restitution à l'utilisateur en respectant les exigences des applications en temps réel.

c. **Variété :** La croissance de la variété des données est largement la conséquence des nouvelles données multi structurelles et de l'expansion des types de données. Aujourd'hui, on trouve des capteurs d'informations aussi bien dans les trains, les automobiles ou les avions, ajoutant à cette variété.

d. **Variabilité :** Le format et le sens des données peuvent varier au fil du temps.

On ne peut pas parler de Big Data sans citer le NoSQL, Not Only SQL. Il est venu pour solutionner les difficultés rencontrées pendant la gestion des données classées « Big Data » avec les systèmes SGBD relationnelles.

III. Problématique

Suite à des besoins de haute disponibilité, d'accès, de calculs des volumes impressionnants de données pour des grandes compagnies telles que Google, Amazon, Facebook, Twitter, LinkedIn , les solutions SQL offertes dans le marché, ne répondaient pas aux besoins, plus car leur limites ont été atteintes.

Les limites :

- Volumétrie.
- Difficulté de la mise à jour sans cesse de serveur de plus en plus puissant.
- Analyser des quantités de données faramineuses (Peta octets).
- Tolérance aux pannes (serveur maître tombe)… etc.

D'où l'apparition de solutions basées sur des systèmes distribués qui sont les SGBDs NoSQL. [11]

IV. Historique du mouvement NoSQL

En 1998, le monde entend pour la première fois le terme NoSQL. Terme inventé et employé par Carlo Strozzi pour nommer son SGBD relationnel Open Source léger qui n'utilisait pas le langage SQL. Ironiquement, le trouvail de M. Strozzi n'a rien à voir avec la mouvance NoSQL que l'on connaît aujourd'hui, vu que son SGBD est de type relationnel. En effet, c'est en 2009, lors d'un rassemblement de la communauté des développeurs des SGBD non-relationnels, que le terme NoSQL a été mis au goût du jour pour englober tous les SGBD de type non-relationnel. [8]

V. Définition

Le NoSQL est un type de base de données, c'est une manière de stocker et de récupérer des données de façon rapide, un peu comme une base de données relationnelle, sauf qu'il n'est pas basé sur des relations mathématiques entre les tables comme dans une base de données relationnelle traditionnelle. [4]

VI. Comment le NoSQL fonctionne ?

VI.1 Scalabilité

Scalabilité est l'aptitude d'un système à conserver, maintenir son niveau de performance par augmentation des ressources matérielles

On distingue deux types de scalabilité :

- Verticale ou interne : ajout de RAM, processeur au sein d'une machine ou remplacement par une de plus grand gabarit.

- Horizontale ou externe : Le principe consiste à simplement rajouter des serveurs identiques en parallèle afin de répondre à l'augmentation de la charge. [11]

Le but des systèmes NoSQL est de renforcer la scalabilité horizontale.

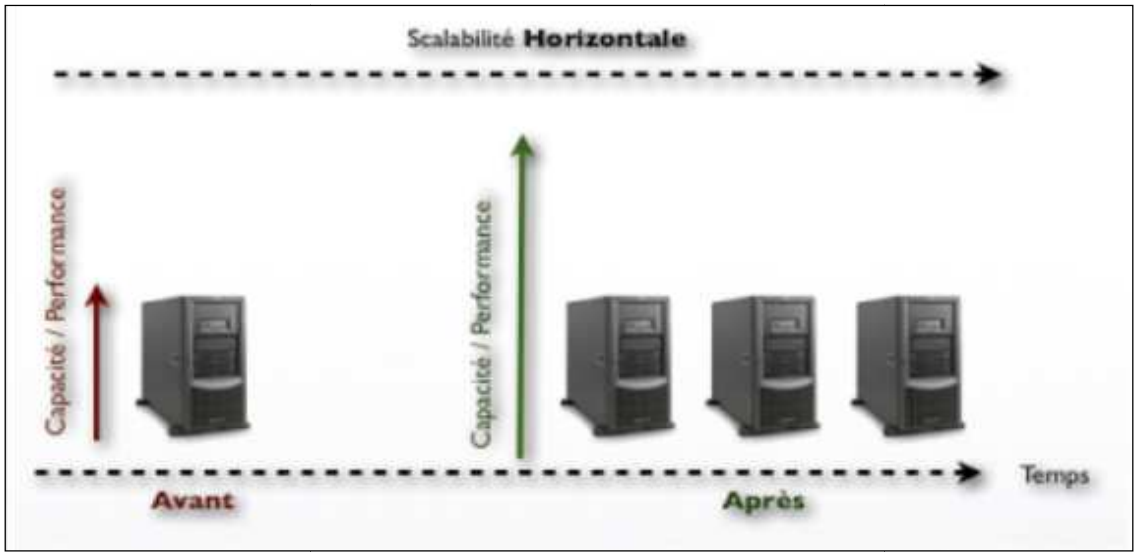


Figure II.1. Scalabilité Horizontale

VI.2 Théorème de CAP

Le théorème de CAP est l'acronyme de « Coherence », « Availability » et « Partition tolérance », Seuls deux des trois contraintes peuvent être respectés en même temps.

Pour que les SGBDs NoSQL puissent garantir une meilleure scalabilité horizontale, il faut respecter le principe de tolérance au partitionnement, ce qui exige l'abandon soit de la cohérence, soit de la haute disponibilité.

- « **Cohérence** » (Cohérence) : Tous les clients du système voient les mêmes données au même instant.

- **« Availibility** » (Haute disponibilité) : Un système est dit disponible si toute requête reçue par un nœud retourne un résultat. Bien évidemment le nœud en question ne doit en aucun cas être victime de défaillance.

- **« Partition tolerance** » (Tolérance à la partition) : Un système est dit tolérant à la partition s'il continue à répondre aux requêtes de manière correcte même en cas de panne autre qu'une panne totale du système. [4]

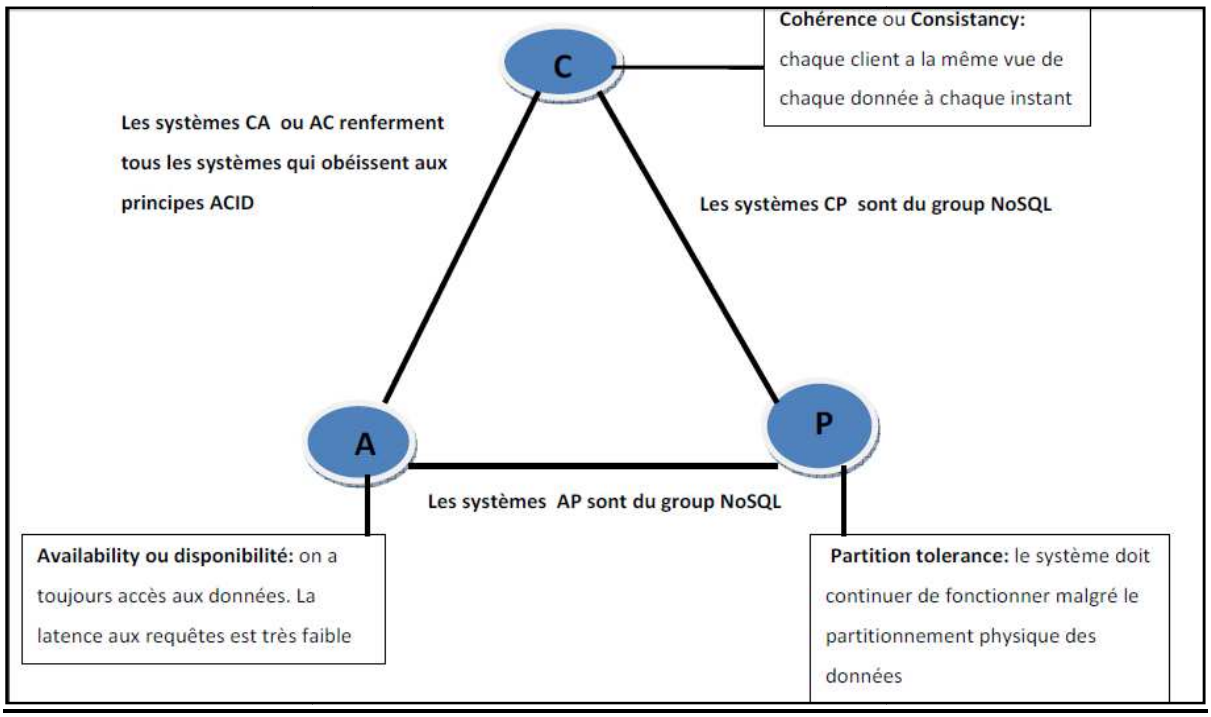


Figure II.2.Théorème CAP

VI.3 Les propriétés de BASE

Les SGBDs NoSQL qui, selon le théorème CAP, privilégient la disponibilité ainsi que la tolérance à la partition plutôt que la cohérence, répondent aux propriétés de BASE.
Les caractéristiques BASE sont fondées sur les limites que montrent les SGBD relationnelles, dont voici la description [8] :

- **Basically Available** (Disponibilité basique) : Même en cas de désynchronisation ou de panne d'un des nœuds du cluster, le système reste disponible selon le théorème CAP.
- **Soft-state** (Cohérence légère) : Cela indique que l'état du système risque de changer au cours du temps, sans pour autant que des données soient entrées dans le système. Cela vient du fait que le modèle est cohérent à terme.
- **Eventual consistancy** (Cohérence à terme) : Cela indique que le système devient cohérent dans le temps, pour autant que pendant ce laps de temps, le système ne reçoive pas d'autres données.

VII. Type de base de données NoSQL

Il en existe 4 types distincts qui s'utilisent différemment et qui se prêtent mieux selon le type de données que l'on souhaite y stocker :

VII .1 Les bases de données clé-valeur

La base de données de type clé-valeur est considérée comme la plus élémentaire. Son principe est très simple, chaque valeur stockée est associée à une clé unique. C'est uniquement par cette clé qu'il sera possible d'exécuter des requêtes sur la valeur. [9]

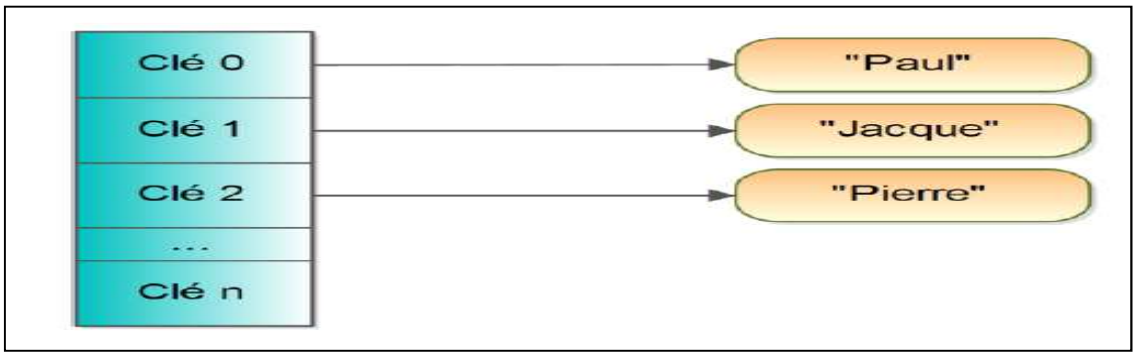


Figure II.3. Les bases de données clé-valeur

Les caractéristiques principales du système associatif [10] :

- Aucun SQL
- Meilleure scalability horizontale du marché.
- Peut ne pas fournir un support des propriétés ACID.
- Peut offrir une architecture distribuée et tolérante aux pannes.

VII .2 Les bases de données orientées document

La représentation en document est particulièrement adaptée au monde du Web. Il s'agit d'une extension du concept de clé-valeur qui représente la valeur sous la forme d'un document contient des données organisées de manière hiérarchique à l'image de ce que permettent XML ou JSON. Étant consciente du contenu qu'elle stocke, la base de données peut alors effectuer des indexations de différents champs et offrir des requêtes plus élaborées. [7]

Les principaux avantages de ce type de système sont donc :

- Il est plus performant d'extraire des données pour une densité importante d'informations.
- Améliore grandement les performances sur les tris ou agrégations de données car ces opérations sont réalisées via des clés de lignes déjà triées. [10]

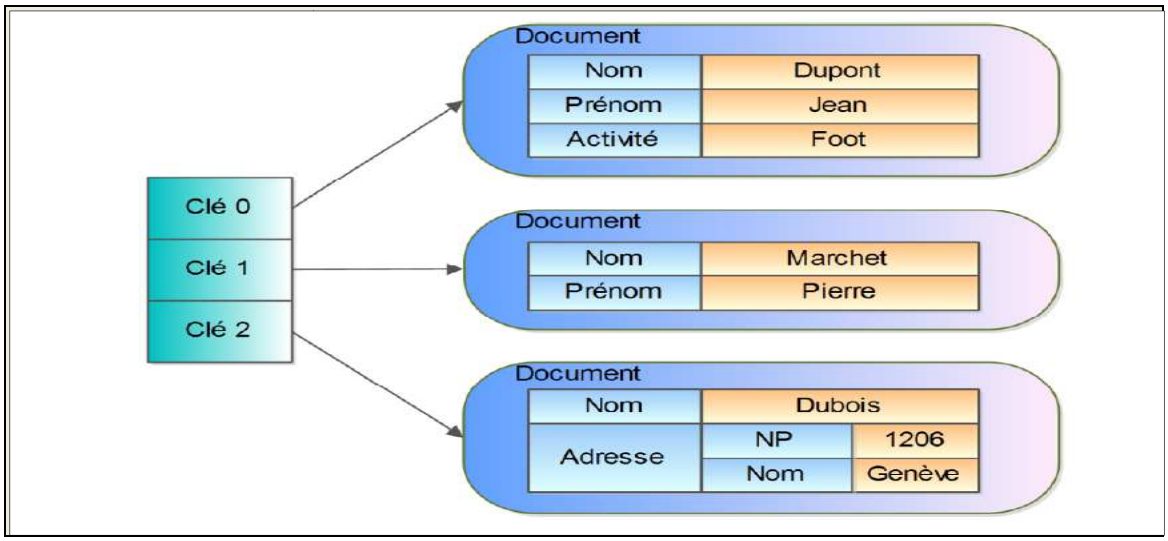


Figure II.4. Les bases de données orientées document

VII .3 Les bases de données orientées colonnes

Le concept de colonnes est le plus simple à saisir, car l'analogie avec les bases relationnelles est proche. Dans les concepts à appréhender il existe des tables, ce qui permet de bien comprendre comment les données sont organisées. [7]

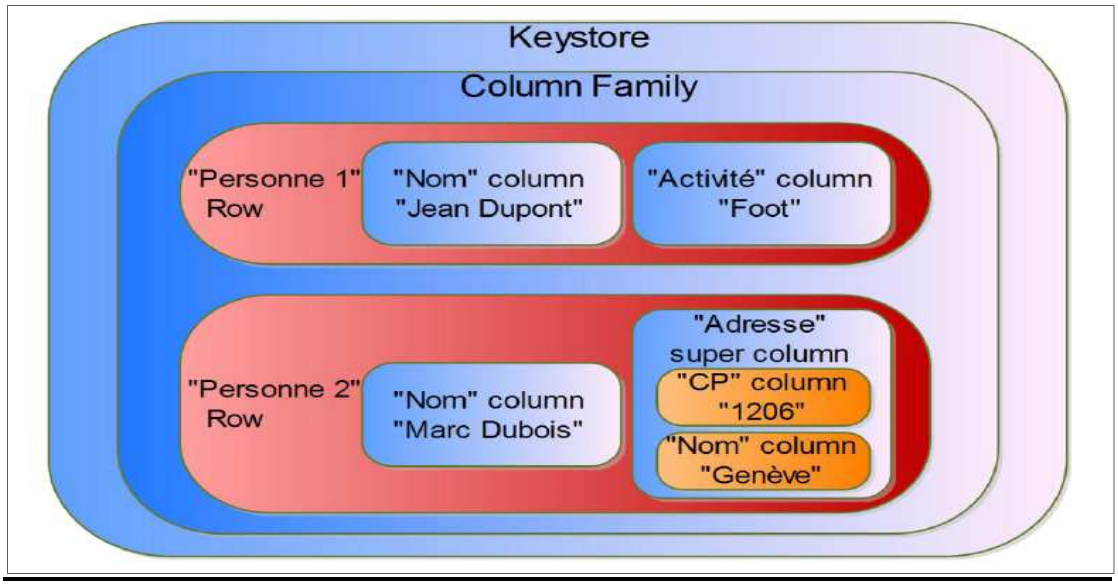


Figure II.5. Les bases de données orientées colonnes

VII .4 Les bases de données orientées graphe

Les bases orientées graphe sont les moins connues de la mouvance NoSQL. Ces bases permettent la modélisation, le stockage ainsi que le traitement de données complexes reliées par des relations. Ce modèle est composé d'un :

- **Moteur de stockage pour les objets** : c'est une base documentaire où chaque entité de cette base est nommé nœud.
- **Mécanisme qui décrit les arcs** : c'est les relations entre les objets, elles contiennent des propriétés de type simple (integer, string, date..). [8]

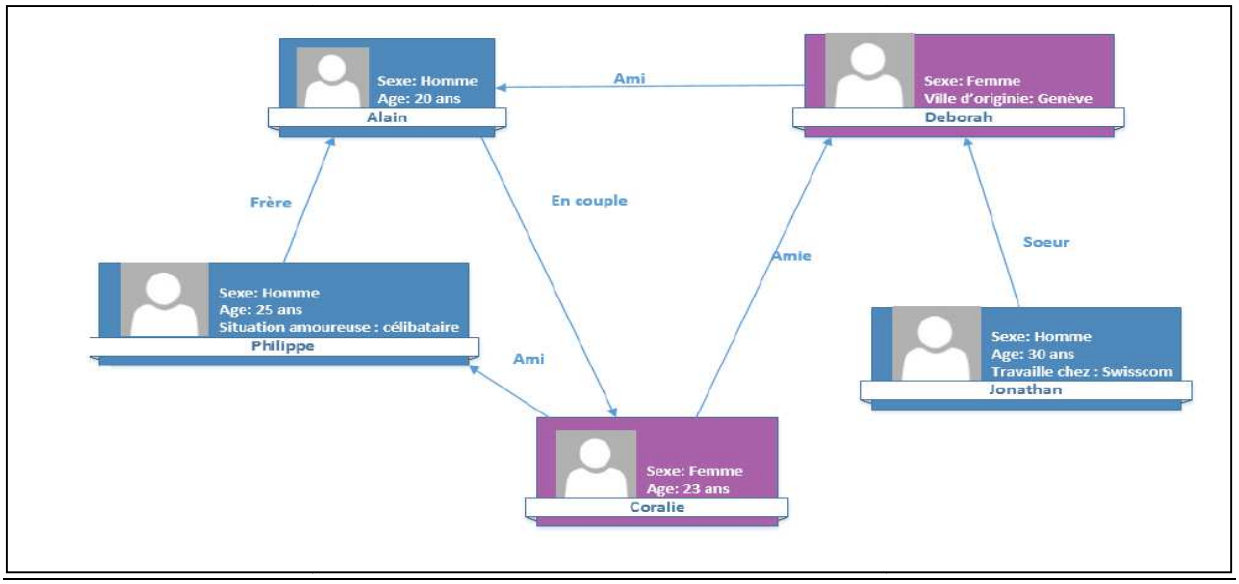


Figure II.6. Les bases de données orientées graphe

VIII. Le marché

Les géants du Web (Facebook, LinkedIn, Google etc.), pour répondre à leurs besoins, ont développé plusieurs solutions. On décompte aujourd'hui 150 bases de données NoSQL de différents types (Clé-valeur, Colonne, Document, Graphe). La majorité de ces bases de données se trouve sous licence Open Source.

VIII .1 Redis (orientée clé-valeur)

Redis est un projet Open Source de type clé/valeur sous licence BSD. Il dispose de plus de fonctionnalités par rapport à la majorité des autres solutions du même type. Notamment par rapport au fait que Redis permet la manipulation des chaînes ainsi que les stockages de collections. [8]

VIII .2 DynamoDB (orientée clé-valeur)

Solution d'Amazon à l'origine de ce type de base clé-valeur. Parce c'est un service basé sur le Cloud, il permet aux clients qui l'utilisent de s'affranchir de lourdes tâches administratives, les données sont stockées sur des disques SSD sur trois zones de disponibilités. [6]

VIII .3 Voldemort (orientée clé-valeur)

Voldemort est un projet Open Source de type clé/valeur sous licence Apache 2.0. Il a été nommé d'après un personnage du célèbre film « Harry Potter ». [8]

VIII .4 Cassandra (orientée colonne)

Cassandra est une solution de type orienté colonnes, mise en Open Source par Facebook en 2008, il est aujourd'hui l'un des principaux projets, de la fondation Apache. Il permet la gestion massive de données réparties sur plusieurs serveurs, assurant ainsi une haute disponibilité des données. [6]

VIII .5 Neo4j (orienté graphe)

Neo4j est un système de base de données. C'est un projet Open Source sous licence GPLv3. Sa première version est sortie en 2007. Ce type de solution est utilisé dans le monde des réseaux sociaux (Ex : amis sur Facebook). [8]

IX. La distribution des données

IX.1 Le sharding

Le Sharding sert à partager les données entre plusieurs Shard, chaque Shard devant stocker une partie des données. Le Sharding sert à Résoudre le problème d'une trop forte charge sur les serveurs pour l'accès à des données disjointes. [9]

IX.2 Réplication point à point

Les lectures / écritures se font sur n'importe quel nœud du cluster. Et Inconsistance en cas d'écritures simultanées sur un même sous ensemble de données à partir de différents maîtres. Et Merge des écritures possible malgré tout. [13]

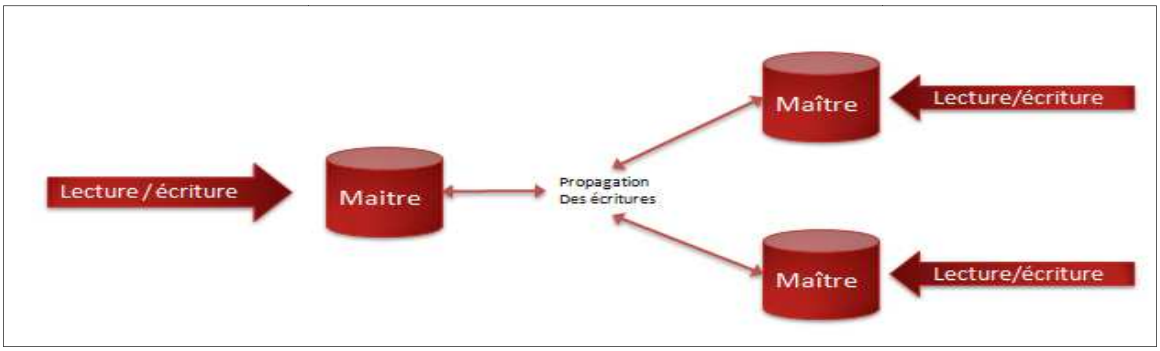


Figure II.7. Réplication point à point

IX.3 Maître/ Esclave

La distribution de données sur un schéma dit maître/esclave consiste à avoir un seul serveur dit maître dans un cluster, les autres serveurs étant esclave, les requêtes des clients arrivent directement sur le serveur maître, pour ensuite être redirigées sur les serveurs «esclaves» qui contiennent l'information de la requête.

L'esclave est présent pour prendre la place du maître si ce dernier a faillit. Le premier avantage de cette structure permet de garantir une forte cohérence des données, car seul le maître s'occupe des clients. [6]

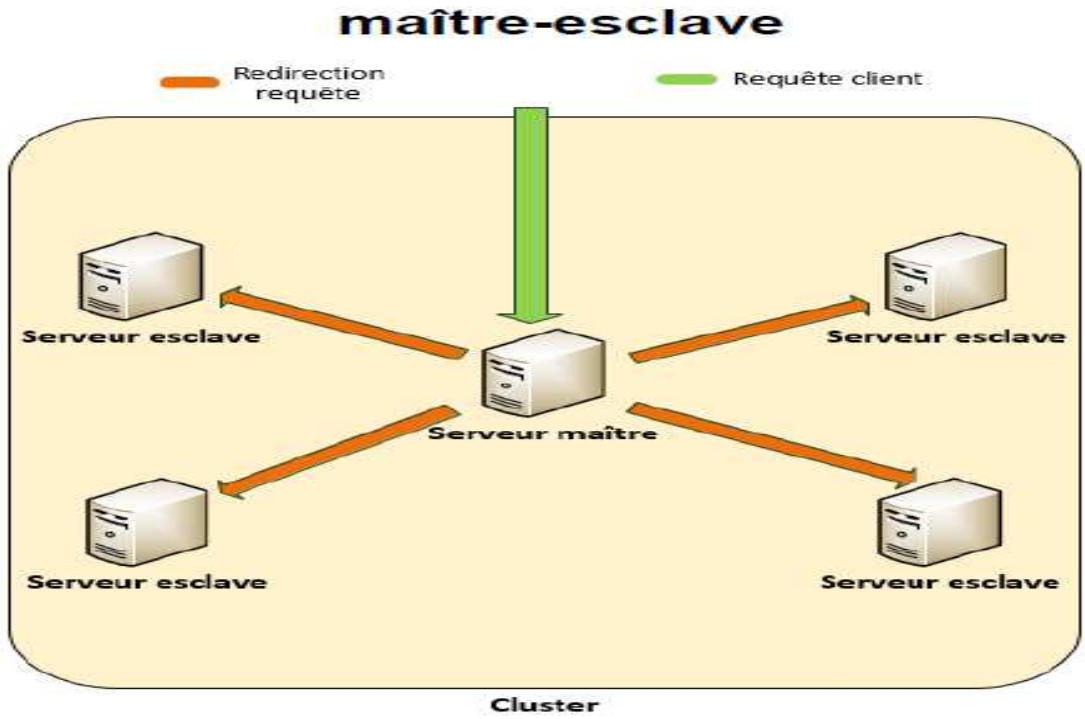


Figure II.8. maître/esclave

X. MapReduce

X.1 Définition

MapReduce permet de traiter une grande quantité de données de manière parallèle, en les distribuant sur divers nœuds d'un cluster. Ce mécanisme a été mis en avant par Google en 2004 et a connu un très grand succès. [6]

X.2 Principe de MapReduce

Le principe de MapReduce est simple, il s'agit de découper une tâche manipulant un gros volume de données en plusieurs tâches traitant chacune un sous-ensemble de ces données. Dans la première étape (Map) les tâches sont donc dispatchées sur l'ensemble des noeuds. Chaque noeud traite un ensemble des données. Dans la deuxième étape, les résultats sont consolidés pour former le résultat final du traitement (Reduce). [6]

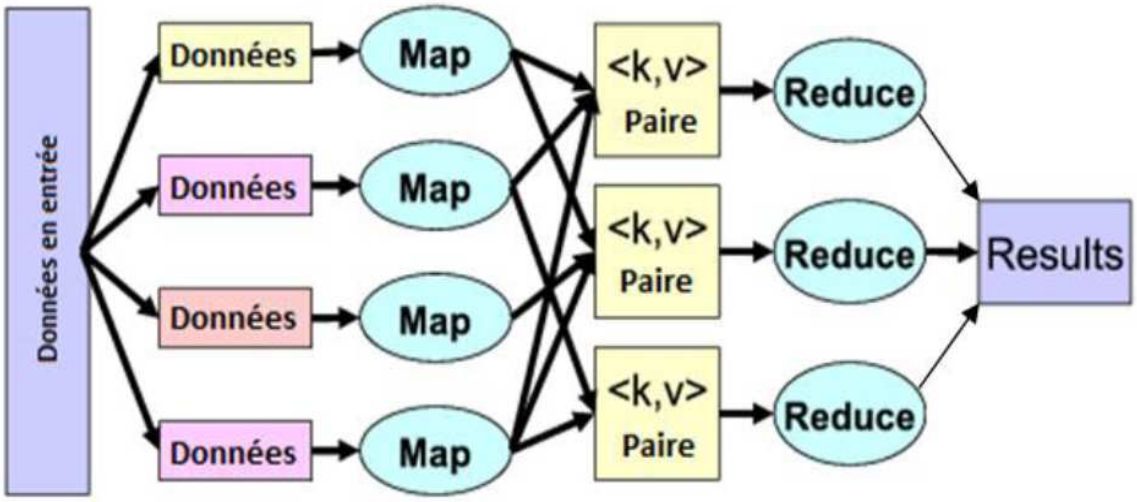


Figure II.9. Les étapes MapReduce

XI. Conclusion

Le NoSQL est une nouvelle approche de stockage et de manipulation de données, dont la plupart ont vu le jour pendant les 4 ou 5 dernières années. Elles ont su se démocratiser grâce leur large utilisation par les grandes entreprises du domaine informatique, qui ont adopté ces nouvelles solutions pour répondre à leurs besoins d'évolutivité

I. Introduction

Nous nous intéresserons particulièrement dans ce chapitre aux technologies MongoDB, HBASE et Hadoop, en tant comme framework Java destiné aux applications distribuées et à la gestion intensive des données. Ce sont des technologies récentes, encore relativement peu connues du grand public mais auxquelles nous associons déjà des grands noms parmi lesquels: Facebook, Yahoo, Twitter et autres.

Nous présentons en fin de chapitre l'outil Yahoo ! Cloud Serving Benchmark (YCSB) qui est un benchmark et générateur de banque d'essais pour les bases de données NoSQL.

II. Justification des choix des SGBDs

- SGBD Hbase : il fait partie, actuellement, de l'implémentation de référence dans le domaine des Big Data et Cloud computing, à savoir Hadoop. On rappelle aussi que FaceBook a basculé du Cassandra vers le Hbase pour des questions de performance.

- SGBD MongoDB : Par sa simplicité d'utilisation du point de vue développement client, ainsi que ces performances remarquables. MongoDB est la base de données NoSQL la plus utilisée, actuellement.

III. HADOOP

III.1 Historique

Doug Cutting, l'homme qui a créé Hadoop, le framework open source destiné à la gestion intensive des données, cherchait une solution quant à la distribution du traitement de Lucene afin de bâtir le moteur d'indexation web Nutch. Il décidait donc de s'inspirer de la publication de Google sur leur système de fichier distribué GFS (Google File System). Premièrement, renommer NDFS, il sera rebaptisé HDFS pour Hadoop Distributed File System [5].

III.2 Présentation d'Hadoop

Hadoop est un framework open source développé en java, et fondée sur le modèle MapReduce de Google et les systèmes de fichiers distribués (HDFS). Faisant partie des projets de la fondation de logiciel Apache depuis 2009. Il est destiné à faciliter le développement d'applications distribuées et scalables, permettant la gestion de milliers de noeuds ainsi que des péta octets de données. [5]

Hadoop est utilisé particulièrement dans l'indexation et le tri de grands ensembles de données, le data mining, l'analyse de logs, et le traitement d'images. Le succès de Google lui est en partie imputable. Il est aujourd'hui l'un des outils les plus pertinents pour répondre aux problèmes du Big Data.

Nous allons détailler le fonctionnement des deux technologies phares de Hadoop : HDFS et Map/Reduce car les technologies sont très liées aux concepts développés dans ces deux produits.

III.3 Hadoop Distributed FileSystem (HDFS)

Dans Hadoop, les différents types de données**,** qu'elles soient structurées ou non, sont stockées à l'aide du HDFS. Le HDFS va prendre les données en entrée et va ensuite les partitionner en plusieurs blocs de données. Afin de garantir une disponibilité des données en cas de panne d'un nœud, le système fera un réplica des données. Par défaut les données sont répliquées sur trois nœuds différents. [5]

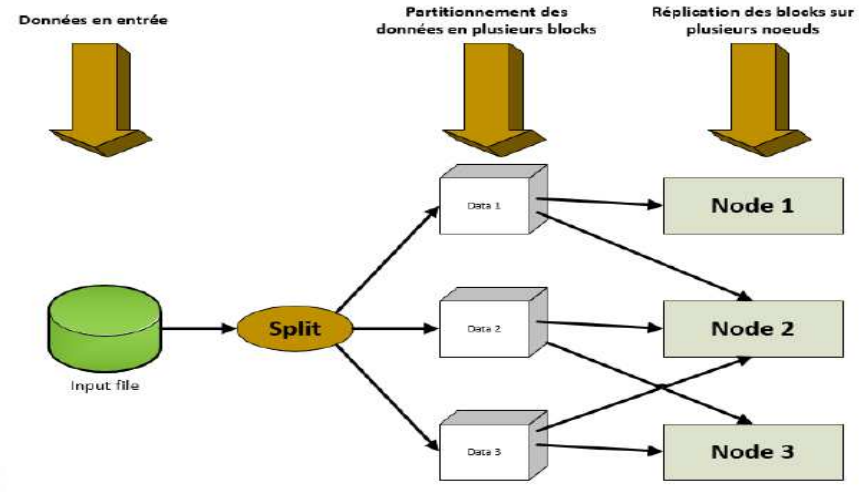


Figure III .1.Hadoop Distributed File System (HDFS)

III.4 Typologie d'un cluster Hadoop

Hadoop repose sur un schéma dit « maître-esclave » et peut être décomposé en cinq éléments (Figure III.2) [5] :

- **Le nom du nœud (NameNode) :** Le « NameNode » est la pièce centrale dans le HDFS, il maintient une arborescence de tous les fichiers du système et gère l'espace de nommage. Il centralise la localisation des blocs de données répartis sur le système. Sans le « NameNode», les données peuvent être considérées comme perdues car il s'occupe de reconstituer un fichier à partir des différents blocs répartis dans les différents « DataNode ». Il n'y a qu'un «NameNode» par cluster HDFS.

- **Le gestionnaire de tâches (Job Tracker) :** Il s'occupe de la coordination des tâches sur les différents clusters. Il attribue les fonctions de MapReduce aux différents « TaskTrackers ».

- **Le moniteur de tâches (Tasktracker) :** Il permet l'exécution des ordres de mapReduce, ainsi que la lecture des blocs de données en accédant aux différents « Data Nodes ». Par ailleurs, le « TaskTracker » notifie de façon périodique au « Job Tracker » le niveau de progression des tâches qu'il exécute,

- **Le noeud secondaire (Secondarynode) :** N'étant initialement pas présent dans l'architecture Hadoop, celui-ci a été ajouté par la suite afin de répondre au problème du point individuel de défaillance. Le « Secondary Node » va donc périodiquement faire une copie des données du « NameNode » afin de pouvoir prendre la relève en cas de panne de ce dernier.

- **Le nœud de données (DataNode) :** Il permet le stockage des blocs de données. Il communique périodiquement au « NameNode » une liste des blocs qu'il gère. Un HDFS contient plusieurs nœuds de données ainsi que des réplications d'entre eux. Ce sont les nœuds esclaves.

Un Cluster Hadoop peut être constitué de machines hétérogènes, que ce soit au niveau du hardware comme au niveau software (système d'exploitation). [5]

Figure III.2. Architecture d'un Cluster Hadoop

Les points forts d'Hadoop se résument dans ses caractéristiques suivantes :

- Evolutif, car pensé pour utiliser plus de ressources physiques, selon les besoins, et de manière transparente.

- rentable, car il optimise les coûts via une meilleure utilisation des ressources présentes.

- souple, car il répond à la caractéristique de variété des données en étant capable de traiter différents types de données. Et enfin, résilient, car pensé pour ne pas perdre d'information et être capable de poursuivre le traitement si un nœud du système tombe en panne. [14]

III.5 Implémentation de MapReduce dans Hadoop

Le second composant majeur d'Hadoop est MapReduce, qui gère la répartition et l'exécution des requêtes sur les données stockées par le cluster. Le framework MapReduce est conçu pour traiter des problèmes parallèlisables à très grande échelle en s'appuyant sur un très grand nombre de nœuds .L'objectif de MapReduce et de son mécanisme avancé de distribution de tâches est de tirer parti de la localité entre données et traitements sur le même nœud de façon à minimiser l'impact des transferts de données entre les nœuds du cluster sur la performance. [19]

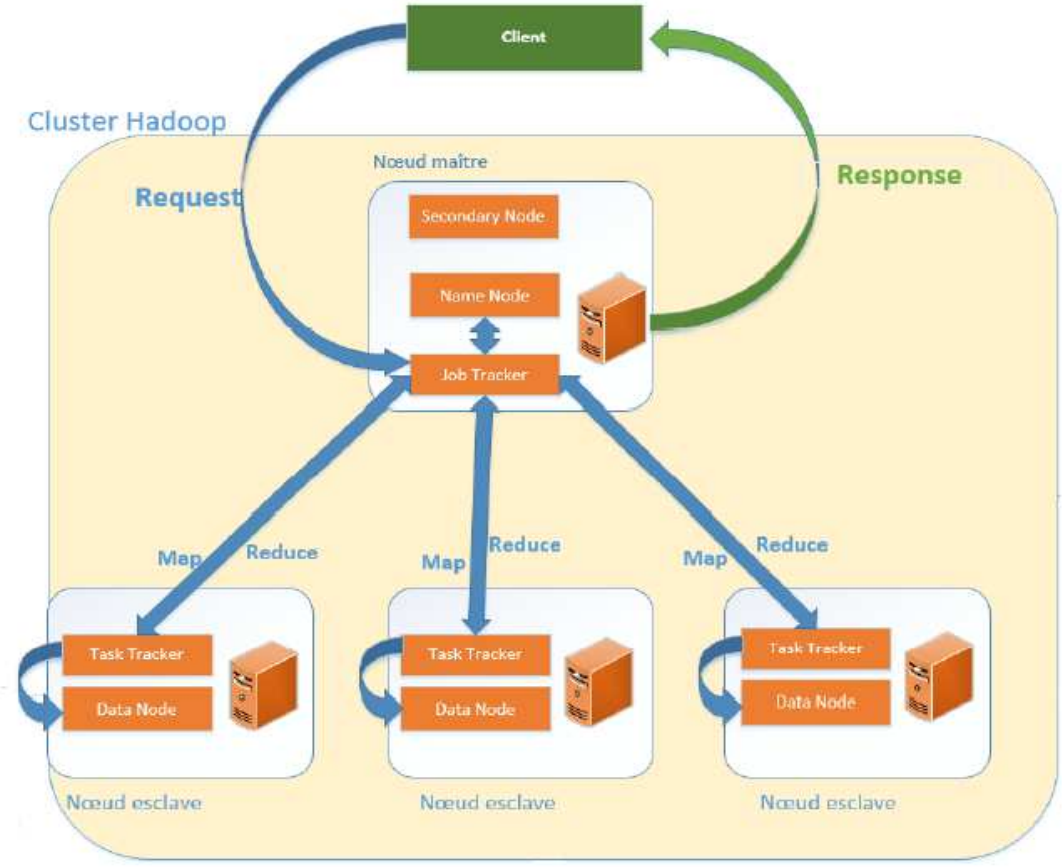


Figure III.3.Fonctionnement de MapReduce dans Hadoop

Le processus MapReduce dans le framework Hadoop, fonctionne de la manière suivante : Pour commencer, le « Job Tracker » va recevoir une requête cliente « Map », il va alors demander au « Name Node » quelles sont les informations nécessaires ainsi que leur localisation dans le cluster pour répondre à la requête du client. Une fois la réponse reçue, le « Job Tracker » enverra la fonction « map » aux différents « Task Trackers ».

Les « Task Trackers » vont alors chercher les données correspondant au traitement sur leur « Data Nodes ». Une fois que les fonctions « map » ont terminés leur traitement, les résultats deux-ci sont stockés. Il est bon de noter que les données sont compilées au niveau de chaque « Data Nodes », et non pas de manière centralisée. C'est ce qui fait la caractéristique principale de Hadoop. Une fois que la partie « Map » est terminée, c'est la partie Reduce qui commence à faire remonter les différents résultats et les consolider en un seul résultat final (voir Figure III.3 fonctionnement de MapReduce) pour répondre à la requête du client. [5]

IV. HBASE

IV.1 Historique

- 2004 : Google publie le MapReduce (algorithme pour les systèmes distribués).
- 2006 : Doug Cutting crée Hadoop (framework de stockage en cluster) et Google à été utilisé Hadoop pour créer BigTable (SGBD BigData propriétaire).
- 2007 : la création de HBase (BigTable en version open source) par Powerset qui contribue au développement de Hadoop F. Chang, R.E. Gruber et al.
- 2008 : Powerset se fait racheter par Microsoft qui abandonne le développement de HBase (car open source) et Hadoop devient le projet principal de la fondation Apache.
- 2010 : HBase devient le projet principal de la fondation Apache L. George. [15]

IV.2 Présentation de HBase

HBase est un système de gestion de base de données non-relationnelles distribué, écrit en Java, disposant d'un stockage est distribué sur toutes les machines du cluster HBase. Fait partie du projet HADOOP de la fondation Apache, il s'agit d'un SGBD orientée colonne.Il a été calqué sur les travaux de BigTable de Google. La solution HBase se repose sur le système de fichier distribué (HDFS : hadoop distributed file system) qui se caractérise par une haute tolérance aux pannes et pouvant être déployée sur du matériel a faible coût. [16]

Il permet un accès très rapide aux données et est adapté aux applications gérant de grands ensembles de données. D'autres parts il propose aussi la possibilité de gérer les performances dans l'optique d'une utilisation plus flexible. Orientée colonne désigne la manière dont le SGBD sérialise les données. Contrairement au SGBD orienté ligne (SGBD orienté relationnel), les données sont sérialisées sous forme de colonne plutôt que de ligne. [2]

IV.3 Le modèle de données

Le modèle se base sur six concepts, qui sont :

- Table : dans HBase les données sont organisées dans des tables.
- Row : dans chaque table les données sont organisées dans des lignes. Une ligne est identifiée par une clé unique (RowKey).
- Column Family : Les données au sein d'une ligne sont regroupées par column family. Chaque ligne de la table à les mêmes column family.
- Column qualifié : L'accès aux données au sein d'une column family se fait via le column qualifié ou column.
- Cell : Les données stockées dans une cellule sont appelée les valeurs de cette cellule.
- Version : Les versions sont identifiées par leur timestamp (de type long). Le nombre de versions est configuré via la Column Family.

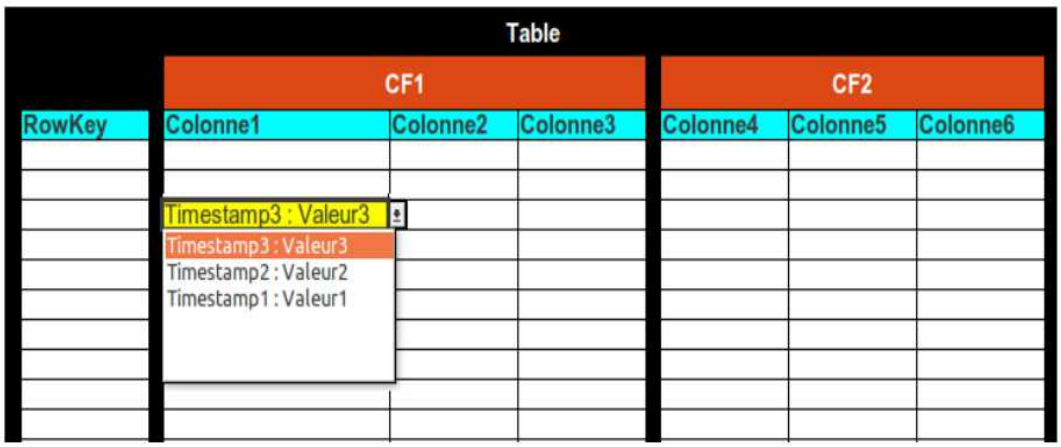


Figure III.4. Vue synthétique du modèle de données dans HBase

IV.4 Architecture

HBase utilise HDFS pour stocker les données et un ensemble d'algorithmes qui permettent une lecture/écriture aléatoire. Dans ce qui suit nous allons détailler les différents éléments de l'architecture de la base NoSQL HBase.

Figure III.5. Vue conceptuelle de la base donnée dans HBase

IV.4.1 Répartition de la données (partitionnement)

Dans HBase, la scalabilité est basé sur le principe de la répartition de charge. Chaque partition est une Région qui stocke d'une manière contigüe, un ensemble de lignes triées selon la rowkey. Chaque Région est héberge par un serveur physique.

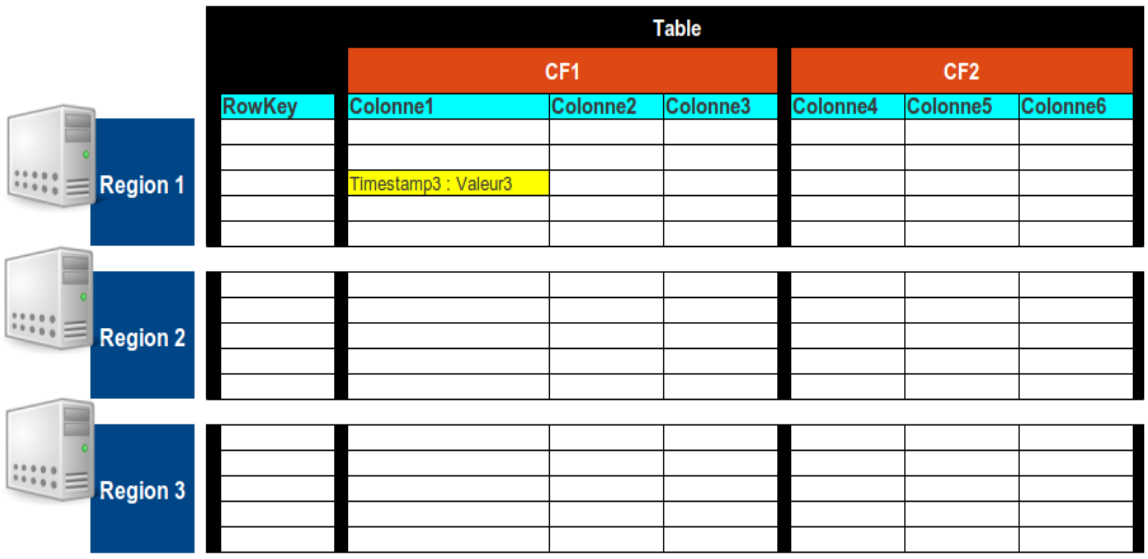


Figure III.6. Répartition des données dans HBase

IV.4.2 Les composants de HBase

La Figure III .7 illustre l'architecture de HBase avec ses composants [16] :

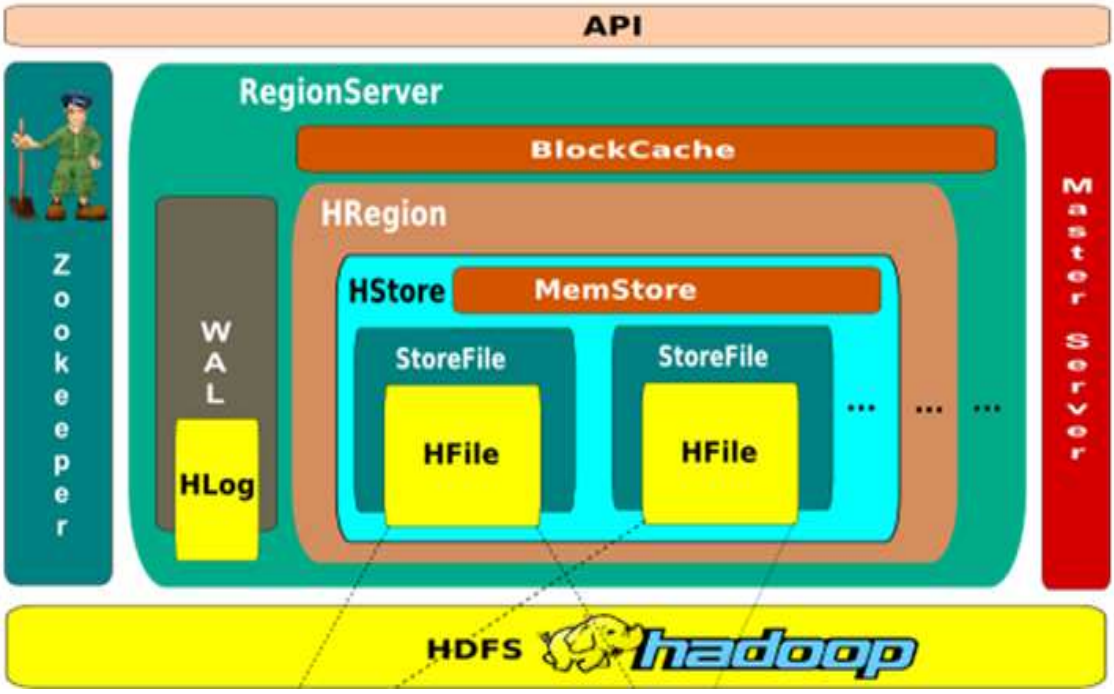


Figure III.7. Anatomie de la base de données NoSQL HBase

- **RegionServer :** Le RegionServer est le point d'entrée pour l'accès à la donnée. IL gère plusieurs processus et composants.

- **BlockCache :** Le BlockCache est activé par défaut pour toutes les tables, qui charge toute opération de lecture.

- **Write Ahead Log (WAL) :** Chaque ajoute ou mises à jour dans un RegionServer sont systématiquement écrit dans WAL en premier lieu. Avant d'être répliquer dans le MemStore. Ce mécanisme garantit la durabilité de la donnée en cas de défaillance du serveur. Le processus WAL écrit dans le fichier Hlog qui est placé dans HDFS. Pour chaque instance RegionServer il y a une instance de WAL.

- **Region :** C'est l'élément de base dans le stockage et la distribution de la donnée dans HBase, dont l'implémentation est HRegion. Chaque Region gère un sous ensemble d'une table HBase (une partition).

- **Master Server :** HMaster est l'implémentation du Master Server. Le Master Server est chargé de coordonner et de surveiller toutes les instances de RegionServer du cluster. Il en charge de la répétition des Region(s) sur les nœuds du cluster.
- **Zookeeper :** Il surveille l'état du cluster et informe régulièrement le Master Server des différents états. De plus, il stocke les informations critiques du cluster, dont l'emplacement de la table système -ROOT- :

-ROOT- : contient la liste des tables .META. et leur emplacement.

.META. : contient la liste des Region et leurs emplacement.

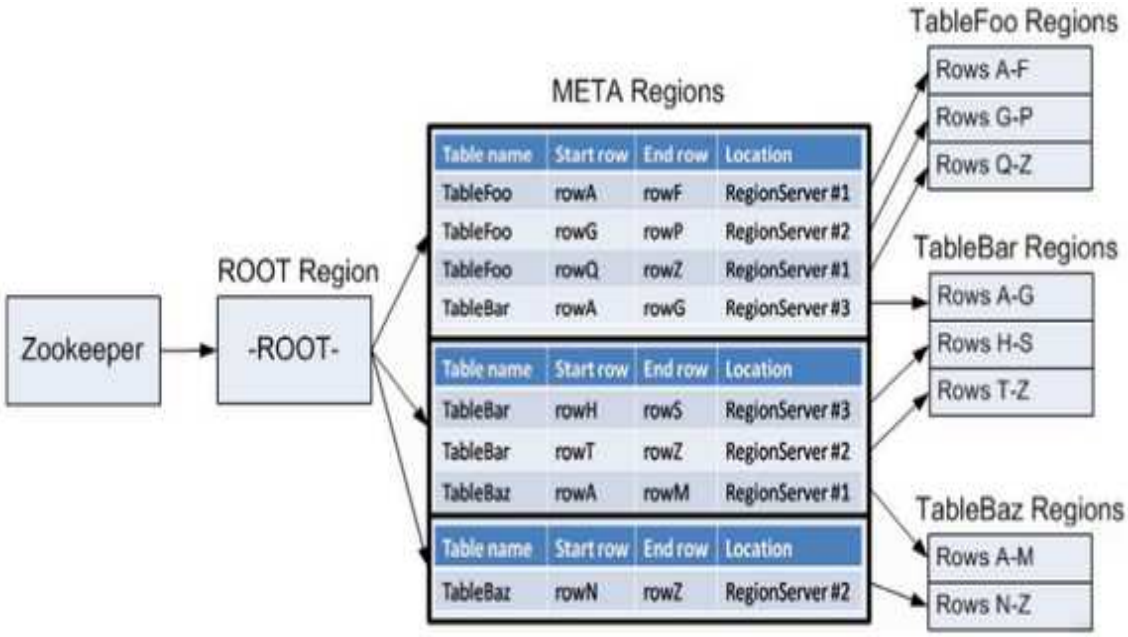


Figure III .8.Gestion des métadonnées avec ZooKeeper

IV.4.3 Lecture

Le client HBase envoi directement une requête de type « get » au Server Region qui contient l'information souhaitée.

IV.4.4 Écriture (Stockage des données)

Contrairement au SGBD orienté ligne (SGBD relationnel), les données sont stockées sous forme de colonne. On retrouve deux types de composants pour HBase : le composant maître appelé «HMaster», qui contient le schéma des tables de données.

Les nœuds esclaves appelés «Region Server», qui gèrent des sous-ensembles de tables, appelés « Region ». HBase utilise le système de fichier HDFS , les données des tables sont stockées sur les nœuds « DataNode» (HDFS) par les noeuds de type « Region Server »*(*HBASE*)*. [17]

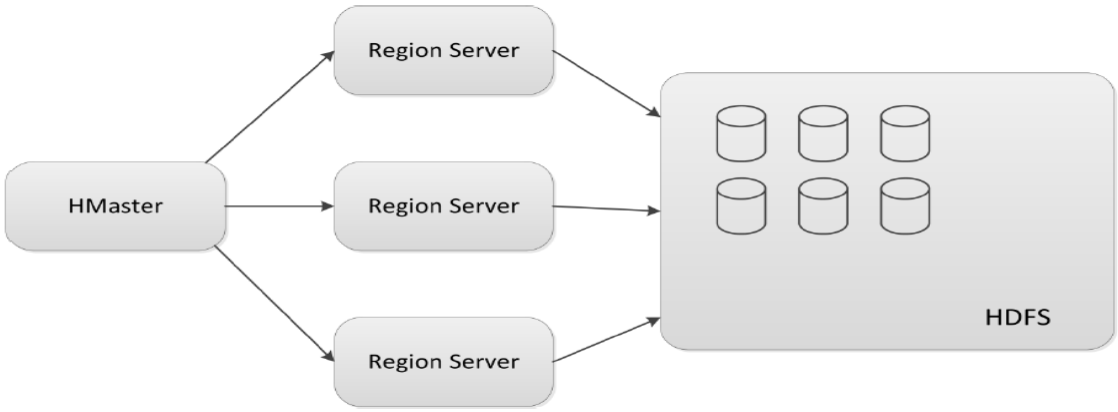


Figure III.9. Stockage des données dans Hbase

V. MONGODB

V .1 Présentation de MongoDB

MongoDBest un système de gestion de base de données orientée document. Développé en C++ et distribué sous licence AGPL (licence libre) depuis 2007 par 10gen (une entreprise travaillant sur des outils de Cloud computing), elle est très adaptée aux applications web. MongoDB a été adopté par plusieurs grands noms de l'informatique, tels que Foursquare, SAP, ou bien même GitHub. [6]

La base de données MongoDB utilise des fichiers au format BSON, un dérivé du JSON. Comparé à ce dernier, le BSON a été pensé pour faciliter le scan des données, les éléments sont préfixés par leur longueur ce qui permet d'accéder aux données plus rapidement. En contrepartie, le BSON utilise plus d'espace, ceci est dû aux préfixes devant chaque élément. [9]

Par sa simplicité d'utilisation du point de vue de développement client, ainsi que ces performances remarquables, MongoDB est l'une de base de données orientées document la plus utilisé. [6]

V .2 Documents sous MongoDB

MongoDB est une base documentaire dans laquelle les documents sont regroupés sous forme de collections, les collections étant l'équivalent des tables du SQL. Il est possible de représenter chaque document au format JSON (MongoDB utilise une variante binaire plus compacte de JSON nommé BSON pour son stockage interne). Chaque document dispose d'une clé unique permettant de l'identifier dans la collection.

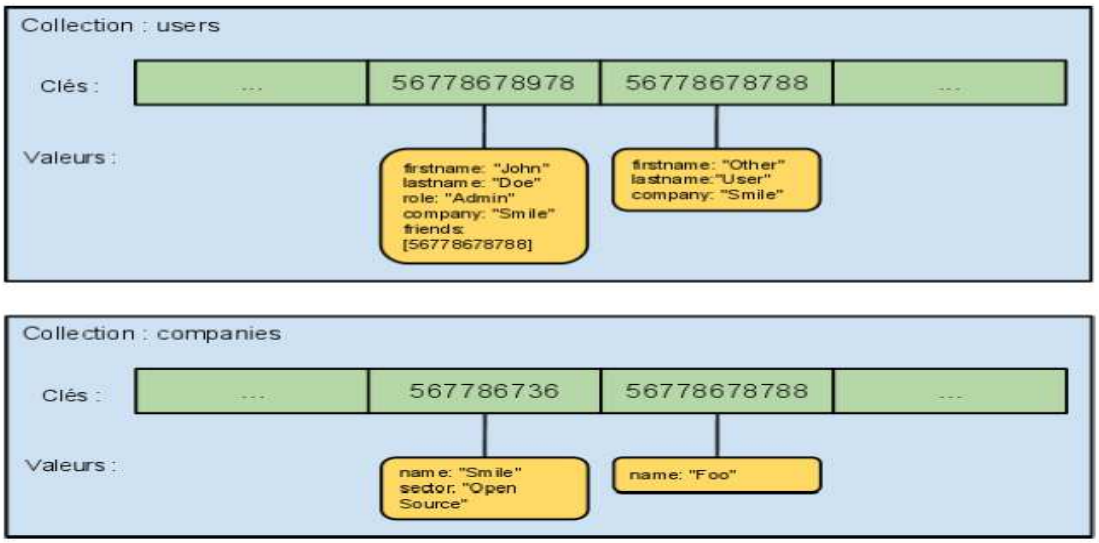


Figure III.10. Documents sous MongoDB

V .3 Architecture

Mongo a été conçu pour fonctionner selon plusieurs modes distincts :

V.3.1 Serveur seul

Le mode Single, qui ne sert à faire fonctionner une base de données que sur un seul serveur. Dans ce mode mettant en jeu un seul serveur, un seul processus nommé mongod est utilisé et traite directement les données issues des requêtes du client. [18]

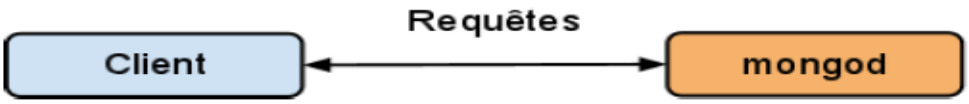


Figure III.11.MongoDB Serveur seul

V.3.2 Réplication Maître/ Esclave

Le mode Master / Slave bien connu dans l'informatique. Un serveur officie en tant que maître et s'occupe des demandes des clients. A côté de cela, il s'occupe de répliquer les données sur le serveur esclave de façon asynchrone. L'esclave est présent pour prendre la place du maître si ce dernier venait à tomber. Il permet de garantir une forte cohérence des données, car seul le maître s'occupe des clients. [18]

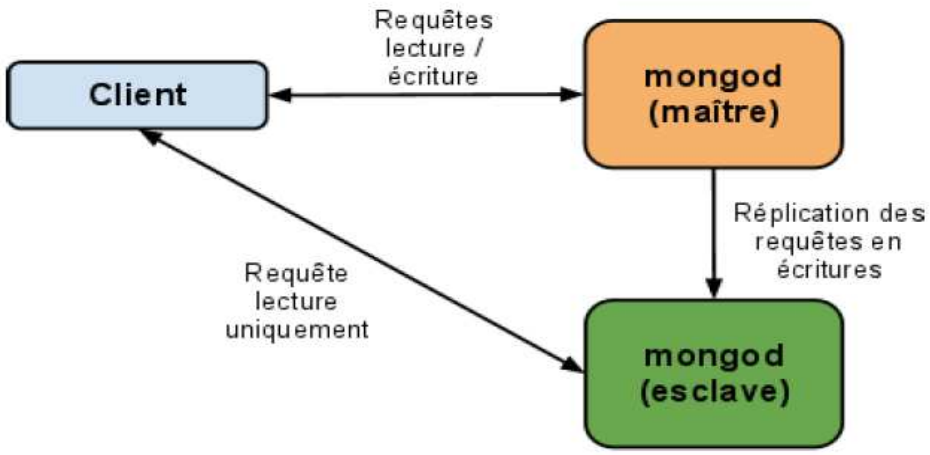


Figure III.12. MongoDB maitre /esclave

V.3.3 Réplication par Replica Set

Il s'agit d'un mode de réplication plus avancé que le mode maître / esclave présenté ci-dessus, ce mode permet d'éviter la présence d'un point de défaillance unique au sein de l'infrastructure par la méthode suivante :

- On ajoute n serveurs au Replica Set : Chaque serveur dispose d'une priorité.
- Un des serveurs est considéré comme le nœud primaire : On peut voir le rôle de nœud primaire comme celui du serveur maître dans le modèle maître / esclave.
- En cas de défaillance du nœud primaire, un nouveau nœud au sein du Replica Set est choisi et devient nœud primaire. [18]

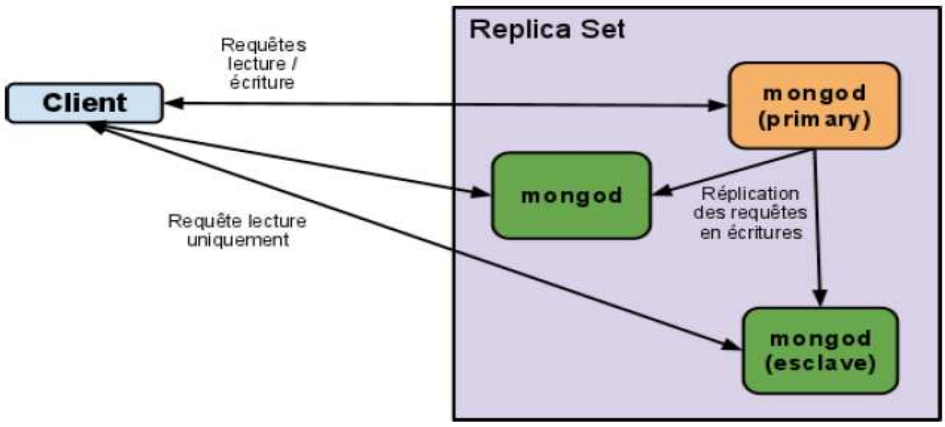


Figure III.13. Réplication par Replica Set

V.3.4 Partitionnement des donnés via sharding

Sharding sert à partager les données entre plusieurs Shard, chaque Shard devant stocker une partie des données.

- Les Shards : Ce sont un groupe de serveurs (Master / Slave ou Réplica Sets).
- Les mongos : Ce sont des serveurs qui savent quelles données se trouvent dans quel Shard et leur donnent des ordres (lecture, écriture).
- Les Config Servers : Ils connaissent l'emplacement de chaque donnée et en informent les mongos. [18]

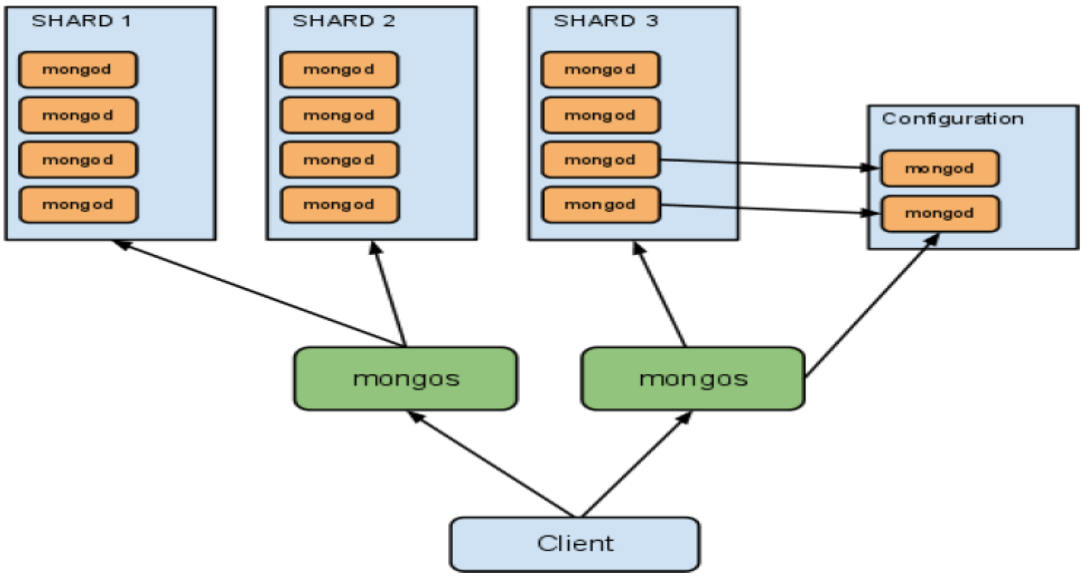


Figure III.14. Partitionnement des donnés via sharding

VI. YCSB : Le Benchmark Yahoo

YCSB est un outil de java très puisant proposé par Yahoo qui se présente comme une nouvelle méthodologie de bancs d'essais (benchmark en anglais) pour les bases de données NoSQL.

YCSB est un excellent outil pour comparer les performances SGBD NoSQL des solutions de stockage diverses. [20]

YCSB propose plusieurs charges de travail configurables qui peuvent être exécutées sur les différentes bases de données NoSQL, par exemple :

- Workload A : 50% de lectures, 50% de mises à jour.
- Workload B : 95% de lectures, 5% des mises à jour.
- Workload C : lectures seules 100% lectures.
- Workload D : Insertion de nouvelles données .Ces données sont lues massivement

VII. Conclusion

Dans ce chapitre nous avons présenté les outils utilisés Hadoop, HBASE, MongoDB, et YCSB.

Nous avons décrit les architectures, les principes de fonctionnement, les caractéristiques techniques et les composants de chaque système. Le chapitre a fait l'objet d'une présentation générale de l'environnement dans lequel notre étude comparative va être menée, à savoir les modèles NoSQL qui vont être rapprochés ainsi que le l'outil de comparaison YCSB qui va évaluer et comparer les performances des systèmes HBASE et MongoDB.

I. Introduction

Ce chapitre va décrire, en premier lieu, les étapes d'installation des composants Hadoop, HBASE et Mongo DB. L'exécution des différentes charges de travail, une analyse et interprétation des résultats obtenus à partir des tests en deuxième lieu, pour parvenir à une évaluation globale de performances.

II. Première partie

Dans ce chapitre, nous allons aborder l'installation d'un cluster Hadoop Version 1.0.4 sur Ubuntu 14.10. Pour cela, on va suivre les étapes suivantes :

II.1 La mise à jour du système d'exploitation Ubuntu

```
sudo apt-get update
```

On doit fournir le mot de passe de l'utilisateur.

```
manel@manel:~$ sudo apt-get update
[sudo] password for manel:
```

II.2 Installation la version récente de java

```
manel@manel:~$ sudo apt-get install default-jdk
```

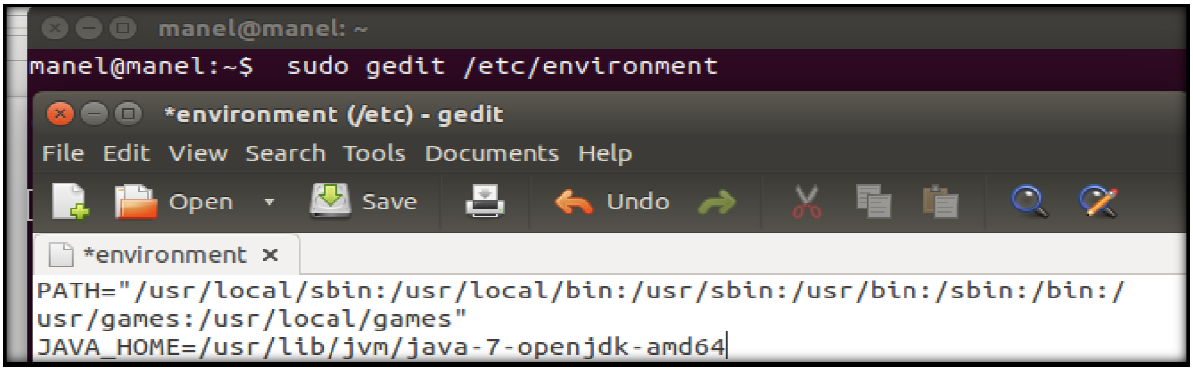
Vérification de la version de java :

```
hduser@manel:~$ java –version
```

```
hduser@manel: ~
hduser@manel:~$  java -version
java version "1.7.0_79"
OpenJDK Runtime Environment (IcedTea 2.5.5) (7u79-2.5.5-0ubuntu0.14.10.2)
OpenJDK 64-Bit Server VM (build 24.79-b02, mixed mode)
hduser@manel:~$
```

II.3 L'ajout du code à l'environnement de ubuntu

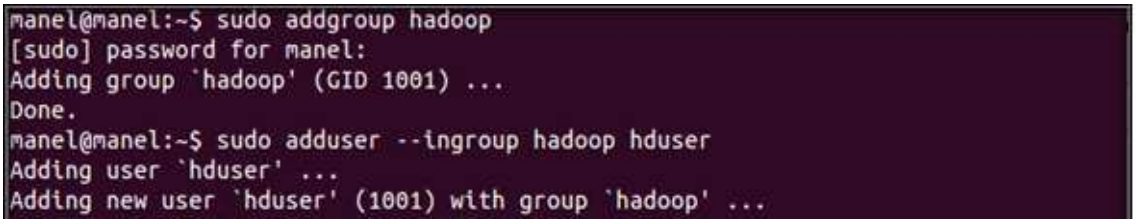
```
JAVA_HOME=/ussr/lib/jvm/java-7-openjdk-amd64
```



II.4 Les permissions d'utilisation de Hadoop

Pour cela, nous devons créer un groupe d'utilisateurs avec tous les droits sur Hadoop.

```
manel@manel:~$  sudo adduser --ingroup hadoop hduser
manel@manel:~$ sudo adduser --ingroup hadoop hduser
```

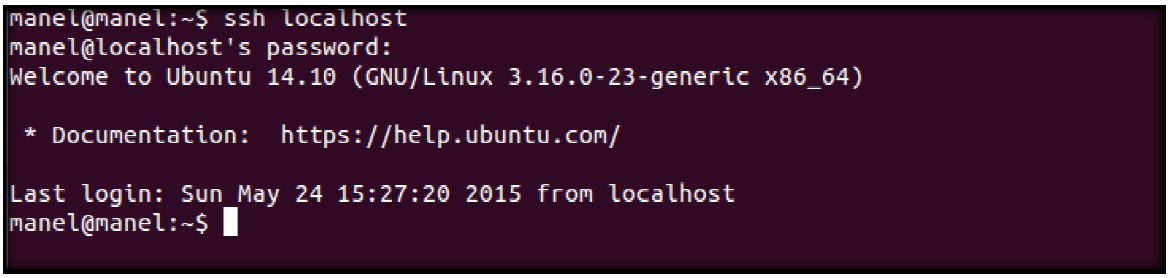


II.5 Installation du openssh-server

```
manel@manel:~$ sudo apt-get install openssh-server
```

Vérifier si open ssh-server est bien installer :

```
manel@manel:~$ ssh localhost
```



Remarque : on doit ajouter l'utilisateur hduser au groupe sudo pour exécuter les commandes comme étant qu'administrateur (root).

```
manel@manel:~$ sudo adduser hduser sudo
manel@manel:~$ su - hduser
```

II.6 Création et installation des certificats SSH

Hadoop nécessite un accès SSH pour gérer ses nœuds, les machines distantes ainsi que notre machine locale. Pour notre configuration à un nœud unique, on a besoin de configurer l'accès SSH à localhost :

```
hduser@manel:~$ ssh-keygen -t rsa -P ""
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hduser/.ssh/id_rsa):
Created directory '/home/hduser/.ssh'.
Your identification has been saved in /home/hduser/.ssh/id_rsa.
Your public key has been saved in /home/hduser/.ssh/id_rsa.pub.
The key fingerprint is:
cd:66:c0:6d:78:39:08:94:ef:9b:79:3a:12:dc:8e:a4 hduser@manel
The key's randomart image is:
+--[ RSA 2048]----+
|      .o.        |
|       .o + .    |
|        .= *     |
|         .* .    |
|      . oS =     |
|       + oo      |
|      o + +      |
|     E o * .     |
|        ..+      |
+-----------------+
hduser@manel:~$
```

Afin qu'Hadoop puisse utiliser SSH sans redemander un mot de passe à chaque accès; nous avons besoin d'ajouter la clé nouvellement créée à la liste des clés autorisées. Pour cela nous devons taper la commande suivante :

```
hduser@manel:~$ cat $HOME/.ssh/id_rsa.pub >>$HOME/.ssh/authorized_keys
```

```
hduser@manel:~$ cat $HOME/.ssh/id_rsa.pub >>$HOME/.ssh/authorized_keys
```

Nous pouvons vérifier si SSH fonctionne :

```
hduser@manel:~$ ssh localhost
```

```
hduser@manel:~$ ssh localhost
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is be:a5:30:af:6d:1a:20:0f:f8:fd:c4:d9:bc:26:a9:f5.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 14.10 (GNU/Linux 3.16.0-23-generic x86_64)
```

On désactive**IPV6**

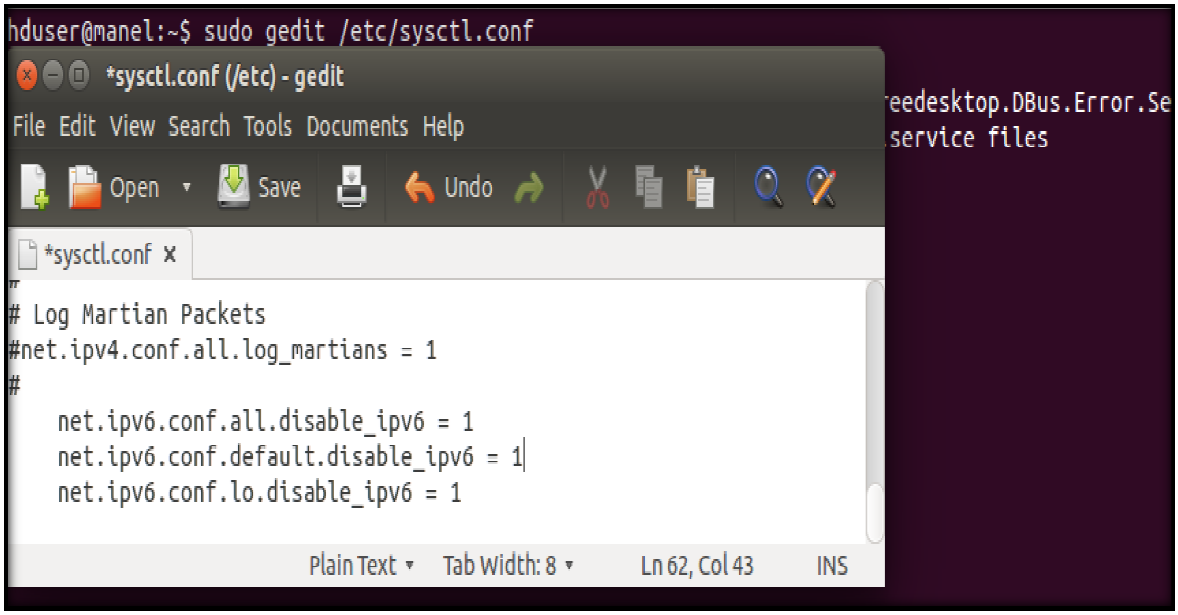
```
hduser@manel:~$ sudo gedit /etc/sysctl.conf
```

Ajouter les lignes suivantes à la fin du fichier :

```
net.ipv6.conf.all.disable_ipv6 = 1

net.ipv6.conf.default.disable_ipv6 = 1

net.ipv6.conf.lo.disable_ipv6 =
```



On doit redémarrer le système pour que le prend le changement en considération.

Apres le redémarrage on lance la commande :

```
hduser@manel:~$ cat /proc/sys/net/ipv6/conf/all/disable_ipv6
```

II.7 L'installation de hadoop 1.0.4

Pour installer Hadoop-1.0.4, nous l'avons téléchargé à partir du site Apache.

Après l'extraction, le dossier Hadoop est déplacé vers /usr/local.

```
hduser@manel:~$ cd /usr/local
hduser@manel:/usr/local$ sudo tar xzf hadoop-1.0.4.tar.gz
hduser@manel:/usr/local$ sudo mv hadoop-1.0.4 hadoop
hduser@manel:/usr/local$ sudo chown -R hduser:hadoop hadoop
hduser@manel:/usr/local$
```

Réglage des fichiers de paramètres :

1. ~/.bashrc
2. /usr/local/hadoop/conf/hadoop-env.sh
3. /usr/local/hadoop/conf/core-site.xml
4. /usr/local/hadoop /conf /mapred-site.xml
5. /usr/local/hadoop/conf /hdfs-site.xml

1. ~/.bashrc

Avant de modifier le fichier /. bashrc dans notre répertoire personnel, nous devons trouver le chemin où Java a été installée pour définir la variable d'environnement JAVA_HOME en utilisant la commande suivante :

```
hduser@manel:/usr/local$ update-alternatives --config java
```

Maintenant, nous pouvons ajouter les lignes suivantes à la fin de ~ / .bashrc

```
hduser@manel:~$ sudo gedit ~/.bashrc
```

Dans ce fichier on a fait la mise jour pour hadoop, hbase et de même temps

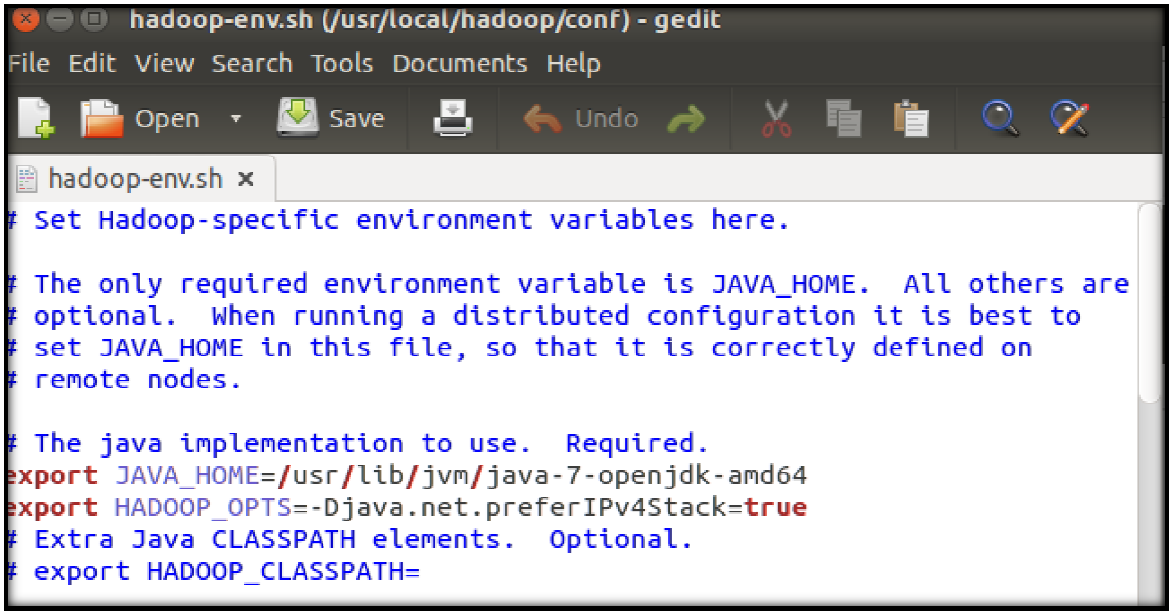
```
# Set Hadoop-related environment variables export
HADOOP_HOME=/usr/local/hadoop
 # Set JAVA_HOME (we will also configure JAVA_HOME directly for
Hadoop later on) export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-
amd64
# Some convenient aliases and functions for running Hadoop-
related commands unalias fs &> /dev/null alias fs="hadoop fs"
unalias hls &> /dev/null alias hls="fs -ls"
# If you have LZO compression enabled in your Hadoop cluster and
 # compress job outputs with LZOP (not covered in this
tutorial):
# Conveniently inspect an LZOP compressed file from the command
 # line; run via:
# # $ lzohead /hdfs/path/to/lzop/compressed/file.lzo
# # Requires installed 'lzop' command.
 # lzohead () { hadoop fs -cat $1 | lzop -dc | head -1000 | less
}
# Add Hadoop bin/ directory to PATH export
PATH=$PATH:$HADOOP_HOME/bin
```

2. /usr/local/hadoop/conf/hadoop-env.sh

```
sudo gedit  /usr/local/hadoop/conf/hadoop-env.sh
```

Nous devons définir JAVA_HOME en modifiant le fichier hadoop-env.sh :

```
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
```

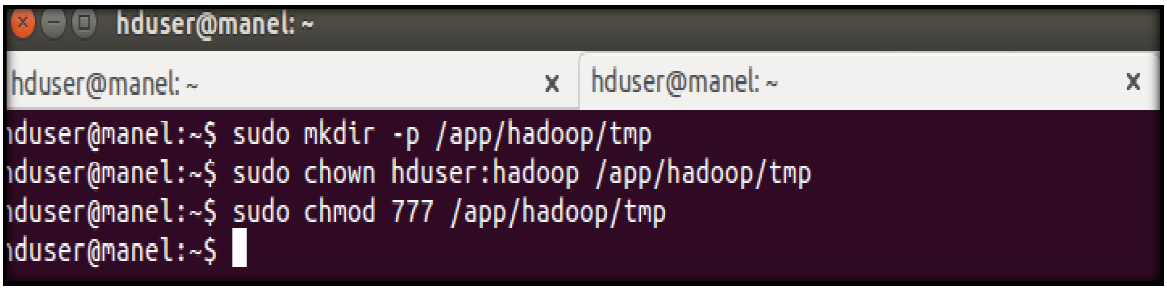


L'ajout de la déclaration ci-dessus dans le fichier hadoop-env.sh assure que la valeur de la Variable JAVA_HOME sera disponible pour Hadoop chaque fois qu'il est mis en marche.

3. /usr/local/hadoop/conf/core-site.xml

Le fichier /usr/local/hadoop/conf/core-site.xml contient les propriétés de configuration utilisées par Hadoop au démarrage.

- Premièrement nous devons créer un dossier de travail pour Hadoop :



Ensuite il faut modifier le fichier /usr/local/hadoop/conf/core-site.xml

```
hduser@manel:~$ sudo gedit /usr/local/hadoop/conf/core-site.xml
```

```
<property>
<name>hadoop.tmp.dir</name>
<value>/app/hadoop/tmp</value>
<description>A base for other temporary directories.
</description>
</property>
<property>
<name>fs.default.name</name>
<value>hdfs://localhost:54310</value>
<description>The name of the default file system. A URI whose
scheme and authority determine the FileSystem implementation. The
uri's scheme determines the config property (fs.SCHEME.impl) naming
the FileSystem implementation class. The uri's authority is used
to determine the host, port, etc. for a filesystem.</description   </property>
```

4. /usr/local/hadoop /conf /mapred-site.xml

```
hduser@manel:~$ sudo gedit  /usr/local/hadoop /conf /mapred-site.xml
```

```
<property>
<name>mapred.job.tracker</name>
<value>localhost:54311</value>
<description>The host and port that the MapReduce job tracker runsat. If "local", then jobs are
run in-process as a single map and reduce task.
</description>  </property>
```

5. /usr/local/hadoop/conf /hdfs-site.xml

```
hduser@manel:~$ sudo gedit /usr/local/hadoop/conf /hdfs-site.xml
```

Le fichier /usr/local/hadoop/conf/hdfs-site.xml doit être configuré pour chaque hôtedu cluster qui va l'utiliser. Il est utilisé pour spécifier les répertoires qui seront utilisés comme NameNode et DataNode sur cette hôte.

Ouvrez le fichier et entrez le contenu suivant entre <configuration></ configuration> :

```
<property>
<name>dfs.replication</name>
<value>1</value>
<description>Default block replication.
The actual number of replications can be specified when the file is
created.
```

```
The default is used if replication is not specified in create time. </description> </property>
```

Formater le nouveau système de fichier HDFS pour Hadoop :

```
hduser@manel:~$ /usr/local/hadoop/bin/hadoop namenode -format
```

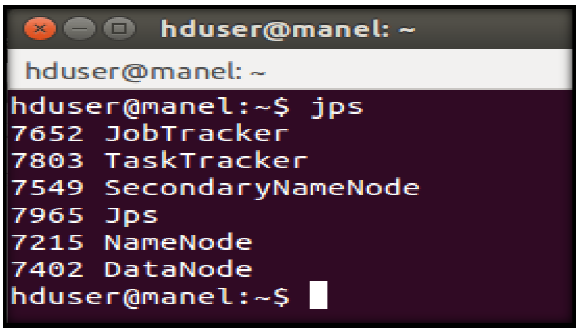
```
 has been successfully formatted.
15/05/26 06:49:08 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
```

II.8 Lancement d'Hadoop

Nous passons au démarrage du cluster Hadoop nouvellement installé. On peux lancer (start-dfs.sh et start-yarn.sh) ou start-all.sh.

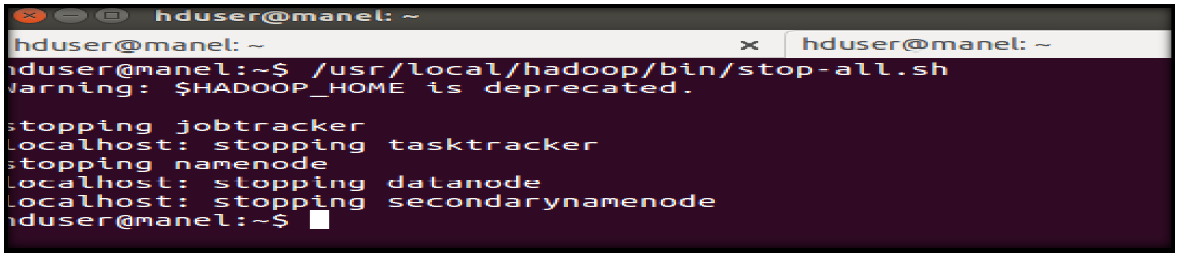
```
hduser@manel:~$ /usr/local/hadoop/bin/start-all.sh
```

Pour vérifier le bon fonctionnement d'Hadoop : On tape la commande : jps



II.9 Pour arrêter Hadoop

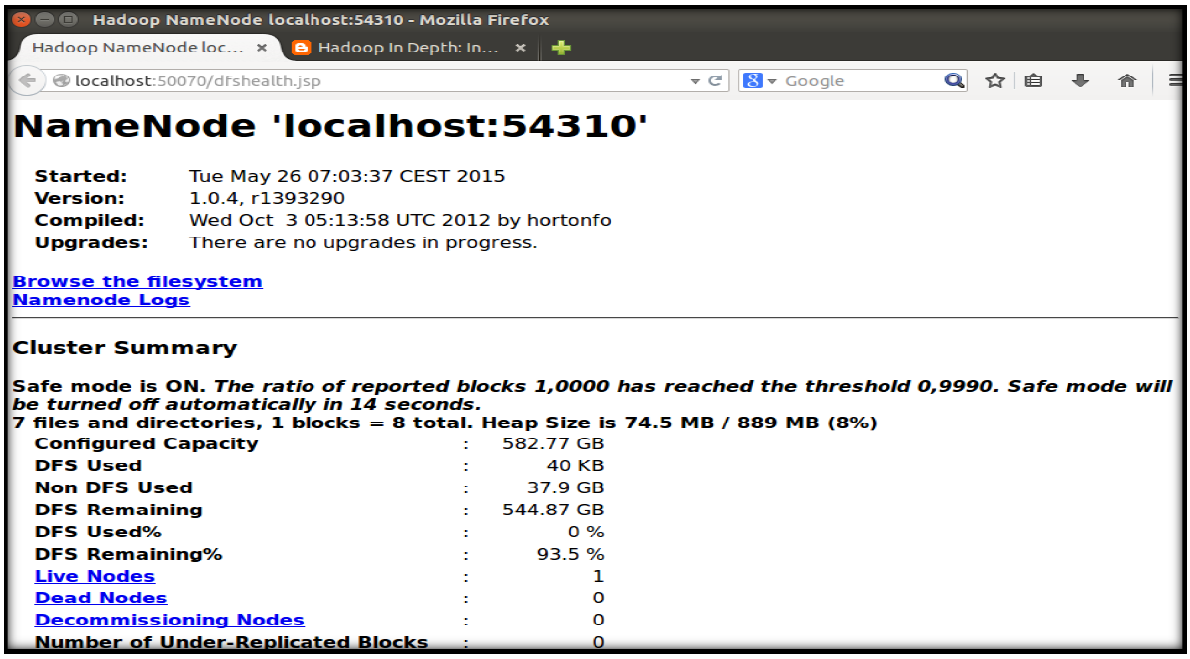
```
hduser@manel:~$ /usr/local/hadoop/bin/stop-all.sh
```



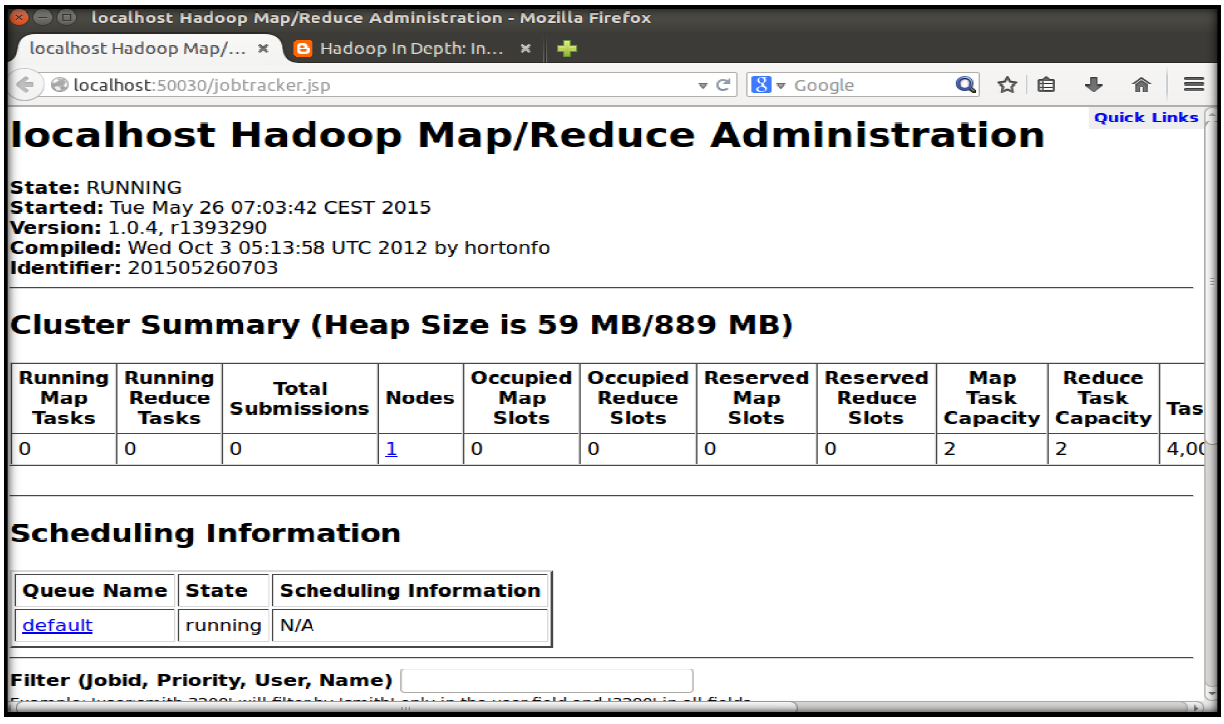
Par défaut, Hadoop est configuré avec des interfaces WEB afin de faciliter la gestion de ces composants :

- http://localhost:50070/ – Web UI pour le NameNode
- http://localhost:50030/ – Web UI pour le JobTracker
- http://localhost:50060/ – Web UI pour le TaskTracker

http://localhost:50070/ – Web UI pour le NameNode



http://localhost:50030/ – Web UI pour le JobTracker



II.10 Installation de HBASE

L'installation de HBASE est très facile à réaliser. Il faut choisir une version stable et conforme à la configuration disponible. Les étapes à suivre pour effectuer l'installation :

1- Télécharger HBASE, version stable, depuis le site :
http://www.apache.org/dyn/closer.cgi/hbase/.

2- Extraire le fichier téléchargé dans le chemin /usr/local, On exécute les commandes suivante :

```
hduser@manel:~$ cd /usr/local
hduser@manel:/usr/local$ sudo tar xzf hbase-0.94.8.tar.gz
hduser@manel:/usr/local$ sudo mv hbase-0.94.8 hbase
hduser@manel:/usr/local$ sudo chown -R hduser:hadoop hbase
```

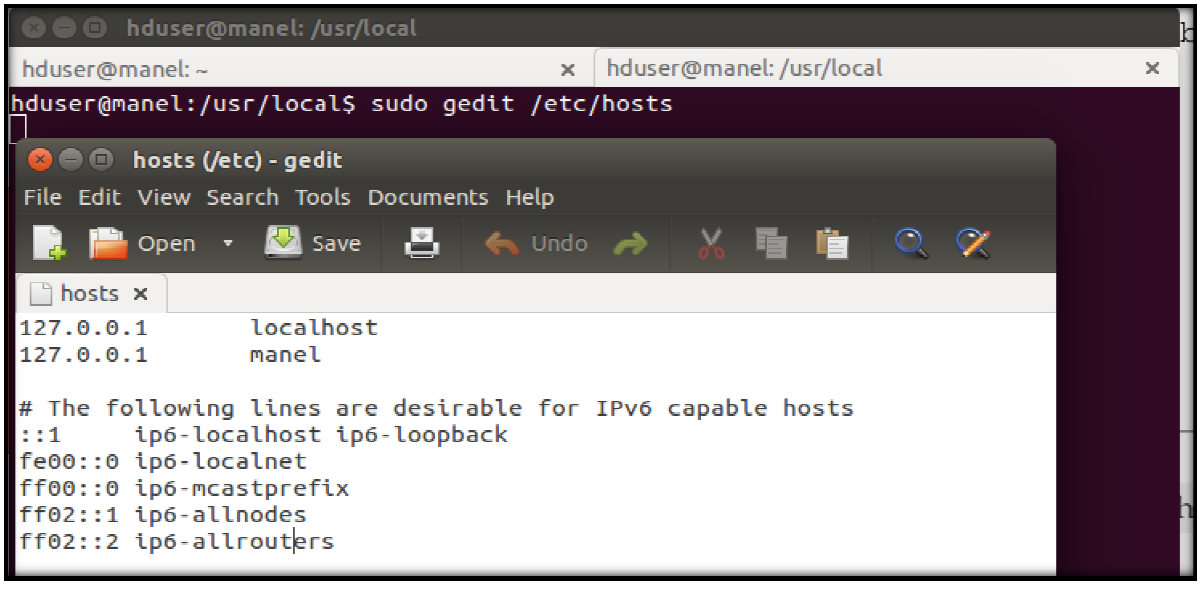
3- Modifier le fichier /usr/local/hbase/conf/hbase-site.xml

```
<configuration>
<property>
<name>hbase.cluster.distributed</name>
<value>true</value>
</property>
<property>
<name>hbase.rootdir</name>
<value>hdfs://localhost:54310/hbase</value>
<description>The directory shared by RegionServers.
</description>
</property>
<property>
<name>dfs.replication</name>
<value>1</value>
<description>The replication count for HLog and HFile
storage. Should not be greater
than HDFS datanode count.
</description>
</property>
</configuration>
```

4- Modifier le fichier /usr/local/hbase/conf/hbase-site.xml

```
<configuration>
<property>
<name>hbase.cluster.distributed</name>
<value>true</value>
</property>
<property>
<name>hbase.rootdir</name>
<value>hdfs://localhost:54310/hbase</value>
<description>The directory shared by RegionServers.
</description>
</property>
<property>
<name>dfs.replication</name>
<value>1</value>
<description>The replication count for HLog and HFile
storage. Should not be greater
than HDFS datanode count.
</description>
</property>
</configuration>
```

5- Il faut modifier les fichiers /etc/hosts pour éviter des erreurs :



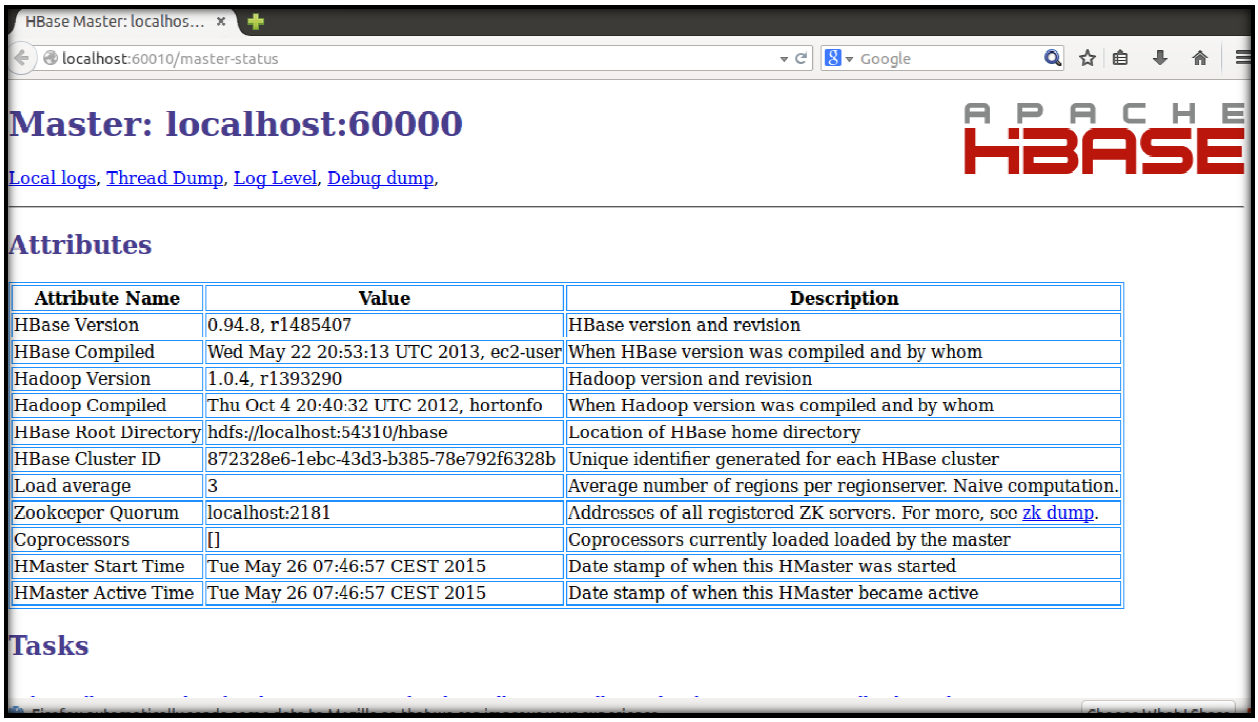
Modifier l'adresse IP de 127.0.1.1 manel à 127.0.0.1 manel

6- Lancer Hadoop avec la commande :

```
hduser@manel:~$ start-hbase.sh
```

7- La gestion des tables HBase se fait à l'aide de l'invite : hbase shell

L'interface WEB de HBASE est définit par l'adresse : http://localhost:60010



II.11 Installation de Mongo DB

L'installation de mongo dB est la plus simple. Il suffit de lancer la commande suivante :

```
hduser@manel:~$ sudo apt-get install mongodb-server
```

Après installation, on peut lancer la console mongo db par la commande :

```
hduser@manel:~$ mongo
```

II.12 Installation d'YCSB

- Télécharger la source YCSB avec la commande :

```
hduser@manel:~$ gitclone http://github.com/brianfrankcooper/YCSB.git
```

- Accéder au dossier YCSB et compiler avec la commande :

```
hduser@manel:~$ /YCSB/ mvn clean package
```

- Après la phase de compilation ; copier et extraire le fichier ycsb-0.1.4.tar.gz se trouvant dans le dossier /YCSB/distribution/ vers un emplacement de votre choix.

III. Deuxième partie : Tests et analyse des résultats

III.1 Evaluation de performance

Dans ce chapitre, nous allons présenté et analysé les résultats de chargement de 800 000 enregistrements générés par YCSB puis on va effectuer différents tests sous forme de charges de travail (Workloads) composées de 10 000 opérations. Le critère de performance principal sur lequel les comparaisons vont être faites, est le temps d'exécution, qui sera estimé par la moyenne des temps d'exécutions des Workloads obtenus pendant 3 journées différentes. Les 10 000 opérations seront réparties entre lecture, écriture, et mises à jour effectuées sur ces enregistrements.

III.2 Initialisation et présentation des Wokloads

Nous avons commencé par initialiser l'outil YCSB avec 8 Workloads afin de rendre les scénarios plus significatifs :

1. Workload A: Mise à jour lourde, se compose d'un rapport de 50% Read /50% Update.
2. Workload B: Lire Partiellement, Il consiste en un rapport de 95% Read /5% Update .
3. Workload C: Lecture seule, ce workload est 100% Read .
4. Workload D: Ce Workload se compose de 95% Read /5% Insert .
5. Workload E: Se compose d'un rapport de 95% Scan /5% Insert .
6. Workload F: Ce workload se compose d'un rapport de 50% Read /50% Read-Modify-Write.
7. Workload G: Se compose d'un rapport de 5% Read /95% Update .
8. Workload H: Ce workload est composé de 100% Update.

III.3 Chargement de données (LoadProcess)

Nous commençons par l'étape de e chargement de 800000 enregistrements générés par YCSB (Load Process).

1) **Hbase :** Taper la commande suivante :

```
bin/ycsb load hbase -P workloads/workloadi -p columnfamily=f1 -p recordcount=800000 -s > loadp.dat
```

Le résultat du test est le suivant :

```
10 sec: 32773 operations; 3277,3 current ops/sec; [INSERT AverageLatency(us)=279,54]
20 sec: 98321 operations; 6552,83 current ops/sec; [INSERT AverageLatency(us)=144,9]
30 sec: 140459 operations; 4213,8 current ops/sec; [INSERT AverageLatency(us)=169,65]
40 sec: 182597 operations; 4213,38 current ops/sec; [INSERT AverageLatency(us)=285,27]
50 sec: 236423 operations; 5382,06 current ops/sec; [INSERT AverageLatency(us)=193,1]
60 sec: 285601 operations; 4917,8 current ops/sec; [INSERT AverageLatency(us)=196,35]
70 sec: 360513 operations; 7490,45 current ops/sec; [INSERT AverageLatency(us)=125,09]
80 sec: 407333 operations; 4682 current ops/sec; [INSERT AverageLatency(us)=175,95]
90 sec: 480145 operations; 7280,47 current ops/sec; [INSERT AverageLatency(us)=160,5]
100 sec: 538429 operations; 5828,4 current ops/sec; [INSERT AverageLatency(us)=165,93]
110 sec: 580567 operations; 4213,38 current ops/sec; [INSERT AverageLatency(us)=227,61]
120 sec: 636751 operations; 5618,4 current ops/sec; [INSERT AverageLatency(us)=169,9]
130 sec: 683571 operations; 4681,53 current ops/sec; [INSERT AverageLatency(us)=197,51]
140 sec: 735073 operations; 5149,69 current ops/sec; [INSERT AverageLatency(us)=207,24]
150 sec: 772529 operations; 3745,23 current ops/sec; [INSERT AverageLatency(us)=262,02]
160 sec: 777211 operations; 468,2 current ops/sec; [INSERT AverageLatency(us)=415,76]
170 sec: 786575 operations; 935 current ops/sec; [INSERT AverageLatency(us)=1915,13]
172 sec: 800000 operations; 5175,4 current ops/sec; [INSERT AverageLatency(us)=148,55]
```

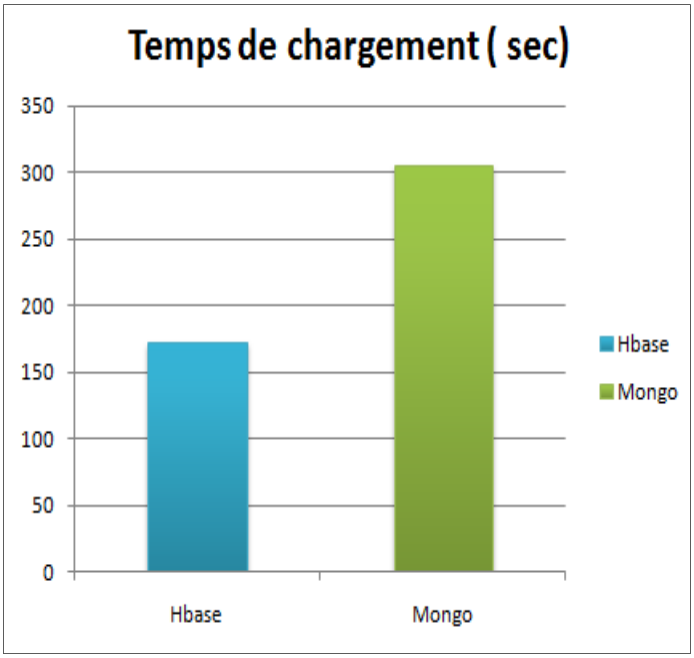
2) **MongoDB :** Taper la commande suivante :

```
bin/ycsb load mongodb -P workloads/workloada –p recordcount=800000 -s > loadm.dat
```

Le résultat du test est le suivant :

```
Loading workload...
Starting test.
 0 sec: 0 operations;
 10 sec: 16763 operations; 1642,47 current ops/sec; [INSERT AverageLatency(us)=533,08]
 20 sec: 48634 operations; 3172,82 current ops/sec; [INSERT AverageLatency(us)=309,38]
 30 sec: 73396 operations; 2475,95 current ops/sec; [INSERT AverageLatency(us)=389,56]
 40 sec: 101354 operations; 2795,8 current ops/sec; [INSERT AverageLatency(us)=358,3]
 50 sec: 132623 operations; 3126,59 current ops/sec; [INSERT AverageLatency(us)=313,45]
 60 sec: 163420 operations; 3079,39 current ops/sec; [INSERT AverageLatency(us)=318,32]
 70 sec: 194020 operations; 3060 current ops/sec; [INSERT AverageLatency(us)=320,38]
 80 sec: 224807 operations; 3078,39 current ops/sec; [INSERT AverageLatency(us)=318,43]
 90 sec: 248166 operations; 2335,9 current ops/sec; [INSERT AverageLatency(us)=322,07]
 100 sec: 261828 operations; 1366,06 current ops/sec; [INSERT AverageLatency(us)=895,98]
 110 sec: 292386 operations; 3055,49 current ops/sec; [INSERT AverageLatency(us)=320,93]
 120 sec: 322742 operations; 3035,6 current ops/sec; [INSERT AverageLatency(us)=323,07]
 130 sec: 353035 operations; 3029 current ops/sec; [INSERT AverageLatency(us)=323,74]
 140 sec: 383096 operations; 3005,8 current ops/sec; [INSERT AverageLatency(us)=326,28]
 150 sec: 406321 operations; 2322,5 current ops/sec; [INSERT AverageLatency(us)=324,75]
 160 sec: 421177 operations; 1485,6 current ops/sec; [INSERT AverageLatency(us)=822,35]
 170 sec: 450491 operations; 2930,81 current ops/sec; [INSERT AverageLatency(us)=334,6]
 180 sec: 479312 operations; 2881,81 current ops/sec; [INSERT AverageLatency(us)=339,84]
 190 sec: 509088 operations; 2977,6 current ops/sec; [INSERT AverageLatency(us)=329,96]
 200 sec: 538424 operations; 2933,6 current ops/sec; [INSERT AverageLatency(us)=334,42]
 210 sec: 561393 operations; 2296,67 current ops/sec; [INSERT AverageLatency(us)=326,26]
 220 sec: 576528 operations; 1513,5 current ops/sec; [INSERT AverageLatency(us)=810,45]
 230 sec: 605784 operations; 2925,31 current ops/sec; [INSERT AverageLatency(us)=335,06]
 240 sec: 634976 operations; 2918,91 current ops/sec; [INSERT AverageLatency(us)=336,4]
 250 sec: 659918 operations; 2494,2 current ops/sec; [INSERT AverageLatency(us)=393,57]
 260 sec: 689581 operations; 2966 current ops/sec; [INSERT AverageLatency(us)=330,68]
 270 sec: 711541 operations; 2196 current ops/sec; [INSERT AverageLatency(us)=343,01]
 280 sec: 726333 operations; 1479,05 current ops/sec; [INSERT AverageLatency(us)=825,88]
 290 sec: 756147 operations; 2981,4 current ops/sec; [INSERT AverageLatency(us)=329,97]
 300 sec: 785504 operations; 2935,41 current ops/sec; [INSERT AverageLatency(us)=334,27]
 305 sec: 800000 operations; 2965,02 current ops/sec; [INSERT AverageLatency(us)=330,54]
```

.



SGBDs Nosql	Temps d'exécution (sec)
Hbase	**172**
MongoDB	**305**

Tableau IV.1. Temps de chargement

Figure IV.1. Histogramme Temps de chargement

La Figure IV.1 montre les temps d'exécution pendant le chargement de 800 000 enregistrements dans les deux bases de données. Nous avons constaté que durant le chargement de 800 000 enregistrements, le meilleurs temps d'insertion a été fournit par HBASE avec un temps de chargement de seulement 172 seulement secondes contre 305 secondes pour MongoDB. Cela signifie que HBASE était plus rapide que MongoDB pendant la phase de chargement. La cause est le fait que HBASE ne nécessite pas de grande quantité de mémoire pendant l'exécution des opérations de chargement initial.

III.4 Exécution des Workloads

Dans la section suivante, on va présenter et analyser les résultats des tests. Un rapprochement des temps d'exécutions des Workloads effectuées sur 800 000 enregistrements, sera établi. Le nombre d'opérations est limité à 10 000 opérations.

1. **Workload A (50%/50% de Read / Update)**

- Test de Hbase : Taper la commande suivante :

```
bin/ycsb load hbase -P workloads/workloada -p columnfamily=f1 -p
recordcount=800000 -s > loadp.dat
```

Le résultat du test est le suivant :

```
Engine.java:320)
        at org.apache.hadoop.hbase.ipc.HBaseServer$Handler.run(HBaseServer.java:
1426)
: 4981 times, servers with issues: localhost:60020,
        at org.apache.hadoop.hbase.client.HConnectionManager$HConnectionImplemen
tation.processBatchCallback(HConnectionManager.java:1677)
        at org.apache.hadoop.hbase.client.HConnectionManager$HConnectionImplemen
tation.processBatch(HConnectionManager.java:1453)
        at org.apache.hadoop.hbase.client.HTable.flushCommits(HTable.java:936)
        at com.yahoo.ycsb.db.HBaseClient.cleanup(HBaseClient.java:101)
        ... 2 more
 15 sec: 10000 operations; 599,82 current ops/sec; [UPDATE AverageLatency(us)=50
,48] [READ AverageLatency(us)=2297,65]
```

Le journal d'exécution du Workload A par HBASE :

```
YCSB Client 0.1
Command line: -db com.yahoo.ycsb.db.HBaseClient -P workloads/workloada -p columnfamily=f1 -p recordcount=800000 -s -t
[OVERALL], RunTime(ms), 8775.0
[OVERALL], Throughput(ops/sec), 1139.6011396011395
[UPDATE], Operations, 4928
[UPDATE], AverageLatency(us), 67.32508116883118
[UPDATE], MinLatency(us), 16
[UPDATE], MaxLatency(us), 13015
[UPDATE], 95thPercentileLatency(ms), 0
[UPDATE], 99thPercentileLatency(ms), 0
[UPDATE], Return=0, 4928
[UPDATE], 0, 4923
[UPDATE], 999, 0
[UPDATE], >1000, 0
[READ], Operations, 5072
[READ], AverageLatency(us), 1323.1671924290222
[READ], MinLatency(us), 713
[READ], MaxLatency(us), 1276928
[READ], 95thPercentileLatency(ms), 1
[READ], 99thPercentileLatency(ms), 4
[READ], Return=0, 5072
[READ], >1000, 1
```

- Test de MongoDB

Taper la commande suivante :

```
bin/ycsb load mongodb -P workloads/workloada -p recordcount=800000 -s > loadm.da
```

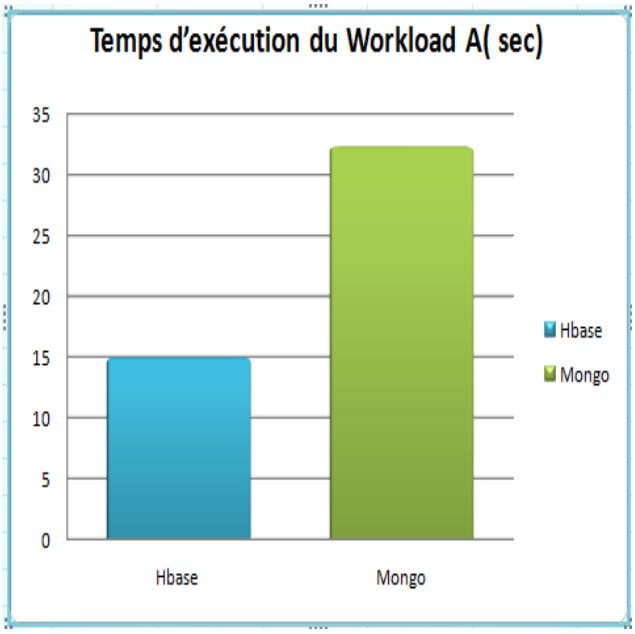
Le résultat du test est le suivant :

```
Loading workload...
Starting test.
 0 sec: 0 operations;
 10 sec: 3348 operations; 333,57 current ops/sec; [UPDATE AverageLatency(us)=3282,12] [READ AverageLatency(us)=2430]
 20 sec: 5350 operations; 200,12 current ops/sec; [UPDATE AverageLatency(us)=6811,11] [READ AverageLatency(us)=3258,59]
 30 sec: 9722 operations; 437,16 current ops/sec; [UPDATE AverageLatency(us)=2814,88] [READ AverageLatency(us)=1703,94]
 30 sec: 10000 operations; 416,17 current ops/sec; [UPDATE AverageLatency(us)=2919,38] [READ AverageLatency(us)=1770,34]
```

Le journal de l'exécution du Workload A par MongoDB :

```
YCSB Client 0.1
Command line: -db com.yahoo.ycsb.db.MongoDbClient -P workloads/workloada -p recordcount=800000 -s -t
new database url = localhost:27017/ycsb
mongo connection created with localhost:27017/ycsb
[OVERALL], RunTime(ms), 12911.0
[OVERALL], Throughput(ops/sec), 774.5333436604445
[UPDATE], Operations, 4935
[UPDATE], AverageLatency(us), 1481.0342451874367
[UPDATE], MinLatency(us), 312
[UPDATE], MaxLatency(us), 75425
[UPDATE], 95thPercentileLatency(ms), 7
[UPDATE], 99thPercentileLatency(ms), 20
[UPDATE], Return=0, 3817
[UPDATE], Return=1, 1118
[UPDATE], 0, 4418
[UPDATE], 999, 0
[UPDATE], >1000, 0
[READ], Operations, 5065
[READ], AverageLatency(us), 955.9855873642646
[READ], MinLatency(us), 205
[READ], MaxLatency(us), 55653
[READ], 95thPercentileLatency(ms), 1
[READ], 99thPercentileLatency(ms), 14
[READ], Return=0, 3909
[READ], Return=1, 1156
[READ], 0, 4783

[READ], 999, 0
[READ], >1000, 0
```



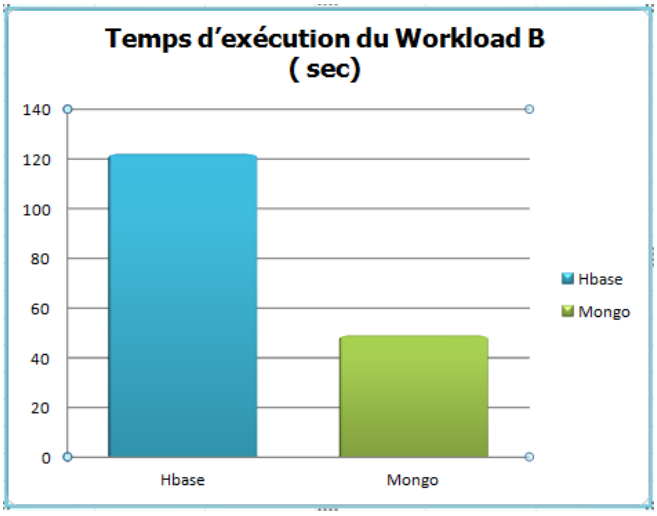
SGBDs Nosql	Temps d'execution (sec)
Hbase	**15**
MongoDB	**32.33**

Tableau IV.2. Temps de chargement

Figure IV.2. Histogramme Temps d'exécution du Workload A

La Figure IV.2. Montre les résultats obtenus après l'exécution du Workload A, qui se compose de 50%d'opérations Read et 50% Update, Nous remarquons que Hbase et plus rapide par rapport à MongoDB.

2. Workload B (95%/5% de Read / Update)



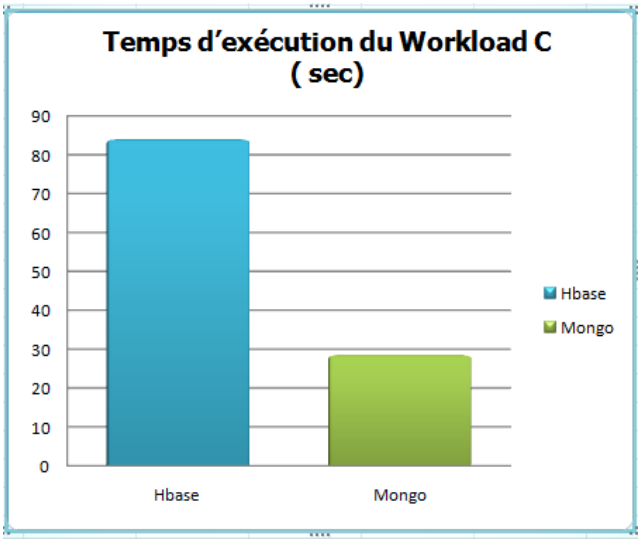
SGBDs Nosql	Temps d'execution (sec)
Hbase	**122**
MongoDB	**49**

Tableau IV.3. Temps d'exécution du Workload B

Figure IV.3. Histogramme Temps d'exécution du Workload B.

La Figure IV.3 montre les résultats obtenus lors de l'exécution du Workload B. Nous avons remarqué que MongoDB a donné de meilleurs résultats par rapport à HBASE. La rapidité de MongoDB pendant les opérations de lecture est justifiée par l'absence d'optimisation pour exécuter ce genre d'opération dans Hbase.

3. Workload C (100% Read)



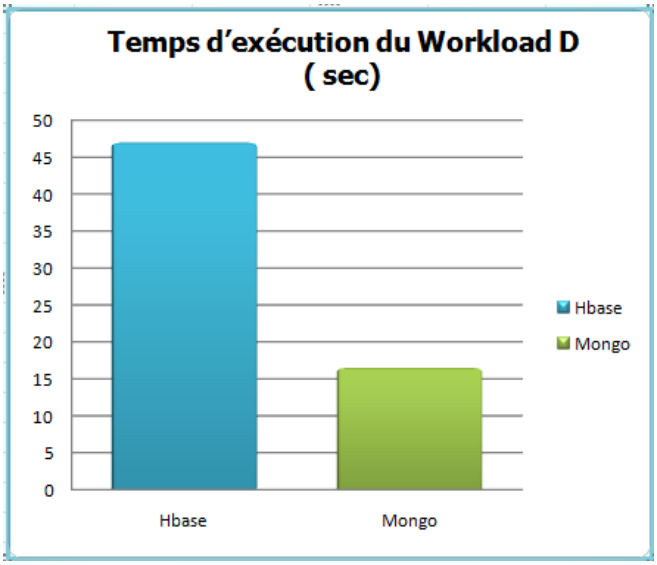
SGBDs Nosql	Temps d'execution (sec)
Hbase	84
MongoDB	28.5

Tableau IV.4. Temps d'exécution du Workload C

Figure IV.4. Histogramme Temps d'exécution du Workload C

La figure IV.4 montre les résultats obtenus lors de l'exécution du Workload C. Les indiquent que MongoDB est plus rapide que HBASE qui a montré un temps d'exécution le plus lent pendant les opérations de lecture.

4. Workload D (95% Read /5% Insert)



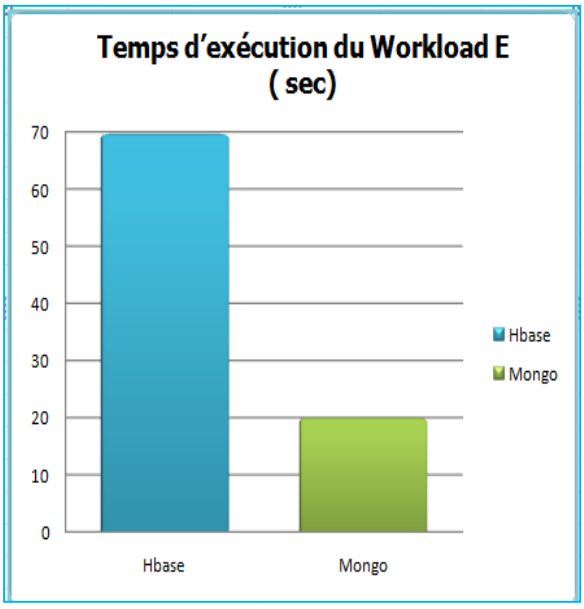
SGBDs Nosql	Temps d'execution (sec)
Hbase	47
MongoDB	16.5

Tableau IV.5. Temps d'exécution du workload D

Figure IV.5. Histogramme Temps d'exécution du workload D

HBASE est toujours lent pendant l'exécution des opérations de lecture (Figure IV.5), MongoDB est plus performant.

5. Workload E (95% Scan /5% Insert)



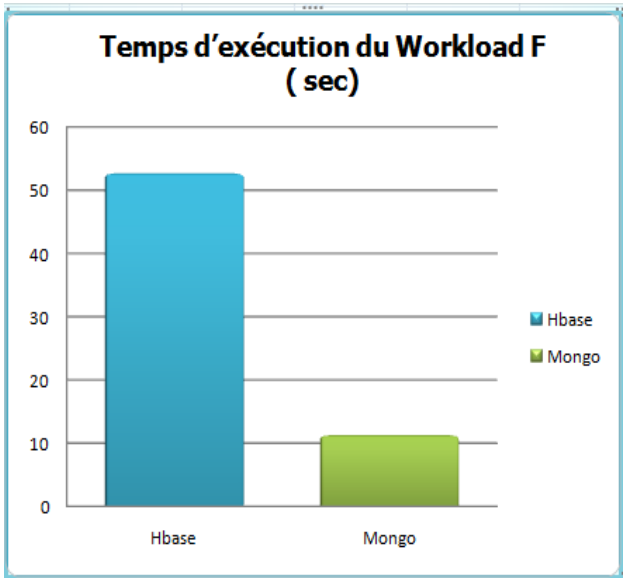
SGBDs Nosql	Temps d'execution (sec)
Hbase	69.66
MongoDB	20

Tableau IV.6. Temps d'exécution du Workload E

Figure IV.6. Histogramme Temps d'exécution du Workload E

HBASE reste lent pendant l'exécution de Scan par rapport à MongoDB (Figue IV.6)

6. Workload F (50% Read /50% Read-Modify-Write)



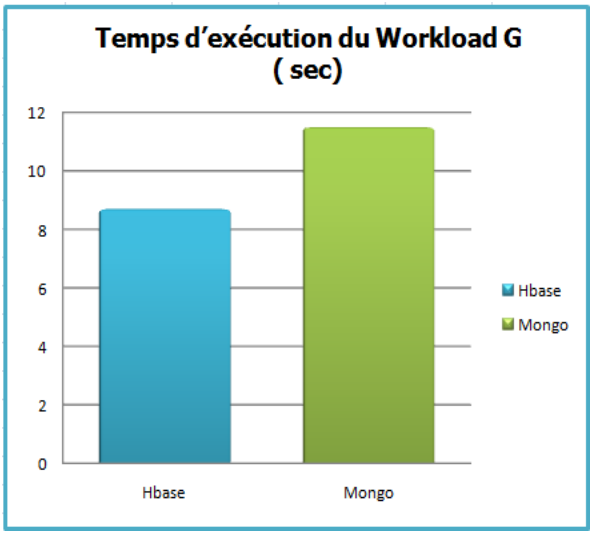
SGBDs Nosql	Temps d'execution (sec)
Hbase	52.66
MongoDB	11.25

Tableau IV.7. Temps d'exécution de Workload F

Figure IV.7. Histogramme Temps d'exécution de Workload F

Dans ce Workload les enregistrements sont lus en premier lieu, mis à jour après et ensuite sauvegardés. Nous constatons que HBASE a montré une performance inférieure en raison de ca difficulté de lecture par rapport à la mise à jour.

7. Workload G (5% Read /95% Update)



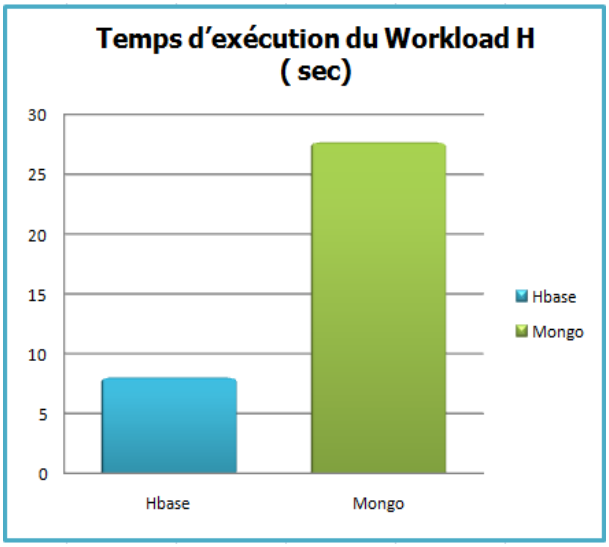
SGBDs Nosql	Temps d'execution (sec)
Hbase	8.5
MongoDB	11.5

Tableau IV.8. Temps d'exécution du Workload G

Figure IV.8. Histogramme Temps d'exécution du Workload G.

HBASE a montré une meilleure performance grâce à son utilisation d'un journal où toutes les transactions sont écrites en mode ajout au départ

8. Workload H (100% Update)



SGBDs Nosql	Temps d'execution (sec)
Hbase	**8**
MongoDB	**27.66**

Tableau IV.9. Temps d'exécution du Workload H

Figure IV.9. Histogramme Temps d'exécution du Workload H

La Figure IV.9 montre les résultats de l'exécution du Workload H (100% update) avec 10 000 opérations de mises à jour dans une base de données de 800 000 enregistrements. HBASE a montré un meilleur résultat par rapport à MongoDB. Cela montre la performance de HBASE dans les opérations de mise à jour.

9. Schéma global

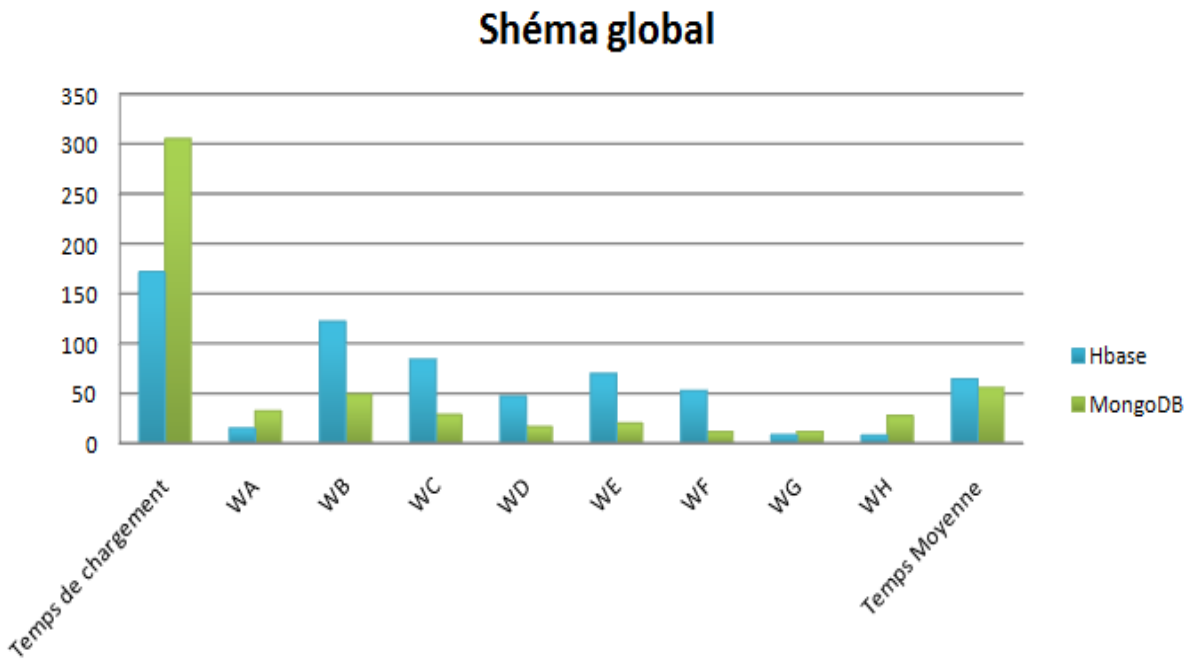


Figure IV.10. Récap des temps d'exécutions

La Figure IV.10 regroupe les temps d'exécutions de chargement des 800 000 enregistrements et ceux des Workloads composées de 10 000 opérations ainsi que le temps moyen global de tous les tests effectués

III.5 Evaluation globale de HBASE et MongoDB

Après lecture des résultats obtenus par les deux systèmes, plusieurs enseignements peuvent être tirés :

- HBase a montré sa performance dans les opérations d'insertion et mise à jour.

- La différence entre les temps d'exécutions durant le chargement des 800 000 enregistrements est justifiée par le fait que HBase ne nécessite pas un grand espace mémoire pour exécuter des opérations de chargement initial.

- HBase utilise un journal pour conserver toutes les modifications effectuées, ce qui accélère le processus de mise à jour.

- MongoDB a été plus performant dans les opérations de lecture et de scan. Ceci est du, principalement, à la cartographie des registres de MongoDB chargée en mémoire qui améliore ces performances en lecture.

- L'exécution d'une opération de mise à jour dans MongoDB, est ralentit relativement par rapport aux nombres de mises à jour appliquées. Elle utilise des mécanismes de verrouillage qui alourdissent le temps d'exécution de mise à jour.

IV. Conclusion

Les tests réalisés dans ce chapitre, ont contribué à dévoiler des résultats intéressants sur les performances de deux types de bases de données NoSQL. La première base de données est une base orientée colonne (HBASE), la deuxième est une base orientée document (MongoDB). La diversité des charges de travail réservée aux tests, a enrichi la base de résultats et a permis une meilleure analyse. On peut affirmer que MongoDB est globalement plus performante par rapport à HBASE pendant l'exécution des opérations de lecture, Scan et lecture/écriture. HBASE de son coté, était plus performante dans les opérations d'insertion et la mise à jour.

Conclusion et perspectives

Conclusion générale

Dans les prochaines années, nous estimons que les technologies des Big Data seront de plus en plus utilisées pour répondre à de nouvelles problématiques pour la gestion de données dans des environnements à grande échelle. Les systèmes NoSQL se placent comme les solutions idéales pour gérer les grands volumes de données, la variété de leurs types caractérisant principalement le concept du Big Data.

Plusieurs solutions NoSQL basées sur des architectures différentes sont développées dans tous les secteurs. La tendance vers une favorisation d'une solution NoSQL précise est loin d'être réalisée à cause du nombre important des solutions existantes et l'absence de normalisation. Actuellement, plusieurs solutions open-source et payantes sont présentées aux acteurs des différents secteurs liés au Big Data, qui mettent les utilisateurs devant un embarras dans le choix du modèle approprié par rapport à leur environnement d'exploitation.

L'objectif ciblé de ce PFE consistait à comparer les performances de deux différents types de bases de données NoSQL, à savoir une orientée colonne (HBASE), et une orientée document (MongoDB).

Après l'étude menée, nous pouvons affirmer que le choix d'utilisation d'un modèle ou un autre, parmi les solutions disponibles dans le marché, dépend de l'environnement dans lequel les données sont exploitées. En effet le type de données et leur traitement sont des indicateurs importants pour définir la ou les solutions utilisées. La fréquence estimée de lecture, d'écriture et de mise à jour ainsi que la taille des données sont les facteurs essentiels pour la sélection de la solution NoSQL appropriée.

Perspectives

Plusieurs travaux de recherche se focalisent actuellement sur le sujet qui se voit très vaste et très ouvert, d'autres comparaisons entre les différentes familles des solutions NoSQL, orientée document, orientée colonne, clé valeur et orientée graphe, dans les environnements Cloud, sont toujours très sollicitées et recommandées pour apporter l'assistance et l'aide nécessaire aux intéressés de Big Data et de Cloud Computing, pour des éventuelles prises de décision sur les solutions convenables.

Conclusion et perspectives

Notons aussi, que notre étude comparative s'est articulée sur une architecture Single Node à cause des contraintes hardware, les tests et les évaluations des performances dans une architecture distribuée en multipliant les nœuds progressivement dans une plateforme High-Performance-Computing (HPC), peuvent faire l'objet d'autres sujets de recherche dans le proche futur.

Références Bibliographiques

[1] François EXERTIER, Extension orientée objet d'un SGBD relationnel, Mémoire du doctorat spécialité informatique, Université Joseph-Fourier - Grenoble 1, 1991, France.

[2] Olivier Losson, Introduction aux Systèmes de Gestion de Bases de Données Relationnelles, cours Master Sciences et Technologies, Université Lille1.

[3] Pierre Stockreiser, LES SYSTEMES DE GESTION DE BASES DE DONNEES, cour PDF, lycée du nord, Septembre 2006

[4] KOUEDI Emmanuel, Approche de migration d'une base de données relationnelle vers une base de données NoSQL orientée colonne, Mémoire master informatique, UNIVERSITE DE YAOUNDE I, mai 2012.

[5] Soufiane RITAL, Not only SQL Introduction, cours pour TELECOM ParisTech, Institut TELECOM CNRS LTCI, paris,2012.

[6] Adriano Girolamo PIAZZA, NOSQL, Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES, Haute École de Gestion de Genève, 9 octobre 2013.

[7] Xavier MALETRAS, Le NoSQL- Cassandra,Thèse Professionnelle, université paris 13, 27/05/2012

[8] Matteo DI MAGLIE, Adoption d'une solution NoSQL dans l'entreprise, Haute École de Gestion de Genève (HEG-GE) Filière Informatique de Gestionn, Carouge, 12 septembre 2012.

[9] Lionel HEINRICH,not only SQL, en vue de l'obtention du Bachelor HES en Informatique de Gestionnn, Genève, le 26 octobre 2012 .

[10] Meyer Léonard, l'avenir du NOSQL, université paris ,2014.

[11] Big data et NoSQL, http://administration-systeme.blogspot.com/?view=classic, visité en 01/03/2015 .

[12] Azhi Faraj , Bilal Rashid ,Twana Shareef, étude comparative entre les base de donnée relationnelle et non relationnelle , Mémoire du master spécialité informatique, Université University, Sulaymaniyah, Iraq, 11/11/2014.

[13] Nosql et concept, 2012.

Références Bibliographiques

[14] Mickaël CORINUS, Rapport d'étude sur le Big Data, 2012

[15] Charly CLAIRMONT, Introduction a HBase Base orientée colonnes au-dessus d'Hadoop, HUG France SL ,2013 – Mai 2013.

[16] HBase la base NoSQL de Hadoop : Concepts & Architecture, 21 septembre 2013

[17] Acef mohamed, utilisation du modele ma educe dans les differents systemes nosql: etude comparative, vuniversite d'oran , mars 2015 .

[18] Aurélien Foucret , Nosql Smile open source

[19] TechTarget, Tout savoir sur Hadoop : Vulgarisation de la technologie et les stratégies de certains acteurs, 2014

[20] Le benchmark 1 : YCSB, http://blog.ippon.fr/2015/03/11/mongodb-v3-la-revolution-22/ visité en 24/04/2015.

Liste des Figures

Liste des Tableaux

– Résumé –

Les SGBDs relationnelles ont pris une place importante sur le marché des SGBD, Ce modèle bien que très puissant, a atteint ses limites pour les environnements distribué. Les bases NoSQL proposent une alternative ou compléter les fonctionnalités des SGBDR pour donner des solutions plus intéressantes dans certains contextes.

Le but de ce travail est d'expliquer l'architecture des différentes bases NoSQL disponibles sur le marché. Et de voir d'une façon plus approfondi deux d'entre elles, HBASE et MongoDB, pour arriver à une analyse de performance utile en utilisant outille YCSB pour. La finalité est de connaître les performances de chacun de deux bases de donnés et d'effectuer une comparaison à partir des résultats des tests générés par YCSB.

Mots-clés : SGBDR, NoSQL, SGBDR, YCSB, HBASE, MongoDB.

–Abstract –

Relational DBMSs have taken an important place on the DBMS market. This model has reached its limits for large scale environments. NoSQL databases offer an alternative to RDBMS to provide more effective solutions. The purpose of this work is to explain the architecture of the different NoSQL databases available on the market and see a greater depth two of them, HBase and MongoDB. One of the two models performance analyses will rely on YCSB benchmark. The aim is to evaluate the performance of each of the two solutions and to make a comparison based on test results generated by YCSB to choose the appropriate solution.

Key-Words: DBMS,NoSQL, Relational DBMSs, YCSB, HBASE , MongoDB.

-الملخص-

وقد اتخدت القوعد البينات االعلائقية SGBDR مكانا هاما في السوق. هذا النموذج، على الرغم من أن قوية جداوصلت حدوده قواعد بيانات وقد تقد نظام قواعد البينات غير العلاقية Nosql مكملا لوظيفة قواعد لبينات العلائقية لتوفير حلول أكثر إثارة للاهتمام في بعض السياقات.والغرض من هذا العمل هو شرح بنية قواعد البيانات NoSQL المختلفة المتاحة في السوق و رؤية أعمق اثنين منهم، HBase و MongoDB . واحد من النموذجين و تحليل الأداء تعتمد على YCSB المعيار الهدف من ذلك هو تقييم أداء كل واحد من الحلين و إجراء مقارنة على أساس نتائج الاختبارات التي تم إنشاؤها بواسطة YCSB لاختيار الحل المناسب.

الكلمات المفتاحية: قوعد البينات العلائقية , قواعد البينات غير علائقية , اداة التحليل و المقارنة YCSB

-