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 :

MongoDb, CouchBase, Cassandra, HBase, Redis, OrientDB

Réalisé par :

- **BENALLAL Zeyneb**
- **TAHRAOUI Hayet**

Présenté le 05 Juin 2016 devant le jury composé de MM.

- *Mr SMAHI Mohammed Ismail* *(Président)*
- *Mr MATALLAH Houcine* *(Encadreur)*
- *Mme El YEBDRI Zeyneb* *(Examinatrice)*
- *Mme HALFAOUI Amel* *(Examinatrice)*

Année universitaire: 2015-2016

Remerciements

Nous tenons tout d'abord à remercier Dieu le tout puissant et miséricordieux, qui nous a donné la force et la patience et le courage d'accomplir ce Modeste travail.

Et c'est avec une immense reconnaissance que nous tenons à remercier notre encadreur Mr «MATALLAH Houcine» pour ses précieux conseils et sa disponibilité tout au long de la réalisation de ce travail, ainsi pour l'orientation, la confiance, l'aide et le temps qu'il a bien voulu nous consacrer.

On lui présente donc nos sentiments de gratitude.

Nos vifs remerciements vont également aux membres du jury pour l'intérêt qu'ils ont porté à notre recherche en acceptant d'examiner notre travail et de l'enrichir par leurs propositions.

Nos sincères remerciements à tous nos professeurs du département d'informatique de la faculté des sciences à Tlemcen

Enfin, nous tenons également à remercier toutes les personnes qui ont participé de près ou de loin à la réalisation de ce travail.

Dédicaces

C’est avec mon énorme plaisir, un cœur ouvert et une joie immense, que je dédie ce modeste travail tout d’abord à mes parents pour leurs amour, leurs sacrifices et leurs encouragements qui ont fait de moi ce que je suis aujourd’hui.

Vous présentez pour moi le symbole de la bonté, la source de tendresse et l’exemple du dévouement. Vos prières et vos bénédictions m’ont été d’un grand secours pour bien mener mes études. Aucune dédicace ne saurait être assez éloquente pour exprimer ce que vous méritez pour tous vos sacrifices.

Je vous dédie ce travail en témoignage de mon profond amour.

A ma sœur Fatima Zohra qui n’a jamais cessé de m’encourager pour qui je souhaite beaucoup de réussite, de bonheur et prospérité.

A mes frères Sofiane, Djawed, et Salim ainsi que leurs petites familles, merci pour tout ce que vous avez fait pour moi, Je vous dédie ce travail avec tous mes vœux de bonheur, de santé et de Réussite.

A mon binôme Zyneb avec qui j’ai partagé les joies et les difficultés durant ce projet merci pour tous.

A mes chères Nesrine et Sabah merci pour votre amitié. Vous étiez toujours disponible pour m’écouter. Merci

Aux personnes qui m'ont encouragé et motivé, qui n'ont cessé d'œuvré pour ma réussite et pour mon bonheur.

TAHRAOUI Hayet

Dédicaces

Ce que personne ne peut compenser les sacrifices qu'ils ont consentis pour mon éducation et pour mon bien être, qui n'ont jamais cessé de me soutenir moralement :

C'est à vous mes chers parents, la source de mes efforts, que je dédie ce modeste travail.

A mon mari Zine El Abidine, mon soutien moral et ma source de joie et de bonheur. Aucun mot ne saurait t'exprimer ma reconnaissance pour l'amour, la tendresse et la gentillesse dont tu m'as toujours entouré.

J'aimerai bien que tu trouve dans ce travail l'expression de mes sentiments de reconnaissance les plus sincères car grâce à ton aide et à ta patience avec moi que ce travail a pu voir le jour... Que dieu le tout puissant nous accorde un avenir meilleur.

A ma belle famille : Je vous dédie ce travail avec tous mes vœux de bonheur et de santé.

A mes frères et mes sœurs, qui m'ont toujours aidé et encouragé, qui étaient toujours à mes côtés avec leurs précieux conseils.

C'était avec toi mon binôme « Hayet » que j'ai partagé les bons moments pour que nous puissions achever notre mémoire dans les meilleures conditions.

A tous mes amis sans exception, et à tous qui me connaisse, et m'aime.

Je vous dis merci.

A toute personne qui a contribué de prés ou de loin à la réalisation de ce mémoire.

BENALLAL *Zyneb*

Table Des Matières

Chapitre III : Les solutions NoSQL étudiées

Chapitre IV : L'étude comparative

Introduction générale

Contexte

Les bases de données relationnelles ont été développées comme une technologie pour stocker des données structurées et organisées sous forme de tableaux. Aux cours des années elles sont devenues l'élément essentiel des organisations et le modèle de référence pour la gestion des données des systèmes d'informations, cependant, avec l'augmentation continue des données stockées et analysées, les bases de données relationnelles commencent à présenter une variété de limitations.

C'est dans ce contexte que les bases de données NoSQL étaient développées pour fournir un ensemble de nouvelles fonctionnalités de gestion des données tout en surmontant certaines limites de bases de données relationnelles.

Problématique

Actuellement, plusieurs utilisateurs des SGBD classiques dits « SQL » veulent basculer vers les nouvelles technologies NoSQL, en revanche, l'apparition de différents modèles NoSQL qui fournissent de nouvelles fonctionnalités d'une part et l'absence de normalisation des solutions disponibles dans le marché, impose une question très pertinente: quelle solution NoSQL à adopter ?

Contribution

Pour apporter les réponses nécessaires, Nous avons proposé, dans le cadre de ce projet de fin d'études, une étude comparative des bases de données NoSQL, pour orienter les utilisateurs vers les solutions les plus appropriées selon leurs besoins.

Autrement dit, il s'agit de présenter une évaluation expérimentale de six bases de données NoSQL très populaires, 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 mémoire

Notre manuscrit est organisé comme suit :

- Le premier chapitre consiste à rappeler les concepts de base du modèle relationnel en première partie, la deuxième partie est consacrée aux limites de cette architecture.
- Le deuxième chapitre, va être consacré à la technologie NoSQL, les différents types de bases de données NoSQL, ainsi que les avantages offerts par rapport aux bases de données relationnelles.

- Dans le troisième chapitre, nous allons présenter une fiche descriptive des solutions étudiées.
- Le quatrième chapitre fera l'objet des différents résultats expérimentaux obtenus après l'exécution d'un ensemble de tests ainsi qu'une interprétation des résultats d'évaluation des performances de chaque base de données.
- Enfin, nous clôturons cette étude par une conclusion générale sur le travail développé ainsi que quelques perspectives.

CHAPITRE I

Bases de données relationnelles et leurs limites

I. Introduction

Les données sont apparues avec les programmes. La gestion des données est l'une des premières raisons pour lesquelles les sociétés se sont intéressées à l'informatique.

Avec la gestion automatique des données, l'entreprise peut évoluer et rivaliser avec ses concurrents à l'aide de nouveaux moyens. Il n'est donc pas surprenant que les technologues d'entreprise conscients aient ciblé très tôt le marché de la gestion des données.

Avant la naissance du concept des bases de données objet, la théorie relationnelle des données proposée par le Docteur E. F.Codd apparaissait dans les applications commerciales de base de données relationnelle. Au milieu des années 80, une conviction quasi fanatique dans certains domaines du secteur informatique soutenait que tous les problèmes théoriques des données étaient désormais résolus et que les problèmes pratiques le seraient bientôt également. Manifestement, ce n'était pas le cas[1].

Les bases de données relationnelles constituent l'objet de ce chapitre, Après une courte introduction nous présenterons quelques principes fondamentaux des BDDR Nous décrivons ensuite les propriétés ACID sur lesquelles s'appuie ce type de base. Et enfin les limites qui peut impacter grandement la performance des bases.

II. Bases de données relationnelles

Le modèle relationnel a été introduit pour la première fois par Ted Codd du centre de recherche d'IBM en 1970 dans un papier désormais classique, et attira immédiatement un intérêt considérable en raison de sa simplicité et de ses fondations mathématique [2].

Dans ce modèle, les données sont représentées par des tables, sans préjuger de la façon dont les informations sont stockées dans la machine. Les tables constituent donc la structure logique du modèle relationnel [3], d'autre part ces données sont gérées par un système de gestion de bases de données relationnelle (relational database management system, en anglais), qui représente un système intégré pour la gestion unifiée des bases de données relationnelles, il est constitué d'un composant de stockage et d'un composant de gestion de données [4].

L'interface standard pour une base de données relationnelle est le langage SQL (Structured Query Language) considéré comme le langage de manipulation des données relationnelles le plus utilisé aujourd'hui. Il est devenu un standard de fait pour les SGBD relationnels. Il possède des caractéristiques proches de l'algèbre relationnelle (jointures, opérations ensemblistes) et d'autres proches du calcul des tuples (variables sur les relations). SQL est un langage redondant qui permet souvent d'écrire les requêtes de plusieurs façons différentes [5].

III. Propriétés ACID

Selon la théorie des bases de données, les propriétés ACID sont les quatre principaux attributs d'une transaction de données. Il s'agit là d'un des concepts les plus anciens et les plus importants du fonctionnement des bases de données, il spécifie quatre buts à atteindre pour toute transaction.

Ces buts sont les suivants :

III.1 Atomicity (Atomicité)

Lorsqu'une transaction est effectuée, toutes les opérations qu'elle comporte doivent être menées à bien : en effet, en cas d'échec d'une seule des opérations, toutes les opérations précédentes doivent être complètement annulées, peu importe le nombre d'opérations déjà réussies. En résumé, une transaction doit s'effectuer complètement ou pas du tout. Voici un exemple concret : une transaction qui comporte 3000 lignes qui doivent être modifiées, si la modification d'une seule des lignes échoue, alors la transaction entière est annulée. L'annulation de la transaction est toute à fait normale, car chaque ligne

ayant été modifiée peut dépendre du contexte de modification d'une autre, et toute rupture de ce contexte pourrait engendrer une incohérence des données de la base.

III.2 Consistancy (Cohérence)

Avant et après l'exécution d'une transaction, les données d'une base doivent toujours être dans un état cohérent. Si le contenu final d'une base de données contient des incohérences, cela entraînera l'échec et l'annulation de toutes les opérations de la dernière transaction. Le système revient au dernier état cohérent. La cohérence est établie par les règles fonctionnelles.

III.3 Isolation (Isolation)

La caractéristique d'isolation permet à une transaction de s'exécuter en un mode isolé. En mode isolé, seule la transaction peut voir les données qu'elle est en train de modifier, c'est le système qui garantit aux autres transactions exécutées en parallèle une visibilité sur les données antérieures. Ce fonctionnement est obtenu grâce aux verrous système posés par le SGBD. Prenons l'exemple de deux transactions A et B : lorsque celles-ci s'exécutent en même temps, les modifications effectuées par A ne sont ni visibles, ni modifiables par B tant que la transaction A n'est pas terminée et validée par un « commit ».

III.4 Durability (Durabilité)

Toutes les transactions sont lancées de manière définitive. Une base de données ne doit pas afficher le succès d'une transaction pour ensuite remettre les données modifiées dans leur état initial. Pour ce faire, toute transaction est sauvegardée dans un fichier journal afin que, dans le cas où un problème survient empêchant sa validation complète, elle puisse être correctement terminée lors de la disponibilité du système [6].

IV. Limites des bases de données relationnelles

Les bases de données existent maintenant depuis environ 56 ans et le modèle relationnel depuis environ 46 ans, pendant plusieurs décennies, ce modèle bien très puissant, représentait la solution parfaite pour les différents acteurs dans le domaine de gestion des données, néanmoins ces architectures ont atteints leurs limites pour certains services ou sites manipulant de grandes masses de données, tels que Google, Facebook, etc. En effet ce genre de sites possède plusieurs millions voire des milliards d'entrées dans leurs bases de données et tout autant de visites journalières, en conséquence les données sont distribuées sur plusieurs machines, de plus pour des raisons de fiabilité ces bases de données sont dupliquées pour que le service ne soit pas interrompu en cas de panne.

Malheureusement le modèle relationnel présente quelques problèmes liés à ce passage à l'échelle tel que [7] :

IV.1 Problème lié à l'application des propriétés ACID en milieu distribué

Une base de données relationnelle est construite en respectant les propriétés ACID (Atomicité, Cohérence, Isolation, Durabilité), ses propriétés bien que nécessaires à la logique du relationnel nuisent fortement aux performances et en particulier 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 soit respectée 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, car on ne peut valider une transaction que si on est certain qu'elle a été effectuée sur tous les serveurs et le système fait patienter l'utilisateur durant ce temps [7].

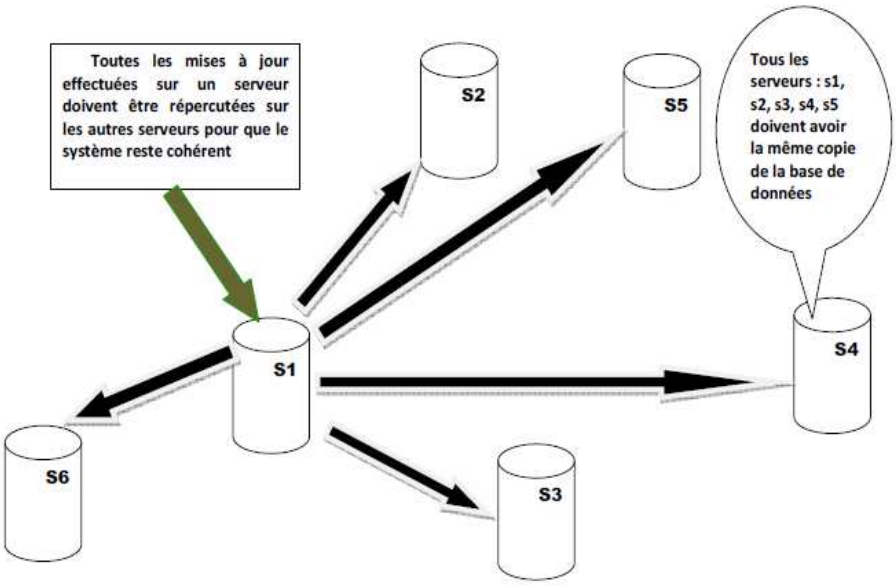


Figure I.1 : Problème lié aux propriétés ACID en milieu distribué [8]

IV.2 Problème de requête non optimale dû à l'utilisation des jointures

Imaginons une table contenant toutes les personnes ayant un compte sur Facebook, soit 1.55 milliards d'utilisateurs actifs par mois, les données dans une base de données relationnelle classique sont stockées par lignes, ainsi si on effectue une requête pour extraire tous les amis d'un utilisateur donné, il faudra effectuer la jointure entre la table des usagers et celle des amitiés (chaque usager ayant au moins un ami) puis parcourir le produit cartésien de ces deux tables. De ce fait, on perd énormément en performances

en raison du temps consommé pour stocker et parcourir une telle quantité de données [8].

IV.3 Problème lié à la gestion des objets hétérogènes

« Le stockage distribué n'est pas la seule contrainte qui pèse à ce jour sur les systèmes relationnels» disait Carl STROZZI. Au fur et à mesure du temps, les structures de données manipulées par les systèmes sont devenues de plus en plus complexes en contrepartie les moteurs de stockage évoluant peu. 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 [8].

- D'autre part le modèle relationnel est fondé sur un modèle mathématique solide s'appuyant sur des concepts simples qui font sa force en même temps que sa faiblesse. Nous expliquerons quelques limites :

IV.4 Surcharge sémantique

Le modèle relationnel s'appuie sur un seul concept (la relation) pour modéliser à la fois les entités et les associations entre ces entités. Il existe donc un décalage entre la réalité et sa représentation abstraite [9].

IV.5 Types de données

Ces modèles sont limités à des types simples (entiers, réels, chaînes de caractères), les seuls types étendus se limitant à l'expression de dates ou de données financières, ainsi que des conteneurs binaires de grande dimension (BLOB, pour Binary Large OBjects) qui permettent de stocker des images ainsi que des fichiers audio ou vidéos. Ces BLOBs ne sont toutefois pas suffisants pour représenter des données complexes (pas de structure), les mécanismes de contrôle BD sont inexistants, et le langage de requêtes (SQL) ne possède pas les opérateurs correspondant aux objets stockés dans ces BLOBs [9].

IV.6 Langage de manipulation

Un autre inconvénient est lié au fait que le langage SQL n'est pas un langage complet. Le développement d'une application autour d'une base relationnelle nécessite l'utilisation d'autres langages (Cobol, Java, C,...) et pose le problème de concordance des types entre le SGBD et le langage de développement [10].

IV.7 Incompatibilités de types ("impedance mismatch")

Elles apparaissent lors de l'utilisation de SQL dans un langage hôte (pour la représentation des nombres, par exemple). Des conversions de types sont nécessaires, et peuvent s'avérer coûteuses en temps et parfois difficiles à mettre en œuvre [9].

IV.8 La pauvreté sémantique

La pauvreté sémantique du modèle relationnel ne permet pas de prendre en compte efficacement les nouveaux besoins liées aux informations multimédia. En particulier, le modèle relationnel est très mal adapté à la gestion de données multivaluées complexes [10].

Et même si on arrive parfois à placer ces données dans des tables, ce n'est pas forcément efficace.

On pourrait imaginer stocker chaque pixel d'une image d'une personne dans une ligne distincte d'une table relationnelle mais il faut alors se poser la question suivante : quel code SQL écrire pour déterminer si l'image représente bien cette personne ? [11].

IV.9 Le partitionnement de données

L'un des problèmes de la normalisation dans un SGBDR concerne la distribution des données et du traitement. S'il y a des données stockées ayant un rapport entre elles, comme des clients, des commandes, des factures, des lignes de facture, etc., dans des tables différentes, des problèmes surgiront en cas de partitionnement de ces données. Pour y remédier, il faut alors s'assurer que les données en rapport les unes avec les autres se trouvent sur le même serveur [12].

V. Conclusion

Les technologies de bases de données relationnelles, qu'on pourrait nommer par "technologies SQL", règnent en maîtres pour le stockage et la manipulation de données depuis plusieurs années. Cette situation de leadership technologique peut facilement être justifiée en raison des différents avantages proposés par ces modèles.

Cependant l'accroissement exponentiel des données, la prise en compte des données faiblement structurées, et les avancées technologiques sont autant d'arguments qui poussent à la migration des SGBD relationnels vers une nouvelle façon de stockage et de manipulation des données.

Dans ce chapitre, nous avons introduit le modèle relationnel et ses propriétés, ensuite on a présenté les limites pratiques et théoriques, liées à l'usage des bases de données relationnelles face aux nouveaux besoins des systèmes d'information, qui peuvent justifier et motiver la migration des SGBD relationnels vers de nouveaux modèles de bases de données dites NoSQL, qui vont être détaillés dans le chapitre suivant.

CHAPITRE II

LE NoSQL

I. Introduction

Nous avons vu dans le chapitre précédent que le modèle relationnel a trouvé ses limites pour la gestion des données massives du Web qui a connu une révolution avec l'avènement des sites web à fort trafic tels que Facebook, Amazon et LinkedIn. Ces grands acteurs du web ont été rapidement limités par les dits systèmes pour deux raisons majeures :

- Les gros volumes de données.
- Les montées 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 manière distincte sont connus sous le nom de base de données NoSQL ou de base de données non relationnelle. Les besoins en performance, lors de traitement de gros volumes de données ainsi que d'augmentation de trafic, ne touchent pas seulement les fameux sites mentionnés ci-dessus, mais aussi de nombreuses entreprises de tous types d'industries confondues, c'est de ce constat qu'est né le mouvement NoSQL[6].

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, la trouvaille 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 [6].

La première partie de ce chapitre consistera à expliquer ce que sont les bases de données de type NoSQL, dans quel but que ce genre de base de données a vu le jour. Nous allons également voir de quelle manière ces type de base de données fonctionnent et sur quelles fondements elles s'appuient, ainsi que les avantages qu'une base de données de type NoSQL peut avoir en comparaison à une base de données relationnel standard.

II. La technologie NoSQL

NoSQL est une combinaison de deux mots : No et SQL. Qui pourrait être mal interprétée car l'on pourrait penser que cela signifie la fin du langage SQL et qu'on ne devrait donc plus l'utiliser. En fait, le « No » est un acronyme qui signifie « Not only », c'est-à-dire en français, « non seulement », c'est donc une manière de dire qu'il y a autre chose que les bases de données relationnelles [6].

NoSQL est un mouvement très récent, qui concerne les bases de données. L'idée du mouvement est simple : proposer des alternatives aux bases de données relationnelles pour coller aux nouvelles tendances et architectures du moment, notamment le Cloud Computing. Les axes principaux du NoSQL sont une haute disponibilité et un partitionnement horizontal des données, au détriment de la cohérence. Alors que les bases de données relationnelles actuelles sont basées sur les propriétés ACID (Atomicité, Consistance ou Cohérence, Isolation et Durabilité).

En effet, NoSQL ne vient pas remplacer les BD relationnelles mais proposer une alternative ou compléter les fonctionnalités des SGBDR pour donner des solutions plus intéressantes dans certains contextes. Le NoSQL regroupe de nombreuses bases de données, récentes pour la plupart, qui se différencient du modèle SQL par une logique de représentation de données non relationnelle. Leurs principaux avantages sont leurs performances et leur capacité à traiter de très grands volumes de données [8].

III. Pourquoi le NoSQL ?

C'est une évidence de dire qu'il convient de choisir la bonne technologie en fonction du besoin. Il existe cependant certains critères déterminants pour basculer sur du NoSQL :

III.1 Taille

Le NoSQL est conçu pour supporter un nombre important d'opérations, de données, d'utilisateurs etc… C'est à cause de cette grande masse de données que le système doit devenir distribué. Bien que tous les systèmes NoSQL ne soient pas conçus pour cette problématique, il est possible de la résoudre sans problème.

III.2 Performance en écriture

Celle-ci est l'intérêt principal du géant Google, Facebook, Twitter, gérant des masses des données qui augmentent considérablement chaque année exigeant un temps très important pour stocker ces données. En conséquence, l'écriture se doit être distribuée sur un cluster de machines, ce qui implique du MapReduce, la réplication, la tolérance aux pannes, et la cohérence [13].

III.3 Type de données flexibles

Les solutions NoSQL supportent de nouveaux types de données et c'est une innovation majeure. Les objets complexes peuvent être mappés sans trop de problèmes avec la bonne solution [13].

III.4 ACID

Bien que ce ne soit pas le but premier du NoSQL, il existe des solutions permettant de conserver certains (voire tous) aspects des propriétés ACID. Se référer au théorème CAP plus bas dans le manuscrit et aux propriétés BASE [13].

III.5 Simplicité de développement

L'accès aux données est simple. Bien que le modèle relationnel soit simple pour les utilisateurs finaux, il n'est pas très intuitif pour les développeurs puisque les données sont restituées selon la structure de la base [13].

III.6 Scalabilité

La scalabilité est le terme utilisé pour définir l'aptitude d'un système à maintenir un même niveau de performance face à l'augmentation de charge ou de volumétrie de données, par augmentation des ressources matérielles [6].

Il y a deux façons de rendre un système extensible : la scalabilité horizontale (externe) ainsi que la scalabilité verticale (interne).

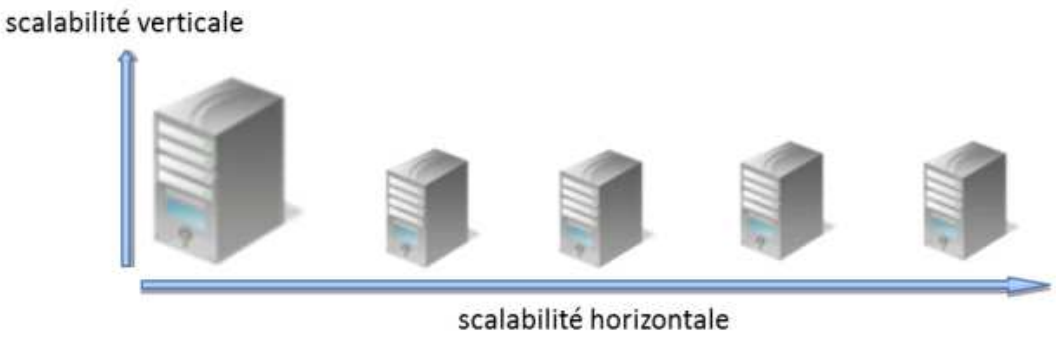


Figure II-1 : Scalabilité horizontale et verticale [6]

III.6.1 Scalabilité horizontale

Le principe de la scalabilité horizontale consiste à simplement rajouter des serveurs en parallèle, c'est la raison pour laquelle on parle de croissance externe. On part d'un serveur basique et on rajoute des nouveaux serveurs identiques afin de répondre à l'augmentation de la charge.

III.6.2 Scalabilité verticale

Le principe de la scalabilité verticale consiste à modifier les ressources d'un seul serveur, comme par exemple le remplacement du CPU par un modèle plus puissant ou par l'augmentation de la mémoire RAM. La croissance interne consiste au fait d'évoluer une machine dans le temps.

Le but des systèmes NoSQL est de renforcer la scalabilité horizontale, les SGBD NoSQL sont basés sur des systèmes innovants permettant la scalabilité horizontale et dont la plupart d'entre eux sont Open Source, ils sont conçus pour répondre à des besoins spécifiques et assurer une extensibilité extrême sur de très grands ensembles de données [6].

IV. Théorème de CAP

Le théorème de CAP est l'acronyme de « Coherence », « Availability » et « Partition tolerance », aussi connu sous le nom de théorème de Brewer. Ce théorème, formulé par Eric Brewer en 2000 et démontré par Seth Gilbert et Nancy Lych en 2002, énonce une conjecture qui définit qu'il est impossible, sur un système informatique de calcul distribué, de garantir en même temps les trois contraintes suivantes [6] :

IV.1 Cohérence (Consistency)

Tous les nœuds (serveurs) du système voient exactement les mêmes données au même moment.

IV.2 Disponibilité (Availability)

Garantie que toute requête reçoive une réponse même si elle n'est pas actualisée.

IV.3 Résistance au partitionnement (Partition tolerance)

Le système doit être en mesure de répondre de manière correcte à toutes requêtes dans toutes circonstances sauf en cas d'une panne générale du réseau. Dans le cas d'un partitionnement en sous-réseaux, chacun de ces sous-réseaux doit pouvoir fonctionner de manière autonome.

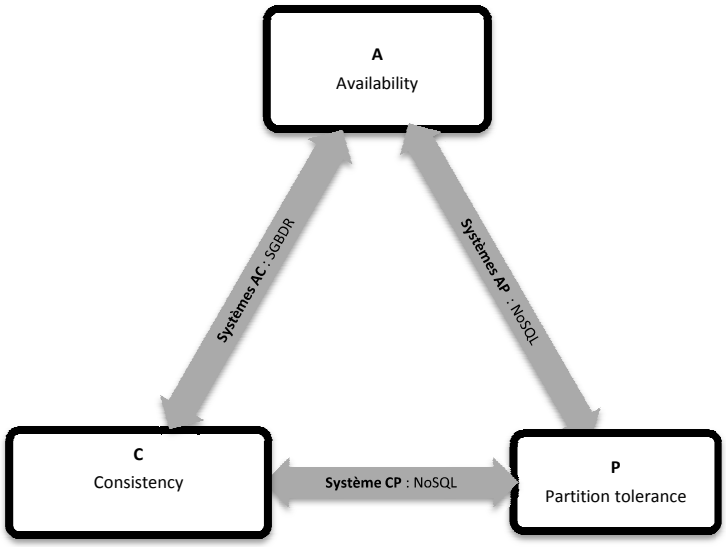


Figure II.2 : Théorème CAP

Dans un système distribué, il est impossible d'obtenir ces 3 propriétés en même temps, il faut en choisir 2 parmi les 3 :

- Les SGBDR assurent les propriétés de Cohérence et de Disponibilité (Availability) => système **AC**

- Les SGBD « NoSQL » sont des systèmes **CP** (Cohérent et Résistant au partitionnement) ou **AP** (Disponible et Résistant au partitionnement) [14].

CP : Les données sont consistantes entre tous les nœuds et le système possède une tolérance aux pannes, mais il peut aussi subir des problèmes de latence ou plus généralement, de disponibilité [13].

AP : Le système répond de façon performante en plus d'être tolérant aux pannes. Cependant rien ne garantit la consistance des données entre les nœuds [13].

CA : Les données sont consistantes entre tous les nœuds (tant que les nœuds sont en ligne). Toutes les lectures/écritures les nœuds concernent les mêmes données. Mais si un problème de réseau apparait, certains nœuds seront désynchronisés au niveau des données (et perdront donc la consistance)[13].

Nous ne pouvons en respecter seulement deux à la fois. Dans la majorité des systèmes se sont les propriétés A et P qui sont respectées.

V. Les propriétés BASE

Dans le premier chapitre consacré aux bases de données relationnelles, nous avons rappelé les propriétés ACID auxquelles doivent répondre les SGBD de type relationnel. Les SGBD 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.

Le principe de BASE est le fruit d'une réflexion menée par Eric Brewer Les caractéristiques de BASE sont fondées sur les limites que montrent les SGBD relationnelles. Voici sa description [6]:

V.1 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 [6].

V.2 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 [6].

V.3 Eventual consistancy (Cohérence à terme)

Cela indique que le système devient cohérent dans le temps, pour que pendant ce laps de temps, le système ne reçoive pas d'autres données [6].

VI. Différents types de base de données NoSQL

Les bases de données NoSQL ne sont plus fondées sur l'architecture classique des bases relationnelles. Quatre grandes catégories se distinguent parmi celles-ci [16] :

VI.1 Bases de données Orientées Clé / Valeur

Les bases de données NoSQL de type clé / valeur s'articulent sur une architecture très basique. On peut les apparenter à une sorte de HashMap, c'est-à-dire qu'une valeur, un nombre ou du texte est stocké grâce à une clé, qui sera le seul moyen d'y accéder. Leurs fonctionnalités sont tout autant basiques, car elles ne contiennent que les commandes élémentaires du CRUD.

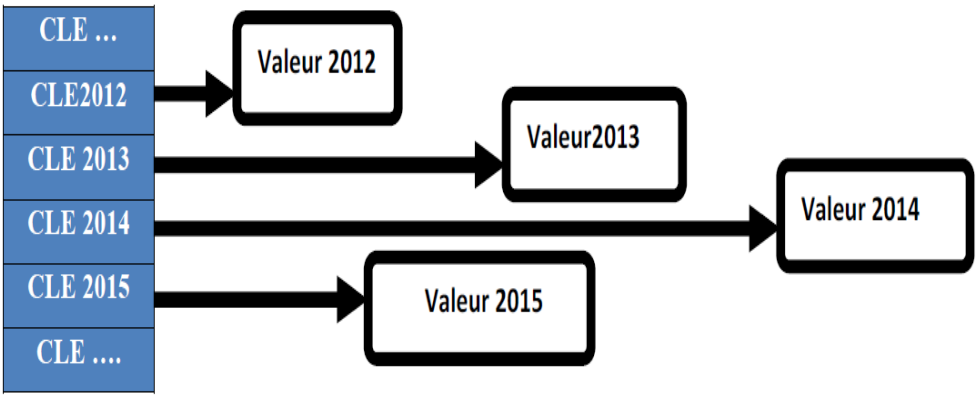


Figure II.3 : Illustration d'une Base de données Orientée Clé / Valeur [8]

Cette approche permet de conserver des performances élevées en lecture/écriture, car de simples accès disques sont effectués pour accéder aux données. Cela permet aux données totalement hétérogènes entre elles d'être stockées. Comme les valeurs stockées sont de type simple (un entier, du texte, un tableau), c'est au développeur de gérer la façon dont elles sont stockées afin de pouvoir les récupérer.

Les base de type clé/valeur les plus utilisées sont Redis et Riak [17].

VI.2 Bases de données Orientées Document

Ces bases de données sont une évolution des bases de données de type clé-valeur. Ici les clés ne sont plus associées à des valeurs sous forme de bloc binaire mais à un document dont le format n'est pas imposé. Il peut être de plusieurs types différents comme par exemple du JSON ou du XML, pour autant que la base de données soit en mesure de manipuler le format choisit afin de permettre des traitements sur les documents. Dans le cas contraire, cela équivaudrait à une base de données clé-valeur. Bien que les documents soient structurés, ce type de base est appelée « schemaless ». Il n'est donc pas nécessaire de définir les champs d'un document [16].

On retrouve principalement MongoDB et CouchBase comme solutions basées sur le concept de base documentaire.

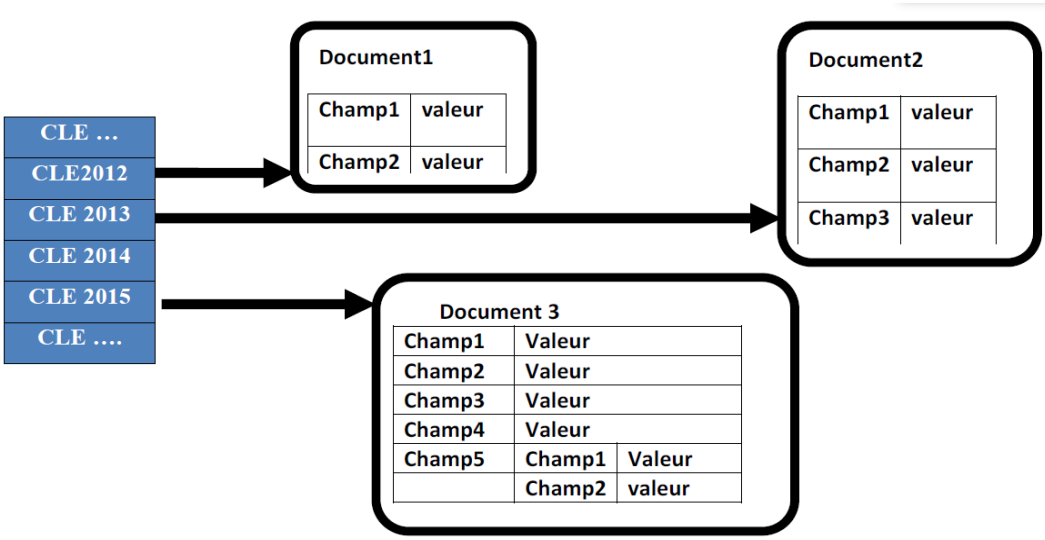


Figure II.4 : Illustration d'une Base de données Orientée Document [8]

VI.3 Bases de données Orientées Colonne

Les bases de données orientées colonne ont été conçues par les géants du web afin de faire face à la gestion et au traitement de gros volumes de données. Elles intègrent souvent un système de requêtes minimalistes proche du SQL.

Bien qu'elles soient abondamment utilisées, il n'existe pas encore de méthode officielle ni de règles définies pour qu'une base de données orientée colonne soit qualifiée de qualité ou non.

Le principe d'une base de données colonne consiste dans leur stockage par colonne et non par ligne. C'est-à-dire que dans une base de données orientée ligne (SGBD classique) on stocke les données de façon à favoriser les lignes en regroupant toutes les colonnes d'une même ligne ensemble. Les bases de données orientées colonne quant à elles vont stocker les données de façon à ce que toute les données d'une même colonne

soient stockées ensemble. Ces bases peuvent évoluer avec le temps, que ce soit en nombre de lignes ou en nombre de colonnes. Autrement dit, et contrairement à une base de données relationnelle où les colonnes sont statiques et présentes pour chaque ligne, celles des bases de données orientées colonne sont dite dynamiques et présentes donc uniquement en cas de nécessité. De plus le stockage d'un « null » est 0[16].

Les bases les plus connues se basant sur ce concept sont HBase et Cassandra.

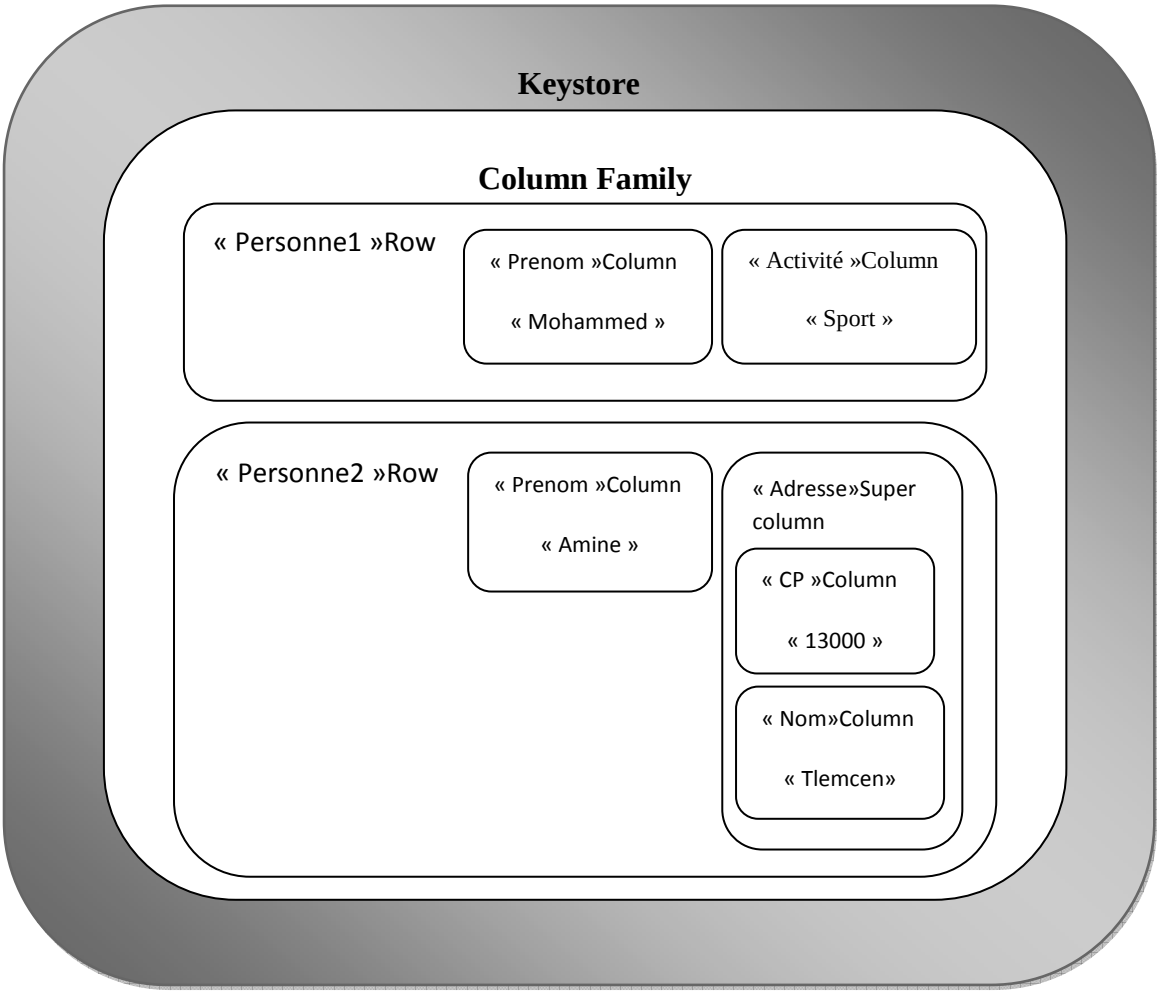


Figure II.5 : Illustration d'une Base de données Orientée Colonne

VI.4 Base de données Orientées Graphe

Bien que les bases de données de type clé-valeur, colonne, ou document tirent leur principal avantage de la performance du traitement de données, les bases de données orientées graphe permettent de résoudre des problèmes très complexes qu'une base de données relationnelle serait incapable de faire. Les réseaux sociaux (Facebook, Twitter,etc), où des millions d'utilisateurs sont reliés de différentes manières, constituent un bon exemple : amis, fans, famille etc. Le défi ici n'est pas le nombre d'élément à gérer, mais le nombre de relations qu'il peut y avoir entre tous ces

éléments. En effet, il y a potentiellement n^2 relations à stocker pour n éléments. L'approche par graphes devient donc inévitable pour les réseaux sociaux tels que Facebook ou Twitter [16].

Comme son nom l'indique, ces bases de données reposent sur la théorie des graphes, avec trois éléments à retenir [16] :

-Un objet (dans le contexte de Facebook nous allons dire que c'est un utilisateur) sera appelé un nœud.

-Deux objets peuvent être reliés entre eux (comme une relation d'amitié).

-Chaque objet peut avoir un certain nombre d'attributs (statut social, prénom, nom etc.)

La principale solution est Neo4J.

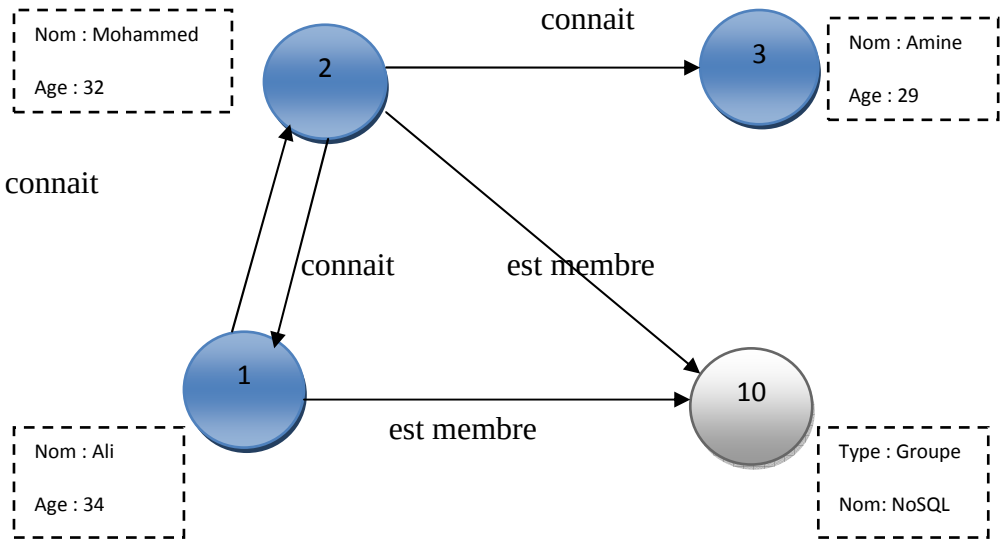


Figure II-6 : Illustration d'une Base de données Orientée Graphe

VII.Le requêtageNoSQL

Dans le monde du NoSQL il n'y a pas de langage standard comme SQL l'est dans le monde des bases de données relationnelles. L'interrogation des bases de données NoSQL se fait au niveau applicatif à travers principalement la technique dite de «MapReduce »[8].

MapReduceest une technique de programmation distribuée très utilisée dans le milieu NoSQL et qui vise à produire des requêtes distribuées. Cette technique se décompose en deux grandes étapes :

VII.1 Etape de mapping

Chaque item d'une liste d'objets clé-valeur passe par la fonction map qui va retourner un nouvel élément clé-valeur. Exemple de la fonction map : à chaque couple (UserId,

User), on assigne le couple (Role, User). A l'issue de l'étape de mapping, on obtient une liste contenant les utilisateurs groupés par rôle [8].

VII.2 Etape de Reduce

La fonction reduce est appelée sur le résultat de l'étape de mapping et permet d'appliquer une opération sur la liste. Exemples de fonction reduce :

- Moyenne des valeurs contenues dans la liste
- Comptabilisation des différentes clés de la liste
- Comptabilisation du nombre d'entrées par clé dans la liste

L'étape de mapping peut être parallélisée en traitant l'application sur différents nœuds du système pour chaque couple clé-valeur. L'étape de réduction n'est pas parallélisée et ne peut être exécutée avant la fin de l'étape de mapping. Les bases de données NoSQL proposent diverses implémentations de la technique MapReduce permettant le plus souvent de développer les méthodes map et reduce en Java Scriptou en Java [8].

VIII. Les avantages du NoSQL

VIII.1 Plus évolutif

NoSQL est plus évolutif. C'est en effet l'élasticité de ses bases de données NoSQL qui le rend si bien adapté au traitement de gros volumes de données. Au contraire, les bases de données relationnelles ont souvent tendance à utiliser la scalabilté verticale, quand celui-ci atteint ses limites [16].

VIII.2 Plus flexible

N'étant pas enfermée dans un seul et unique modèle de données, une base de données NoSQL est beaucoup moins restreinte qu'une base SQL. Les applications NoSQL peuvent donc stocker des données sous n'importe quel format ou structure, et changer de format en production. En fin de compte, cela équivaut à un gain de temps considérable et à une meilleure fiabilité. Par contre une base de données relationnelle doit être gérée attentivement, car un changement, aussi mineur, peut entraîner un ralentissement ou un arrêt du service [16].

VIII.3 Plus économique

Les serveurs destinés aux bases de données NoSQL sont généralement bon marché contrairement à ceux qui sont utilisés par les bases relationnelles. De plus, la très grande majorité des solutions NoSQL sont Open Source, ce qui reflète une économie importante sur le prix des licences [16].

VIII.4 Plus simple

Les bases de données NoSQL ne sont pas forcément moins complexes que les bases relationnelles, mais elles sont beaucoup plus simples à déployer. La façon dont elles ont été conçues permet une gestion beaucoup plus légère [16].

IX. Conclusion

Nous avons vu dans ce chapitre que NoSQL est une technologie principalement développée dans un contexte où la volumétrie des données rendait indispensable l'utilisation des systèmes distribués pour assurer leur stockage et leur traitement. C'est dans ce contexte qu'on a essayé de justifier le changement radical qu'apporte cette technologie aux architectures traditionnelles des bases de données que nous avons l'habitude de manipuler.

Dans ce qui suit, nous allons accentuer sur les solutions NoSQL qui feront l'objet de notre étude comparative dans l'optique de comparer leurs performances.

CHAPITRE III

Les solutions NoSQL étudiées

I. Introduction

On dénombre actuellement dans le marché informatique plus de 305 solutions NoSQL, qu'elles soient de type:

- Base clé/valeur comme : Redis, Riak, Voldemort.
- Base documentaire comme : MongoDB, CouchBase, Terrastore.
- Base orientée colonne comme : Cassandra, HBase Amazon, SimpleDB.
- Base orientée graphe comme : Neo4j.
- Base multi modèle : comme OrientDB.

Dans ce chapitre nous allons nous atteler à la description proprement dite des bases NoSQL en ciblant des solutions très populaires. Une sélection variée de différentes architectures de bases de données NoSQL qui ont obtenu les meilleurs classements selon Solid IT [15] est présentée dans la Figure III.1.

Les six solutions qui feront l'objet de notre étude sont donc : MongoDB, Couchbase, Cassandra, Hbase, Redis, OrientDB.

305 systems in ranking, May 2016

Rank May 2016	Rank Apr 2016	Rank May 2015	DBMS	Database Model	Score May 2016	Score Apr 2016	Score May 2015
1.	1.	1.	Oracle	Relational DBMS	1462.02	-5.51	+19.93
2.	2.	2.	MySQL	Relational DBMS	1371.83	+1.72	+77.56
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1142.82	+7.77	+11.79
4.	4.	4.	MongoDB	Document store	320.22	+7.78	+42.90
5.	5.	5.	PostgreSQL	Relational DBMS	307.61	+3.89	+34.09
6.	6.	6.	DB2	Relational DBMS	185.96	+1.87	-15.09
7.	8.	8.	Cassandra	Wide column store	134.50	+4.83	+27.95
8.	7.	7.	Microsoft Access	Relational DBMS	131.58	-0.39	-14.00
9.	9.	10.	Redis	Key-value store	108.24	-3.00	+13.51
10.	10.	9.	SQLite	Relational DBMS	107.26	-0.70	+2.10
11.	11.	14.	Elasticsearch	Search engine	86.31	+3.73	+21.48
12.	13.	13.	Teradata	Relational DBMS	73.74	+1.48	+3.62
13.	12.	11.	SAP Adaptive Server	Relational DBMS	71.48	-1.94	-14.01
14.	14.	12.	Solr	Search engine	65.62	-0.42	-17.31
15.	15.	15.	HBase	Wide column store	51.84	+0.35	-9.87
16.	16.	17.	Hive	Relational DBMS	47.51	-1.57	+2.94
17.	17.	16.	FileMaker	Relational DBMS	46.71	+0.60	-6.19
18.	18.	18.	Splunk	Search engine	44.31	+1.96	+3.59
19.	19.	21.	SAP HANA	Relational DBMS	41.37	+1.02	+8.59
20.	21.	25.	MariaDB	Relational DBMS	33.97	+2.38	+10.37
21.	20.	22.	Neo4j	Graph DBMS	32.61	+0.70	+3.97
22.	22.	19.	Informix	Relational DBMS	30.58	-0.94	-6.16
23.	23.	20.	Memcached	Key-value store	27.90	-0.11	-5.21
24.	24.	24.	Couchbase	Document store	24.29	-0.73	-0.97
25.	25.	30.	Amazon DynamoDB	Multi-model	23.60	+0.48	+8.73
26.	26.	23.	CouchDB	Document store	21.99	-0.38	-5.18
27.	27.	26.	Firebird	Relational DBMS	19.90	-0.20	-1.93
28.	28.	28.	Microsoft Azure SQL Database	Relational DBMS	19.68	+0.44	+3.95
29.	29.	29.	Vertica	Relational DBMS	19.28	+0.14	+4.28
30.	30.	27.	Netezza	Relational DBMS	19.26	+0.41	+1.13
31.	31.	31.	Riak KV	Key-value store	11.53	+0.04	-1.36
32.	32.	33.	Ingres	Relational DBMS	10.00	-0.32	-0.24
33.	33.	36.	Greenplum	Relational DBMS	9.35	-0.42	+0.77
34.	35.	32.	dBASE	Relational DBMS	9.22	+0.19	-1.21
35.	36.	42.	Amazon Redshift	Relational DBMS	9.19	+0.25	+3.92
36.	34.	34.	MarkLogic	Multi-model	9.07	-0.04	-0.83
37.	38.	46.	Impala	Relational DBMS	8.41	+0.12	+3.80
38.	37.	35.	Sphinx	Search engine	8.34	-0.14	-1.45
39.	39.	40.	Hazelcast	Key-value store	6.89	+0.21	+1.07
40.	40.	39.	Ehcache	Key-value store	6.83	+0.37	-1.09
41.	41.	51.	OrientDB	Multi-model	6.13	-0.18	+2.14

Figure III.1 : Classement des bases de données en fonction de leurs popularités [15]

II. Panorama des solutions NoSQL étudiées

II.1 MongoDB

II.1.1 Description

Développé depuis 2007 par la société de logiciel 10gen. MongoDB est un système de gestion de base de données orientée document, écrit en C++ et distribuée sous licence AGPL (licence libre) et très adaptée aux applications web.

MongoDB a été adoptée par plusieurs grands noms de l'informatique, tels que Foursquare, SAP, ou bien même GitHub.

Elle manipule des documents au format BSON (Binary JSON), qui est un dérivé de JSON plus axé sur la performance, fonctionnant comme une architecture distribuée centralisée. MongoDB réplique les données sur plusieurs serveurs avec le principe de maître-esclave, permettant ainsi une plus grande tolérance aux pannes. La répartition et la duplication des documents sont faites de telle sorte que les documents les plus demandés soient sur le même serveur et que celui-ci soit dupliqué en un nombre de fois suffisant [16].

II.1.2 Architecture

MongoDB possède différents modes de fonctionnement :

- Single
- Réplication
  - Master / Slave
  - Replica Set
- Sharding

- Le mode Single, sert à faire fonctionner une base de données sur un seul serveur.

- La réplication est découpée en deux modes, alors que le Sharding se montre plus comme un méta-mode car il utilise l'une des deux méthodes de réplication comme couche sous-jacente [10].

II.1.2.1 Maître/ Esclave

Le mode Master / Slave bien connu dans l'informatique, n'apporte rien de plus au sein de MongoDB qui s'articule sur un serveur principal en tant que maître s'occupant des demandes des clients et chargé de répliquer les données sur le serveur esclave de façon asynchrone. L'esclave est présent pour prendre la place du maître si ce dernier tombe en panne.

Le premier avantage de cette structure c'est la garantie d'une forte cohérence des données, car seul le maître s'occupe des clients. Aucune opération n'est faite sur l'esclave, hormis quand le maître envoie les mise à jours [17].

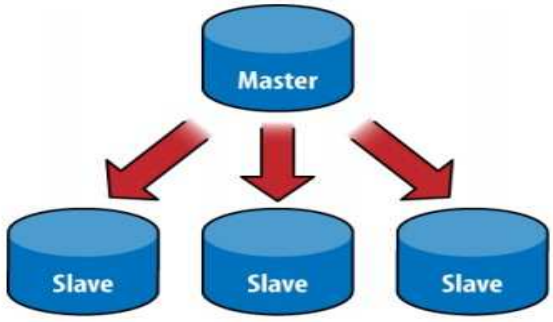


Figure III.2 : MongoDB Maître / Esclave [28]

II.1.2.2 Replica Sets

Le Replica Sets fonctionne avec plusieurs nœuds possédant chacun la totalité des données, de la même manière qu'un réplica. Ces différents nœuds vont alors élire un nœud primaire, qui peut s'apparenter à un maître. Il faut qu'un nœud obtienne la majorité absolue afin d'être élu. Dans le cas où un nœud n'obtiendrait pas la majorité, le processus de vote recommencerait à zéro. De plus, une priorité peut être donnée à chaque nœud afin de lui donner plus de poids lors de l'élection. Un serveur arbitre peut être inséré dans le système qui ne contiendra aucune donnée mais participera aux élections afin de pouvoir garantir la majorité absolue [17].

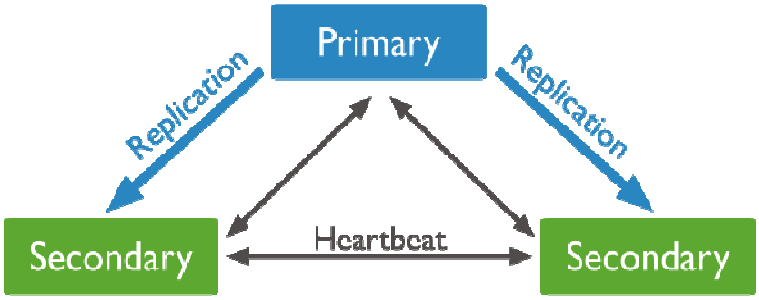


Figure III.3: MongoDB Replica Sets [29]

II.1.2.3 Sharding

Le Sharding est une sur-couche qui est basée sur du Maître / esclave ou du Replica Sets.

- Les Shards : Ce sont un groupe de serveurs en mode Maître /esclave 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. De plus, ils organisent la structure des données entre les Shards.

Le Sharding sert à partager les données entre plusieurs Shard, chaque Shard doit stocker une partie des données. La définition de la manière dont les données seront découpées se fait grâce à une Shared Key qui permet de définir sur quel critère seront partagées les données entre les Shards [17].

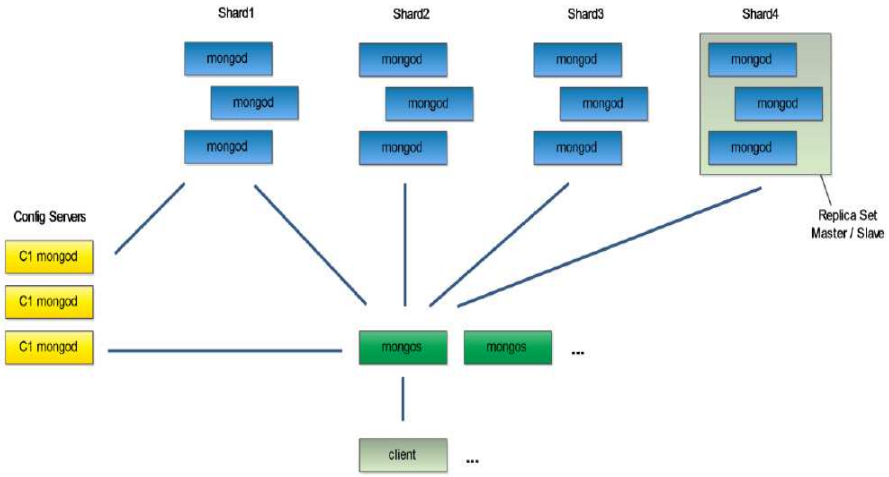


Figure III.4: MongoDB Sharding [17]

II.2 CouchBase

II.2.1 Description

CouchBase Server appartient à la classe des bases de données NoSQL orientées document conçu pour les applications web interactives. Elle dispose d'un modèle de données flexible, facilement extensible offrant une haute performance constante, capable de servir des données d'application avec 100% de disponibilité. C'est une base de données persistante qui exploite une couche de mise en cache RAM intégrée [18].

II.2.2 Architecture

CouchBase Server comprend un seul paquet qui est installé sur tous les nœuds. En utilisant les SDKs (également connu sous le nom des bibliothèques de connectivité client à puce), on peut écrire des applications dans différents langages (Java, node.js, .NET ou autres). Les applications se connectent à un cluster CouchBase Server pour effectuer des opérations de lecture / écriture et exécuter des requêtes avec de faibles latences et haut débit [19].

La caractéristique principale de CouchBase est le fait qu'elle est « schema-less » ou sans schéma. Il n'y a aucun schéma fixe pour stocker des données. En outre, il n'y a aucune

jointure faite sur les données. Il permet le stockage distribué utilisant des ressources informatiques, tels que le CPU et la RAM, s'étendant à travers les nœuds qui font partie du cluster CouchBase.

Les clusters de CouchBase se composent de plusieurs nœuds ou un cluster est une collection d'une ou plusieurs instances du serveur CouchBase qui sont configurés comme un cluster logique [20].

Contrairement à la plupart des technologies des clusters travaillant sur les relations maître/esclave, CouchBase s'appuie sur le mécanisme de nœud peer-to-peer. Autrement dit, qu'il n'y a pas de différence entre les nœuds du cluster ainsi la fonctionnalité fournie par chaque nœud est la même. Il n'ya donc aucun point de défaillance unique : lorsqu'il y a une défaillance d'un nœud, un autre nœud prend ses responsabilités, fournissant ainsi une disponibilité élevée.

II.2.2.1 Le Data Manager (Gestionnaire de données)

Toute opération effectuée sur le système de base de données CouchBase est stockée dans la mémoire, qui agit comme une couche de mise en cache. Par défaut, chaque document est stocké dans la mémoire pour chaque lecture, insertion, mis à jour, et ainsi de suite jusqu'à ce que la mémoire soit pleine. Toutefois, afin d'assurer la persistance de l'enregistrement, il y a une file d'attente du disque. Ceci purgera l'enregistrement sur le disque de manière asynchrone, sans impact sur la demande du client. Cette fonctionnalité est fournie automatiquement par le gestionnaire de données, sans aucune intervention humaine [20].

II.2.2.2 Cluster Management (Gestion du cluster)

Le gestionnaire de cluster est le responsable de l'administration de nœud ou chaque nœud d'un cluster CouchBase inclut le composant de cluster manager et data manager. Il gère la récupération et le stockage de données. CouchBase clients utilisent la carte cluster fourni par le gestionnaire de cluster pour savoir quel nœud contient les données requises et communique ensuite avec le gestionnaire de données pour effectuer des opérations de base de données sur ce nœud [20].

II.2.2.3 Buckets (Seaux)

Pour stocker un document dans un cluster CouchBase, il faut d'abord créer un seau comme un espace de noms logique. Précisément, un seau est un conteneur virtuel indépendant, qui regroupe les documents logiquement dans un cluster CouchBase, ce qui équivaut à un espace de noms de base de données SGBDR. Il peut être consulté par

divers clients dans une application. C'est possible également de configurer des fonctionnalités telles que la sécurité, la réplication et ainsi de suite par seau.

En interne, CouchBase s'organise pour stocker des documents dans différents entrepôts pour différents seaux. Des informations telles que les statistiques d'exécution sont recueillies et rapportées par le cluster CouchBase, regroupés par type de seau [20].

CouchBase fournit deux types de seaux, qui se différencient par le mécanisme de son stockage. Les deux types sont les suivants :

- Memcached : Comme son nom l'indique, des seaux de type Memcached stocke des documents uniquement dans la RAM. Cela signifie que les documents stockés dans le seau Memcache sont volatiles et en conséquence ces types ne survivront pas à un redémarrage du système.
- CouchBase : Le type de seaux CouchBase donne la persistance aux documents. Il est distribué à travers un cluster de nœuds et peut configurer la réplication, qui n'est pas pris en charge dans le type de seaux Memcached. Il est très disponible, puisque les documents sont répliqués entre les nœuds d'un cluster.

II.2.2.4 Views (Les vues)

Une vue est un indice persistant de documents dans une base de données, elle permet l'indexation et l'interrogation des données ainsi que l'extraction des champs de documents JSON sans l'ID de document. Les vues sont écrites sous la forme de MapReduce, dont nous avons déjà parlé. CouchBase implémente MapReduce en utilisant le langage JavaScript [20].

II.3 Cassandra

II.3.1 Description

La base de données Cassandra est initialement développé par Facebook afin de répondre à des besoins concernant son service de messagerie, puis elle a été libérée en Open Source et a été adoptée par plusieurs autres grands du Web tel que Digg.com ou Twitter. Elle est aujourd'hui l'un des principaux projets de la fondation Apache, une organisation à but non lucratif développant des logiciels Open Source. Cassandra est un système de gestion de base de données NoSQL orientée colonne et écrit en java qui permet la gestion massive de données réparties sur plusieurs serveurs, assurant ainsi une haute disponibilité des données. Voici une liste des principales caractéristiques de Cassandra [16] :

Haute tolérance aux pannes : les données d'un nœud sont automatiquement répliquées sur différents nœuds. De cette manière, les données qu'il contient sont également

disponibles sur d'autres nœuds si l'un des nœuds venait à être hors service. Conçu sur le principe qu'une panne n'est pas une exception mais une normalité, il est simple de remplacer un nœud qui tombe sans rendre le service indisponible [16].

Modèle de données riche : Cassandra propose un modèle de données basé sur BigTable (Google) de type clé-valeur. Elle permet de développer de nombreux cas d'utilisation dans l'univers Web [16].

Elastique : Que ce soit en écriture ou en lecture, les performances augmentent de façon linéaire lorsqu'un serveur est ajouté au cluster. Cassandra assure également la disponibilité du système et la non interruption au niveau des applications [16].

II.3.2 Architecture

L'architecture de Cassandra a été conçue pour être complètement distribuée et ne pas présenter de point de défaillance unique (**S**ingle **P**oint **O**f **F**ailure ou **SPOF**). Pour obtenir ce résultat, tous les nœuds de Cassandra sont équivalents. L'ajout d'un nouveau nœud est très simple et il suffit pour cela de connaître un seul nœud du cluster. Au final, les nœuds sont organisés sous la forme d'un anneau, en cas de défaillance de l'un des nœuds, il est simplement retiré de l'anneau.

Les données sont partitionnées et répliquées entre les différents nœuds. Chaque famille de colonnes est en fait stockée comme une table de hachage partitionnée de manière homogène sur le cluster. La stratégie de génération de clés et de partitionnement peut être modifiée par le concepteur de la base pour chaque famille de colonnes.

En outre, Cassandra est capable de gérer deux concepts différents afin de faciliter son exploitation en Datacenters :

- Les racks : Il s'agit de groupe de serveurs liés entre eux. Le groupe de serveurs d'un même rack contient l'ensemble des données et est autonome. On place souvent deux racks dans le même Datacenter ou les données sont automatiquement dupliquées entre les deux racks [21].
- Le Datacenter : A chaque rack, on associe un jumeau dans un autre Datacenter. Cela permet d'assurer la réplication entre les deux sites distants [21].

II.4 Hadoop

II.4.1 Description

Hadoop est un framework Open Source Apache écrit en java qui permet le traitement de grands ensembles de données réparties entre des clusters à l'aide de modèles de programmation simples. L'application framework Hadoop fonctionne dans un environnement qui fournit le stockage et le calcul distribué entre les clusters [22].

Hadoop a deux couches principales à savoir:

- Couche de stockage HDFS.
- Couche de traitement / calcul (MapReduce),

II.4.1.1 HDFS

HDFS est un système de fichiers distribué, inspiré du système GFS développé par Google. Il se démarque des autres systèmes de fichiers distribués par sa grande tolérance aux fautes et le fait qu'il soit conçu pour être déployé sur des machines à faible coût. HDFS fournit un haut débit d'accès aux données et est adapté pour les applications qui nécessitent de grands groupes de données [23].

Dans Hadoop, les différents types de données**,** qu'elles soient structurées ou non, sont stockées à l'aide de HDFS qui 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, deux sur le même support et un sur un support différent. Les différents nœuds de données peuvent communiquer entre eux pour rééquilibrer les données [16].

II.4.1.2 MapReduce

Le second composant majeur d'Hadoop est MapReduce, qui gère la répartition et l'exécution des requêtes sur les données stockées dans le cluster. Le framework MapReduce est conçu pour traiter des problèmes parallélisables à très grande échelle en s'appuyant sur un très grand nombre de nœuds [24].

II.4.2 HBase

II.4.2.1 Description

HBase est un système de gestion de base de données NoSQL distribué, écrit en Java, disposant d'un stockage structuré pour les grandes tables. Il est sous Licence Apache.

La société américaine Cloudera distribue une édition de Hadoop et HBase avec support nommée Cloudera Enterprise.

HBase est inspiré des publications de Google sur BigTable utilisé par des grands acteurs comme Facebook pour stocker tous les messages du réseau social. Il fait partie de la catégorie orientée colonnes basées sur une architecture maître/esclave, et capable de gérer d'énormes quantités d'informations. Il s'installe généralement sur le système de fichiers HDFS d'Hadoop pour faciliter la distribution [25].

Parmi les points forts de Hbase, c'est sa forte utilisation dans les contextes de Big Data, sa solidité de conception, l'excellente tolérance au partitionnement et la garantie des propriétés ACID [25].

II.4.2.2 Architecture

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, et les nœuds esclaves appelés «Region Server», gérant des sous-ensembles de tables, appelés « Region » [26].

HBase est utilisé sur un ensemble de machines gérées par HDFS. Les données des tables sont stockées sur les nœuds «DataNode» HDFS par les nœuds de type « Region Server » HBase.

La Figure III.5 présente le fonctionnement général d'HBase. Les tables sont stockées en orientation par colonne. Chaque table est constituée d'un ensemble de familles de colonnes (regroupements logiques de colonnes). Sur chaque famille de colonnes il est possible d'ajouter des colonnes. Les lignes des tables sont partitionnées en plusieurs régions. Lors de la création d'une table, une seule région est créée, elle est ensuite automatiquement divisée en sous-parties lorsque sa taille atteint un seuil limite. Le rôle du master est la gestion des régions.

Zookeeper est un outil permettant de coordonner des services distribués et permettra de faire le lien entre le maitre et le HDFS [25].

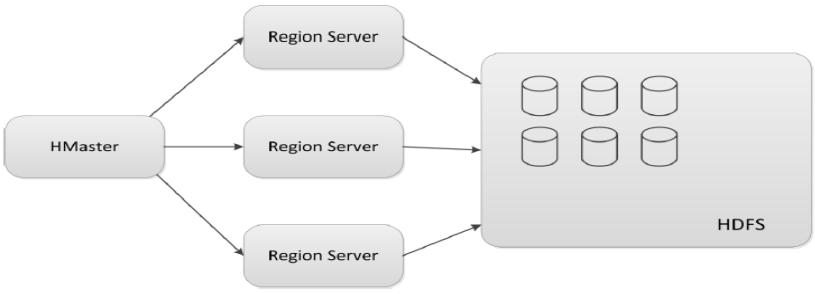


Figure III.5 : Schéma de HBase [25]

II.5 Redis

II.5.1 Description

Redis qui vient de l'anglais REmoteDIctionary Server est une base NoSQL de type clé / valeur. Elle fut développée par Salvatore Sanfilippo et Pieter Noordhuis, tout en étant

dès le départ sous licence BSD. Redis est actuellement utilisée par des entreprises comme The Guardian, GitHub ou encore Blizzard Entertainment [17].

II.5.2 Architecture

Redis est une base de données de type clé/valeur voulant offrir de grandes performances en fonctionnant intégralement en mémoire vive. Bien entendu une première difficulté se pose, c'est le risque de perte de données en cas de coupure électrique. Afin d'y remédier Redis propose deux fonctions [17] :

- RDB : va écrire un fichier sur le disque contenant toute la base de données en mémoire. Le transport de cette base en est ainsi facilité. Cependant, cela n'est pas adapté pour enregistrer une transaction. Il faudra faire un enregistrement de la base pour sauver la transaction désirée. On peut ainsi comprendre que ce n'est pas préconisé pour faire autre chose que des backups complets périodiques [26].
- AOF : C'est un journal dans lequel les transactions sont sauvées dans un temps répété et déterminé [26].

II.5.2.1 Maître / Esclave

Redis se fonde sur une architecture maître / esclave comme beaucoup de base NoSQL et permet aussi d'évoluer dans un environnement de type Sharding, où les données sont découpées et partagées entre plusieurs serveurs.

Dans un environnement maître / esclave, seul le maître s'occupera des requêtes en écriture des clients. Les esclaves quant à eux peuvent s'occuper de répondre aux clients afin d'alléger le serveur maître des requêtes des clients.

Redis est caractérisé par un inconvénient majeur : si une panne venait à survenir sur le serveur maître, aucun esclave ne viendrait le remplacer automatiquement donc il n'existe pas de mécanisme automatique de récupération. Pour y remédier à ce type de panne, l'administrateur procède manuellement pour désigner un nouveau maître parmi les esclaves qui restent toujours disponibles en lecture, afin de pouvoir continuer à répondre aux clients [17].

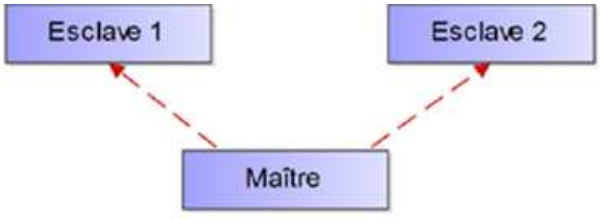


Figure III.6 : Redis Maître / Esclave [17]

II.5.2.2 Sentinel

Redis propose un service à part pour palier au problème de récupération en cas de panne. Pour cela, en plus des divers serveurs Redis s'occupant de la base de données, des serveurs Redis Sentinel seront utilisés et serviront à gérer le cluster des serveurs.

Bien qu'encore en phase béta au moment de la rédaction de ce mémoire, Redis Sentinel devrait bientôt être disponible en version stable. Redis Sentinel propose donc trois types de services :

- Monitoring : Connaître l'état des différentes instances de Redis.
- Notification: Sentinel peut notifier à l'administrateur ou à un ordinateur le dysfonctionnement d'une instance.
- Automatic Failover : Dans le cas où une panne du serveur maître venait à arriver, Sentinel s'occuperait de le remplacer en promouvant un serveur esclave [17].

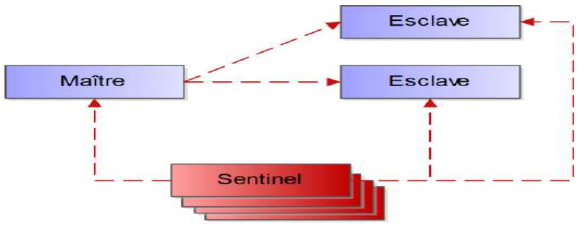


Figure III.7: Redis Sentinel [17]

II.5.2.3 Cluster

Pour en revenir à la configuration d'un cluster avec Redis, la répartition des données se fait aisément grâce à la technique du Consistent Hashing. Cette technique permet de répartir aisément les données entre les différents nœuds et facilite le rajout de nouveaux serveurs. En outre, chaque nœud s'occupant d'une partie des données, définie par le hash, peut avoir autant d'esclaves que nécessaire. En cas de dysfonctionnement du maître, les esclaves éliront un nouveau maître pour prendre sa place. Durant cet intermède d'indisponibilité, les autres nœuds du cluster possédant les autres données ne serviront pas les clients afin de garantir l'intégrité des données [17].

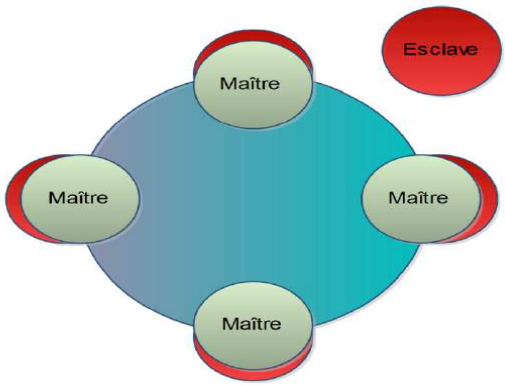


Figure III.8 : Redis Cluster [17]

II.5.2.4 Réplication

Chaque fois qu'une transaction est effectuée sur le maître, ce dernier la répercute directement sur les différents esclaves qui lui sont liés, ce qui permet de les maintenir à jour. Ces derniers vont ensuite la répercuter sur les esclaves liés à eux s'il y en a [17].

II.6 OrientDB

II.6.1 Description

OrientDB est une base de données NoSQL Open Source multi modèle qui peut être utilisée comme une base de données orientée document, orientée graphe ou clé/valeur.

Ce modèle est entièrement écrit en Java, qui peut être déployée sur toutes les plateformes, entièrement transactionnel prenant en charge les transactions ACID garantissant le traitement fiable de toutes les transactions de la base de données ainsi tous les documents en attente, en cas d'accident, sont récupérés et engagés.

Il est possible de l'exécuter en mode distribué, en mode intégré ou en mémoire (utile pour le développement, les essais ou petites applications autonomes) [27].

II.6.2 Multi modèle

II.6.2.1 Le modèle de document

Le modèle de document OrientDB ajoute également le concept d'un " LINK " comme une relation entre les documents. Lors de la récupération d'un document, tous les liens sont automatiquement générés par OrientDB. Ceci est une différence majeure par rapport à d'autres bases de données orientées document, comme MongoDB ou CouchDB, où le développeur doit gérer toutes les relations entre les documents [27].

II.6.2.2 Le modèle graphique

Cette architecture représente une structure en forme de réseau constituée par les sommets reliés entre eux par des bords (également connu sous le nom d'arcs). Le

modèle de graphe d'OrientDB est représenté par le concept d'un graphe de propriétés, définissant les éléments suivants [27]:

- **Vertex** : une entité qui peut être liée à d'autres sommets.
- **Bord** : une entité qui relie deux sommets.

II.6.2.3 Le modèle clé / valeur

Ceci est le modèle le plus simple des trois. Les données sont accessibles par une clé, où les valeurs peuvent être des types simples et complexes. OrientDB soutient les documents et les éléments graphe comme des valeurs permettant un modèle plus riche, qu'on peut trouver normalement dans le modèle clé / valeur classique [27].

III. Conclusion

Ce chapitre a fait l'objet d'une présentation des solutions NoSQL étudiées et comparées dans le chapitre suivant « étude comparative ». L'objectif était d'établir une brève description de chacune de ces solutions ainsi que leurs architectures et leurs modes de fonctionnement.

Tous les déploiements appliqués sur les différentes solutions NoSQL présentées précédemment, le benchmark utilisé ainsi que tous les résultats expérimentaux seront présentés dans le chapitre suivant.

CHAPITRE VI

L'étude comparative

I. Introduction

Ce chapitre est consacré aux processus de déploiement des différents outils à savoir le benchmark YCSB et les solutions NoSQL: MongoDB, CouchBase, Cassandra, HBase, Redis, et OrientDB qui feront l'objet de notre étude comparative, suivi par une analyse des résultats obtenus des différentes expérimentations, afin d'évaluer les performances des différents modèles de bases de données par rapport à la nature des opérations effectuées sur ces bases.

En revanche, cette étude a été réalisée dans un environnement monoposte où tous les tests ont été effectués dans un PC portable, HP 630 avec un processeur : Intel Core i3 CPUM380@2.53 GHz, avec 2 Go de RAM fonctionnant sur un système d'exploitation Ubuntu 14.10.

II. Présentation de l'outil de comparaison

Pour l'analyse expérimentale, nous avons utilisé le YCSB (Yahoo!Cloud Serving Benchmark), qui est un framework Open Source largement utilisé pour l'évaluation et la comparaison des différents types de systèmes de données actifs (y compris les bases de données NoSQL comme HBase, Apache Cassandra, Redis, MongoDB, OrientDb, Couchbase, Voldemort, Tarantool, Elasticsearch..).

Le benchmark se compose de deux éléments : un générateur de données et un ensemble de tests de performances pour évaluer les opérations de lecture et mise à jour. Chacun des scénarios de test s'appelle une charge de travail et est défini par un ensemble de fonctionnalités comme le nombre d'enregistrements à charger, le nombre d'opérations à effectuer ainsi que la proportion de lecture, écriture et mise à jour des opérations. Le paquet de benchmark fournit un ensemble de charges de travail par défaut, mais configurable, qui peuvent être exécuté, comme suit :

- Workload A: 50% Read, 50% Update.
- Workload B: 95% Read, 5% Update.
- Workload C: 100% Read.
- Workload D:5% Insert, 95% Read (insère des enregistrements, avec des lectures des données récemment insérées).
- Workload E: 95% Scan, 5% Insert.
- Workload F: 50% Read, 50% Read-Modify-Write.

Afin de mieux comprendre l'optimisation des bases de données et la vitesse d'exécution de la mise à jour des opérations, nous avons créé deux charges de travail supplémentaires possédant les caractéristiques suivantes :

- Workload G: 5% Read, 95% Update.

- Workload H: 100% Update.

Pour évaluer le temps de chargement, nous avons généré 600000 enregistrements, chacun avec 10 champs de 100 octets générés de manière aléatoire sur la clé d'identification de registre, c'est à peu près le total de 1kb par enregistrement. Chaque enregistrement est identifié par une clé composée par la chaîne "user" suivi de plusieurs chiffres, par exemple "user379904308120", qui est la clé d'enregistrement. Chaque champ de l'enregistrement est identifié comme field0, field1, ..., field i respectivement.
L'exécution des charges de travail consistait à lancer 1000 opérations, ce qui signifie qu'il y avait à chaque fois 1000 demandes à la base de données en cours de test.
Dans notre étude, les bases de données orientées graphes n'ont pas été évaluées. Parce que, comme l'a déclaré Armstrong.T, Ponnekanti.V, Dhruba.B, et Callaghan.M dans [30], elles ne doivent pas être évalués selon les scénarios utilisés dans l'analyse des autres types de bases de données NoSQL (orientées colonne, orientées document, et clé-valeur) car l'utilisation des liens entre les enregistrements nécessite une approche différente, donc il y a des points de repère spécifiques développés pour évaluer les performances des bases de données de graphes tels que, XGDBench.

III. Présentation des versions des solutions NoSQL

L'étude comparative développée a permis de distinguer entre les bases de données NoSQL suivantes:
MongoDB : version 2.6.11
CouchBase : version 2.5.2
Cassandra : version 2.2.5
HBase : version 0.94.8
Redis : version 3.0.2
OrientDB : version 2.1.3

III.1 Mise en place d'YCSB

Pour la mise en place de YCSB, il faut d'abord installer Java, Maven et Git dans notre système.

III.1.1 Java

- Le site utilisé pour le téléchargement des archives binaires d'Oracle Java JDK et JRE : http://www.oracle.com/technetwork/java/javase/downloads/index.html

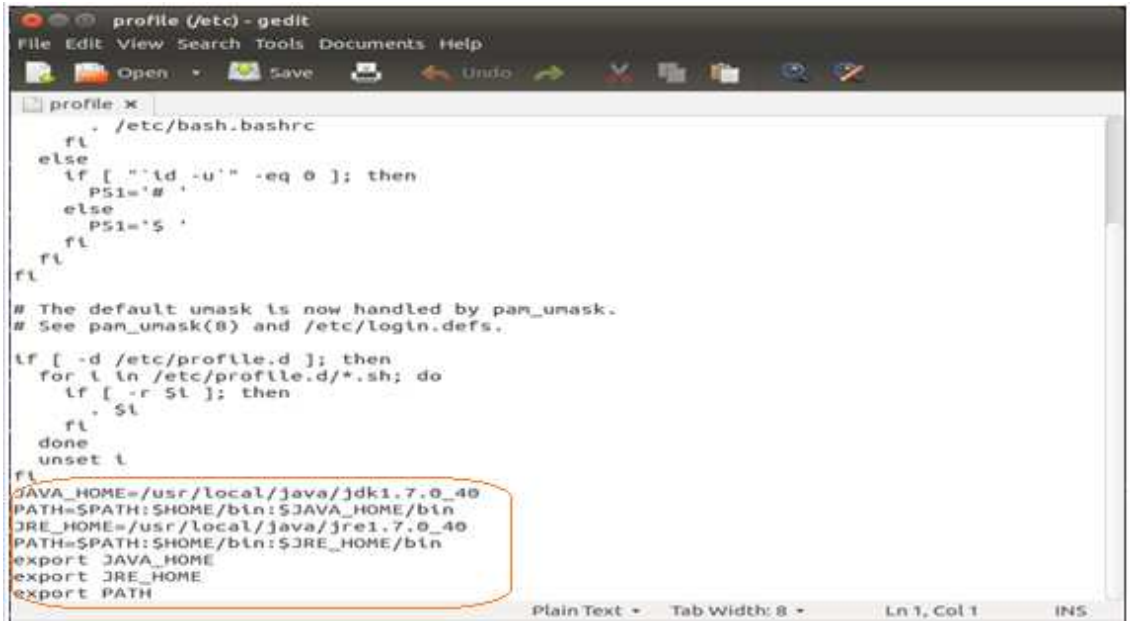
- Après l'accès au répertoire /home/"nom_d'utilisateur"/Téléchargements nous avons copié les archives binaires d'Oracle Java.

```
cd /home/"votre_nom_d'utilisateur"/téléchargements
sudo cp -r jdk-7u40-linux-i586.tar.gz /usr/local/java
sudo cp -r jre-7u40-linux-i586.tar.gz /usr/local/java
```

- Nous avons extrait les archives binaires de Java dans le répertoire /usr/local/java **:**

```
cd /usr/local/java
sudo tar xvzf jdk-7u40-linux-i586.tar.gz
sudo tar xvzf jre-7u40-linux-i586.tar.gz
```

- Après la connexion en tant que root nous avons ajouté ce qui suit dans le fichier descripteur de variables systèmes **.**/etc/profile



- Ce qui suit informe Ubuntu de l'emplacement d'Oracle Java JDK/JRE et indique au système que la nouvelle plateforme Java est disponible :

```
sudo update-alternatives --install "/usr/bin/java" "java"
"/usr/local/java/jre1.7.0_40/bin/java" 1
sudo update-alternatives --install "/usr/bin/javac" "javac"
"/usr/local/java/jdk1.7.0_40/bin/javac" 1
sudo update-alternatives --install "/usr/bin/javaws" "javaws"
```

- Nous avons renseigné Ubuntu qu'Oracle Java sera désormais l'environnement Java par défaut :

```
sudo update-alternatives --set java /usr/local/java/jre1.7.0_40/bin/java
sudo update-alternatives --set javac /usr/local/java/jdk1.7.0_40/bin/javac
sudo update-alternatives --set javaws /usr/local/java/jre1.7.0_40/bin/javaws
```

- Pour que le système prenne en compte la variable PATH il faut exécuter la commande:

```
. /etc/profile
```

III.1.2 Maven

- Le site utilisé pour le téléchargement d'apache Maven :

https://maven.apache.org/download.cgi

- Par la suite nous avons exécuté les commandes suivantes :

```
tar -zxf apache-maven-3.2.5-bin.tar.gz
sudo cp -R apache-maven-3.2.5 /usr/local
sudo ln -s /usr/local/apache-maven-3.2.5/bin/mvn /usr/bin/mvn
```

III.1.3 Git

- Nous avons installé et configuré Git par les commandes suivantes :

```
sudo apt-get install git
git config --global user.name "YOUR NAME"
git config --global user.email "YOUR EMAIL ADDRESS"
```

```
root@ubuntu-HP-630-Notebook-PC: ~
root@ubuntu-HP-630-Notebook-PC:/etc/apt# git config --global user.name infoubunt
u
root@ubuntu-HP-630-Notebook-PC:/etc/apt# git config --global user.email benallal
_tahraoui@yahoo.fr
root@ubuntu-HP-630-Notebook-PC:/etc/apt# git config --list
user.name=infoubuntu
user.email=benallal_tahraoui@yahoo.fr
```

III.1.4 YCSB

- Tout d'abord il faut tester que toutes les dépendances fonctionnent correctement :

```
ubuntu@ubuntu-HP-630-Notebook-PC: ~
ubuntu@ubuntu-HP-630-Notebook-PC:~$ javac -version
javac 1.7.0_40
ubuntu@ubuntu-HP-630-Notebook-PC:~$ mvn -version
Apache Maven 3.2.5 (12a6b3acb947671f09b81f49094c53f426d8cea1; 2014-12-14T18:29:2
3+01:00)
Maven home: /usr/local/apache-maven-3.2.5
Java version: 1.7.0_40, vendor: Oracle Corporation
Java home: /usr/local/java/jdk1.7.0_40/jre
Default locale: fr_FR, platform encoding: UTF-8
OS name: "linux", version: "3.16.0-23-generic", arch: "i386", family: "unix"
ubuntu@ubuntu-HP-630-Notebook-PC:~$ git --version
git version 2.4.6
ubuntu@ubuntu-HP-630-Notebook-PC:~$
```

- Ensuite la source YCSB a été téléchargée à l'aide de la commande :

```
git clone git: //github.com/brianfrankcooper/YCSB.git
```

```
ubuntu@ubuntu-HP-630-Notebook-PC:~$ git clone git://github.com/brianfrankcooper/
YCSB.git
Clonage dans 'YCSB'...
remote: Counting objects: 10094, done.
remote: Total 10094 (delta 0), reused 0 (delta 0), pack-reused 10094
Réception d'objets: 100% (10094/10094), 29.58 MiB | 52.00 KiB/s, fait.
Résolution des deltas: 100% (3842/3842), fait.
Vérification de la connectivité... fait.
ubuntu@ubuntu-HP-630-Notebook-PC:~$
```

- Nous avons compilé YCSB, ceci par :

```
mvn clean package
```

Le résultat de compilation est le suivant :

```
[INFO] Building tar: /home/ubuntu/YCSB/distribution/target/ycsb-0.8.0-SNAPSHOT.tar.gz
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] YCSB Root ..................................... SUCCESS [04:16 min]
[INFO] Core YCSB ..................................... SUCCESS [03:00 min]
[INFO] Per Datastore Binding descriptor .............. SUCCESS [  0.596 s]
[INFO] YCSB Datastore Binding Parent ................. SUCCESS [ 35.141 s]
[INFO] Accumulo DB Binding ........................... SUCCESS [18:05 min]
[INFO] Aerospike DB Binding .......................... SUCCESS [ 20.037 s]
[INFO] Cassandra DB Binding .......................... SUCCESS [04:19 min]
[INFO] Cassandra 2.1+ DB Binding ..................... SUCCESS [12:15 min]
[INFO] Couchbase Binding ............................. SUCCESS [01:10 min]
[INFO] DynamoDB DB Binding ........................... SUCCESS [10:36 min]
[INFO] ElasticSearch Binding ......................... SUCCESS [07:36 min]
[INFO] Geode DB Binding .............................. SUCCESS [27:38 min]
[INFO] Google Cloud Datastore Binding ................ SUCCESS [ 53.262 s]
[INFO] HBase 0.98.x DB Binding ....................... SUCCESS [03:45 min]
[INFO] HBase 0.94.x DB Binding ....................... SUCCESS [02:34 min]
[INFO] HBase 1.0 DB Binding .......................... SUCCESS [16:08 min]
[INFO] Hypertable DB Binding ......................... SUCCESS [01:15 min]
[INFO] Infinispan DB Binding ......................... SUCCESS [01:59 min]
[INFO] JDBC DB Binding ............................... SUCCESS [02:16 min]
[INFO] Kudu DB Binding ............................... SUCCESS [08:08 min]
[INFO] memcached binding ............................. SUCCESS [ 56.065 s]
[INFO] MongoDB Binding ............................... SUCCESS [01:10 min]
[INFO] Oracle NoSQL Database Binding ................. SUCCESS [ 57.889 s]
[INFO] OrientDB Binding .............................. SUCCESS [02:13 min]
[INFO] Redis DB Binding .............................. SUCCESS [  7.203 s]
[INFO] S3 Storage Binding ............................ SUCCESS [ 27.436 s]
[INFO] Solr Binding .................................. SUCCESS [16:42 min]
[INFO] Tarantool DB Binding .......................... SUCCESS [  1.747 s]
[INFO] YCSB Release Distribution Builder ............. SUCCESS [ 45.957 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 02:30 h
[INFO] Finished at: 2016-03-05T14:33:38+01:00
[INFO] Final Memory: 91M/226M
[INFO] ------------------------------------------------------------------------
ubuntu@ubuntu-HP-630-Notebook-PC:~/YCSB$
```

III.2 MongoDB

L'installation de MongoDB se résume dans les étapes suivantes :

- Premièrement il faut d'abord importer la clé publique de 10gen dans notre système en utilisant la commande suivante :

```
sudo apt-key adv –keyserver hkp: //keyserver.ubuntu.com: 80 --recv 7F0CEB10
```

```
hayet@hayet-HP-630-Notebook-PC:~$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
[sudo] password for hayet:
Executing: gpg --ignore-time-conflict --no-options --no-default-keyring --homedir /tmp/tmp.8YJFEEnD1r --no-auto-check-trustdb --trust-model alwa
ys --keyring /etc/apt/trusted.gpg --primary-keyring /etc/apt/trusted.gpg --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
gpg: requesting key 7F0CEB10 from hkp server keyserver.ubuntu.com
gpg: key 7F0CEB10: public key "Richard Kreuter <richard@10gen.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
```

- Nous avons ajouté l'url de MongoDB dans /etc/apt/sources.list.d/mongodb.list, ceci avec :

```
echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstartdist 10gen' |
sudo tee /etc/apt/sources.list.d/mongodb.list
```

- Pour l'installation de MongoDB :

```
sudo apt-get update
sudo apt-get install mongodb-org
```

```
hayet@hayet-HP-630-Notebook-PC:~$ sudo apt-get install mongodb-org
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
Les paquets supplémentaires suivants seront installés :
  mongodb-org-mongos mongodb-org-server mongodb-org-shell mongodb-org-tools
Les NOUVEAUX paquets suivants seront installés :
  mongodb-org mongodb-org-mongos mongodb-org-server mongodb-org-shell mongodb-org-tools
0 mis à jour, 5 nouvellement installés, 0 à enlever et 1203 non mis à jour.
Il est nécessaire de prendre 115 Mo dans les archives.
Après cette opération, 289 Mo d'espace disque supplémentaires seront utilisés.
Souhaitez-vous continuer ? [O/n]
```

-Le lancement et la vérification de la version de MongoDB ont fait par :

```
sudo service mongod start
Mongo --version
```

```
ubuntu@ubuntu-HP-630-Notebook-PC:~$ mongo --version
MongoDB shell version: 2.6.11
```

- L'option « mongo » permet de lancer mongo shell pour interroger les données:

```
ubuntu@ubuntu-HP-630-Notebook-PC:~$ mongo
MongoDB shell version: 2.6.11
connecting to: test
Server has startup warnings:
2016-03-24T21:18:16.858+0100 [initandlisten]
2016-03-24T21:18:16.858+0100 [initandlisten] ** NOTE: This is a 32 bit MongoDB b
inary.
2016-03-24T21:18:16.858+0100 [initandlisten] **       32 bit builds are limited
to less than 2GB of data (or less with --journal).
2016-03-24T21:18:16.858+0100 [initandlisten] **       Note that journaling defau
lts to off for 32 bit and is currently off.
2016-03-24T21:18:16.858+0100 [initandlisten] **       See http://dochub.mongodb.
org/core/32bit
2016-03-24T21:18:16.858+0100 [initandlisten]
>
```

- La commande suivante est utilisée pour arrêter MongoDB :

```
sudo service mongod stop
```

III.3 CouchBase

- Nous avons installé tout d'abord libssl :

```
sudo apt get install libssl1.0.0
```

```
hayet@hayet-HP-630-Notebook-PC:~$ sudo apt-get install libssl1.0.0
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
Les paquets suivants seront mis à jour :
  libssl1.0.0
1 mis à jour, 0 nouvellement installés, 0 à enlever et 1206 non mis à jour.
Il est nécessaire de prendre 862 ko dans les archives.
Après cette opération, 6 144 o d'espace disque supplémentaires seront utilisés.
Réception de : 1 http://dz.archive.ubuntu.com/ubuntu/ vivid-updates/main libssl1.0.0 i386 1.0.1f-1ubuntu11.5 [862 kB]
862 ko réceptionnés en 11s (77,0 ko/s)
Préconfiguration des paquets...
(Lecture de la base de données... 172572 fichiers et répertoires déjà installés.)
Préparation du dépaquetage de .../libssl1.0.0_1.0.1f-1ubuntu11.5_i386.deb ...
Dépaquetage de libssl1.0.0:i386 (1.0.1f-1ubuntu11.5) sur (1.0.1f-1ubuntu9) ...
Paramétrage de libssl1.0.0:i386 (1.0.1f-1ubuntu11.5) ...
Traitement des actions différées (« triggers ») pour libc-bin (2.19-10ubuntu2) ...
hayet@hayet-HP-630-Notebook-PC:~$
```

-Par la suite nous avons exécuté la commande :

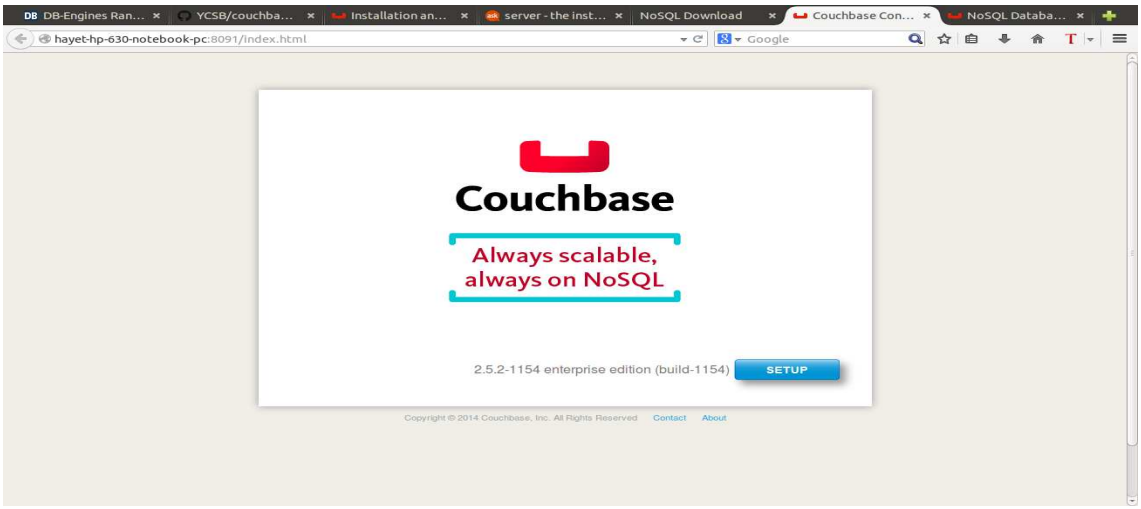
```
sudo dpkg -i couchbase-server-enterprise_2.5.2_x86.deb
```

```
hayet@hayet-HP-630-Notebook-PC:~/couchbase-server-enterprise_2.5.2_x86$ cd
hayet@hayet-HP-630-Notebook-PC:~$ sudo dpkg -i couchbase-server-enterprise_2.5.2_x86.deb
Sélection du paquet couchbase-server précédemment désélectionné.
(Lecture de la base de données... 172572 fichiers et répertoires déjà installés.)
Préparation du dépaquetage de couchbase-server-enterprise_2.5.2_x86.deb ...
libssl1* is installed. Continue installing
Minimum RAM required  : 4 GB
System RAM configured : 1926408 kB

Minimum number of processors required : 4 cores
Number of processors on the system    : 4 cores
Dépaquetage de couchbase-server (2.5.2) ...
Paramétrage de couchbase-server (2.5.2) ...
 * Started couchbase-server

You have successfully installed Couchbase Server.
Please browse to http://hayet-HP-630-Notebook-PC:8091/ to configure your server.
Please refer to http://couchbase.com for additional resources.

Please note that you have to update your firewall configuration to
allow connections to the following ports: 11211, 11210, 11209, 4369,
8091, 8092, 18091, 18092, 11214, 11215 and from 21100 to 21299.

By using this software you agree to the End User License Agreement.
See /opt/couchbase/LICENSE.txt.

insserv: warning: script 'mongod' missing LSB tags and overrides
insserv: Default-Start undefined, assuming empty start runlevel(s) for script `mongod'
insserv: Default-Stop  undefined, assuming empty stop  runlevel(s) for script `mongod'
Traitement des actions différées (« triggers ») pour ureadahead (0.100.0-16) ...
ureadahead will be reprofiled on next reboot
```

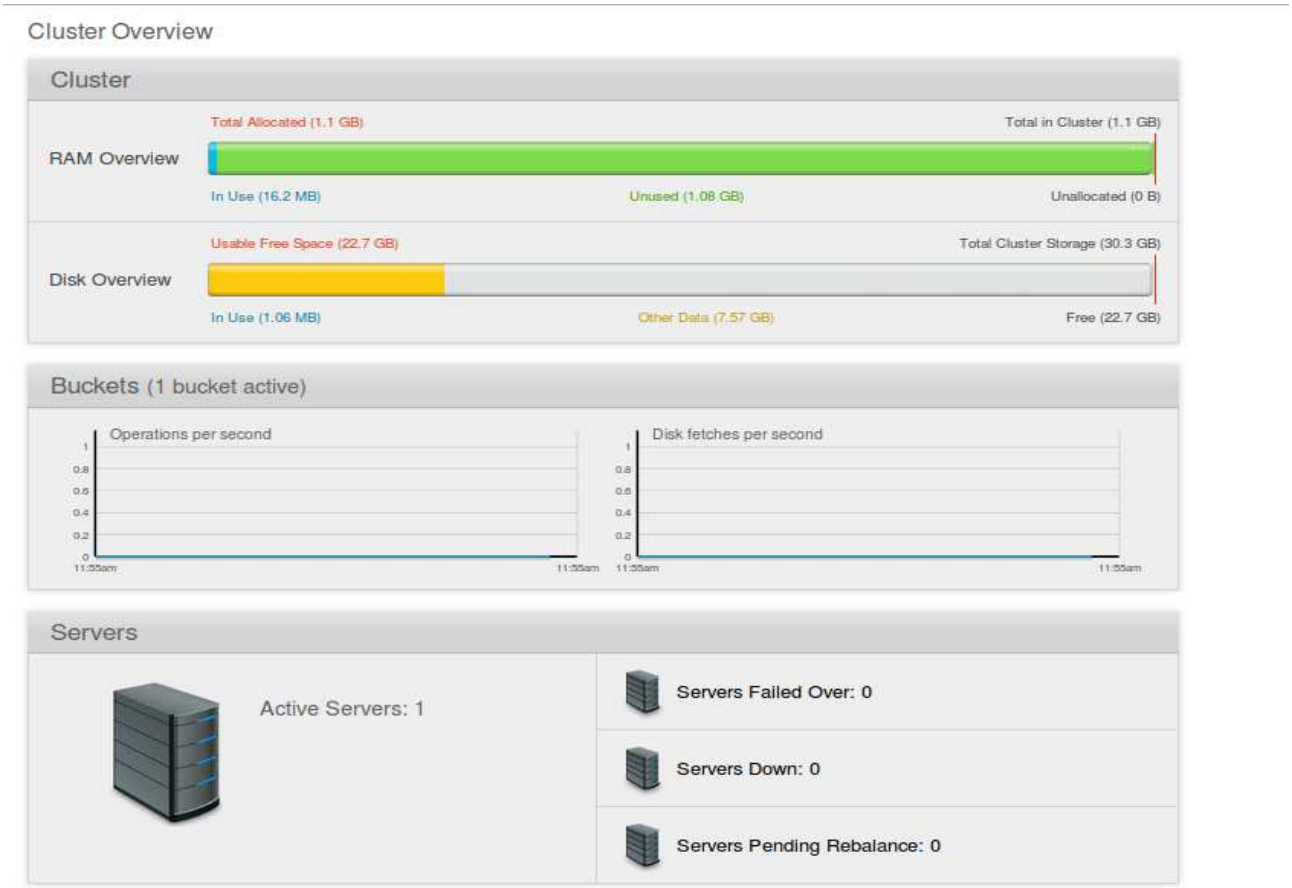
- Démarrage de CouchBase :

```
sudo /etc/init.d/couchbase-server start
```

- Accès par la suite à l'interface graphique :



- Après avoir créé un 'DEFAULT BUKET', et configuré le serveur on a eu le droit à une interface qui permet de visualiser toutes les informations nécessaires:



III.4 Cassandra

-Premièrement nous avons téléchargé Cassandra depuis :

http://cassandra.apache.org/download/

-Ensuite nous avons extrait le fichier téléchargé dans / tmp et le relocalisé dans HOME :

```
mv  /tmp/apache-cassandra* $HOME/cassandra/
```

- Après nous avons créé les répertoires suivants :

```
mkdir $HOME/cassandra/{commitlog,log,data,saved_caches}
```

-La modification de fichier de configuration« cassandra.yaml » a été fait tel que :

```
data_file_directories: – /home/<username>/cassandra/data
commitlog_directory: /home/<username>/cassandra/commitlog
saved_caches_directory: /home/<username>/cassandra/saved_caches
```

- La définition de répertoire Log :

```
nano conf/log4j-server.properties
Modifier:
log4j.appender.R.file=/home/[username]/cassandra/log/system.log
```

- Démarrage de Cassandra :

```
sudo sh ~/cassandra/bin/cassandra
```

- Nous avons vérifié la connexion à Cassandra instance avec nodetool :

```
sudo sh ~/cassandra/bin/nodetool --host 127.0.0.1 ring
```

```
ubuntu@ubuntu-HP-630-Notebook-PC:~$
ubuntu@ubuntu-HP-630-Notebook-PC:~$ sudo sh ~/cassandra/bin/nodetool --host 127.0.0.1 ring

Datacenter: datacenter1
==========
Address    Rack        Status State   Load            Owns                Token
                                                                          9052959872453055328
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -9109978298695564668
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -9103208520055629798
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -9088728925272495462
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8990953282109418954
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8879162094196454305
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8818043954455636578
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8736166131524687185
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8672826070244241237
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8672034074208567492
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8513108959011133362
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8509321140474918759
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8401836198234140580
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8384904085193139809
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8322552040103674892
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8317759320813479851
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8311917673697818931
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8168766447481534252
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8117779682610303280
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7995188559701218116
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7993228157876842222
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7929079150165027774
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7860179453034889017
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7787377756692757750
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7719463177043259343
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7540481259536583977
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7451877945368807830
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7394324980012103665
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7226891571066326829
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7221753625496618653
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7124490554840420174
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7122743112580771342
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7092350078853911637
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -6953720549496199461
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -6937145278184369493
```

- Nous avons démarré cqlsh en utilisant la commande «cqlsh » comme illustré ci – dessous :

```
ubuntu@ubuntu-HP-630-Notebook-PC:~$ sudo sh ~/cassandra/bin/cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 2.2.5 | CQL spec 3.3.1 | Native protocol v4]
Use HELP for help.
cqlsh>
```

III.5 Hadoop

III.5.1 Groupe et utilisateur Hadoop

- Tous d'abord nous avons créé un groupe d'utilisateurs avec tous les droits pour exécuter un nœud Hadoop :

```
sudo addgroup hadoop
sudo adduser --ingroup hadoop hduser
```

```
ubuntu@ubuntu-HP-630-Notebook-PC:~$ sudo adduser --ingroup hadoop hduser
Adding user `hduser' ...
Adding new user `hduser' (1001) with group `hadoop' ...
Creating home directory `/home/hduser' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for hduser
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
ubuntu@ubuntu-HP-630-Notebook-PC:~$
```

Ceci permet de créer un utilisateur « hduser » avec le mot de passe « hduser » et ajouter au groupe hadoop.

III.5.2 Configuration SSH

- Hadoop nécessite un accès SSH pour gérer les différents nœuds, pour cela nous avons installé openssh-server :

```
ubuntu@ubuntu-HP-630-Notebook-PC:~$ sudo apt-get install openssh-server
[sudo] password for ubuntu:
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
Les paquets supplémentaires suivants seront installés :
  libck-connector0 ncurses-term openssh-client openssh-sftp-server
  ssh-import-id
Paquets suggérés :
  libpam-ssh keychain monkeysphere rssh molly-guard
Les NOUVEAUX paquets suivants seront installés :
  libck-connector0 ncurses-term openssh-server openssh-sftp-server
  ssh-import-id
Les paquets suivants seront mis à jour :
  openssh-client
1 mis à jour, 5 nouvellement installés, 0 à enlever et 1203 non mis à jour.
Il est nécessaire de prendre 1,324 ko dans les archives.
Après cette opération, 3,972 ko d'espace disque supplémentaires seront utilisés.
Souhaitez-vous continuer ? [O/n] O
```

- Nous avons généré ensuite une clé SSH pour l'utilisateur hduser :

```
ubuntu@ubuntu-HP-630-Notebook-PC:~$ su hduser
Password:
hduser@ubuntu-HP-630-Notebook-PC:/home/ubuntu$ 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:
d2:12:ba:6a:76:7b:5e:5f:af:bd:4a:4b:2e:e2:9a:cb hduser@ubuntu-HP-630-Notebook-PC
The key's randomart image is:
+--[ RSA 2048]----+
|                 |
|                 |
|      .          |
|     . o         |
|    . o S        |
|     . o         |
|    .   .   +    |
|  o....o...= +   |
| o...+E+....=o+. |
+-----------------+
hduser@ubuntu-HP-630-Notebook-PC:/home/ubuntu$
```

Cette commande va créer une clé RSA avec un mot de passe vide.

- Nous avons donné l'autorisation à cette nouvelle clé fraîchement créée pour accéder au SSH de la machine :

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

- Nous avons testé par la suite la connexion SSH à partir de l'utilisateur hduser :

```
hduser@ubuntu-HP-630-Notebook-PC:~$ ssh 127.0.0.1
The authenticity of host '127.0.0.1 (127.0.0.1)' can't be established.
ECDSA key fingerprint is ac:3d:f7:2a:b5:e6:a9:e1:0c:ab:bb:af:5e:24:d9:2e.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '127.0.0.1' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 14.10 (GNU/Linux 3.16.0-23-generic i686)

 * Documentation:  https://help.ubuntu.com/

Your Ubuntu release is not supported anymore.
For upgrade information, please visit:
http://www.ubuntu.com/releaseendoflife

New release '15.04' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Sun Mar  6 16:06:58 2016 from localhost
hduser@ubuntu-HP-630-Notebook-PC:~$
```

III.5.3 Installation

- Le téléchargement du paquet Hadoop a été fait depuis :

http://www-us.apache.org/dist/hadoop/common/hadoop-2.6.0/

- Nous avons modifié le fichier « sysctl.conf » en ajoutant les lignes :

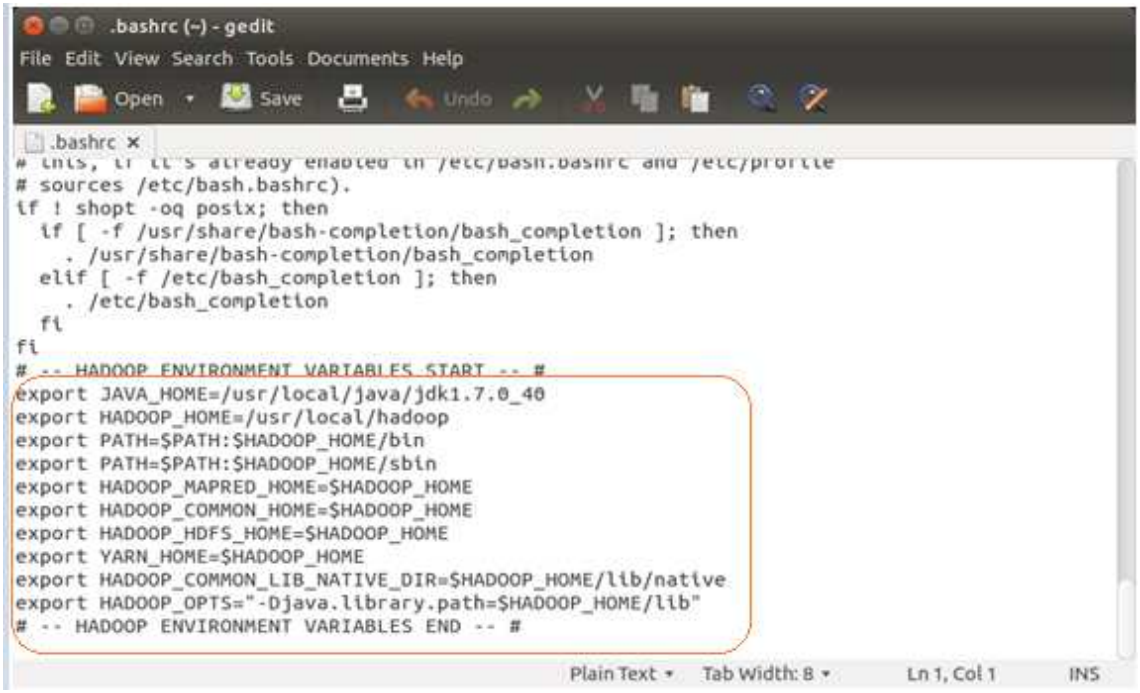
```
# disable ipv6
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
```

- Après la décompression de fichier téléchargé les étapes suivies sont :

```
hduser@ubuntu-HP-630-Notebook-PC:/usr/local$ sudo mv hadoop-2.6.0 /usr/local/hadoop
hduser@ubuntu-HP-630-Notebook-PC:/usr/local$ sudo chown hduser:hadoop -R /usr/local/hadoop
hduser@ubuntu-HP-630-Notebook-PC:/usr/local$ sudo mkdir -p / usr / local / hadoop_tmp / hdfs / NameNode
hduser@ubuntu-HP-630-Notebook-PC:/usr/local$ sudo mkdir -p / usr / local / hadoop_tmp / hdfs / DataNode
hduser@ubuntu-HP-630-Notebook-PC:/usr/local$ sudo chown hduser:hadoop -R /usr/local/hadoop_tmp/
hduser@ubuntu-HP-630-Notebook-PC:/usr/local$
```

III.5.4 Mise à jour des fichiers de configuration Hadoop

- **~/ .bashrc**



- Nous avons exécuté ensuite la commande :

- **hadoop-env.sh**

La variable JAVA_HOME doit être définie dans ce fichier, pour cela nous avons ajouté la ligne :

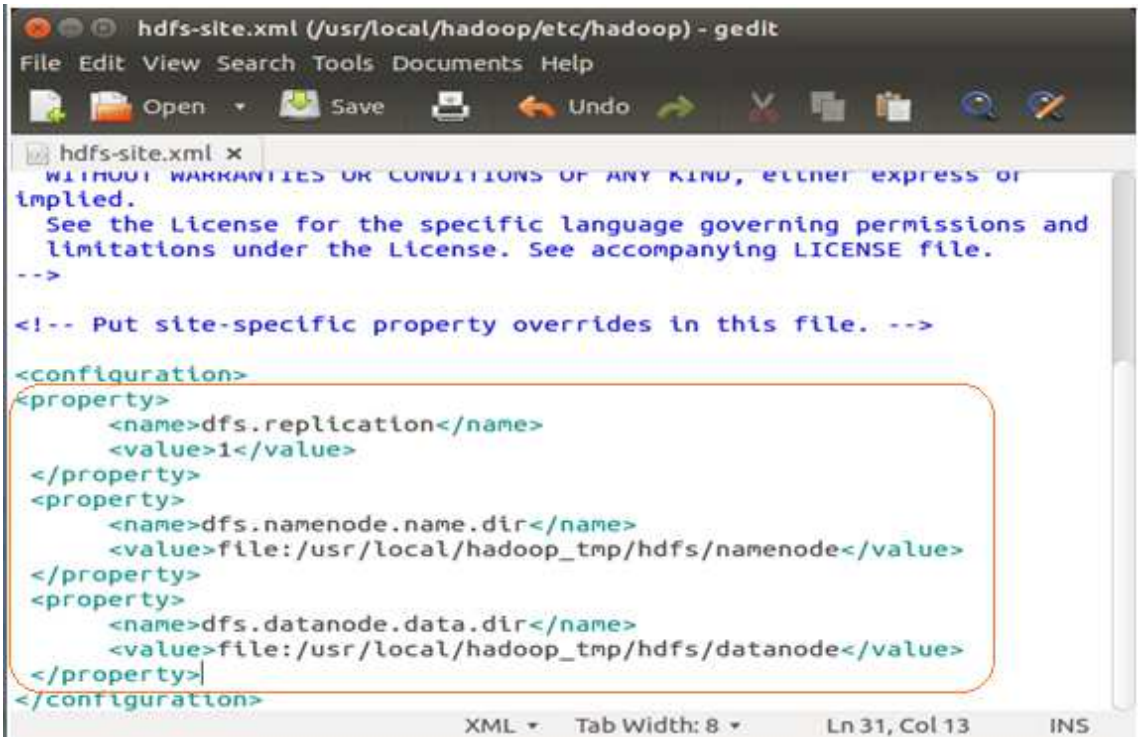
```
export JAVA_HOME=/usr/local/java/jdk1.7.0_40
```

- **core-site.xml**

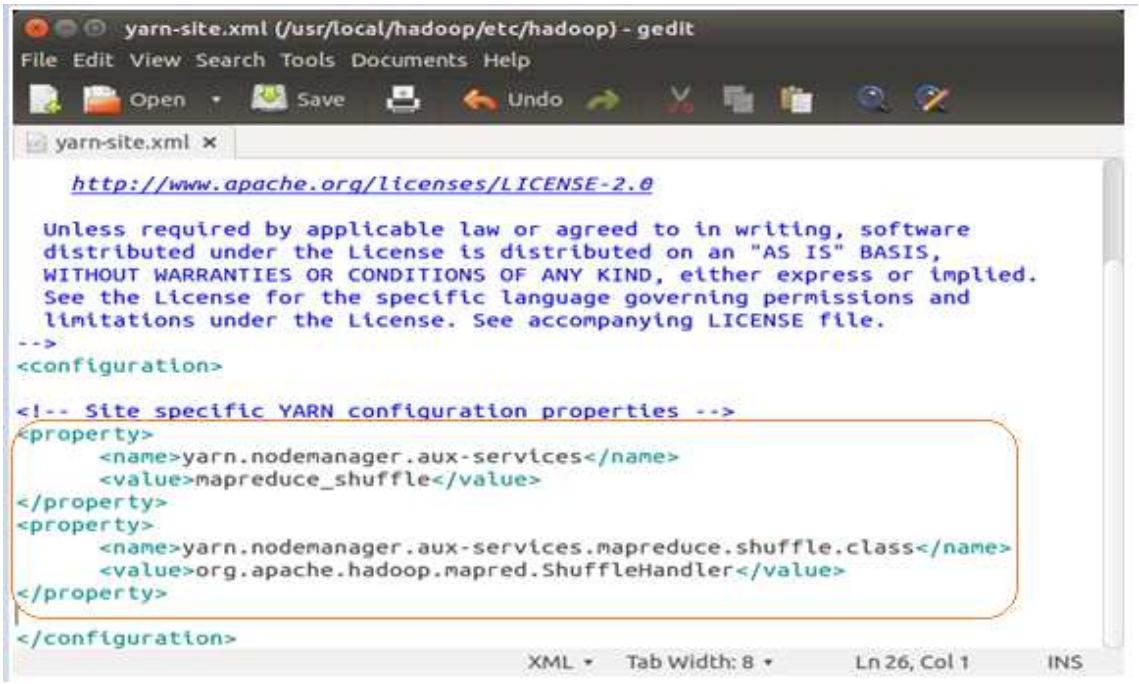
Nous avons ajouté les lignes suivantes dans la balise <configuration>

```
<configuration>
<property>
<name>fs.default.name</name>
<value>hdfs://localhost:9000</value>
</property>
</configuration>
```

- **hdfs-site.xml**



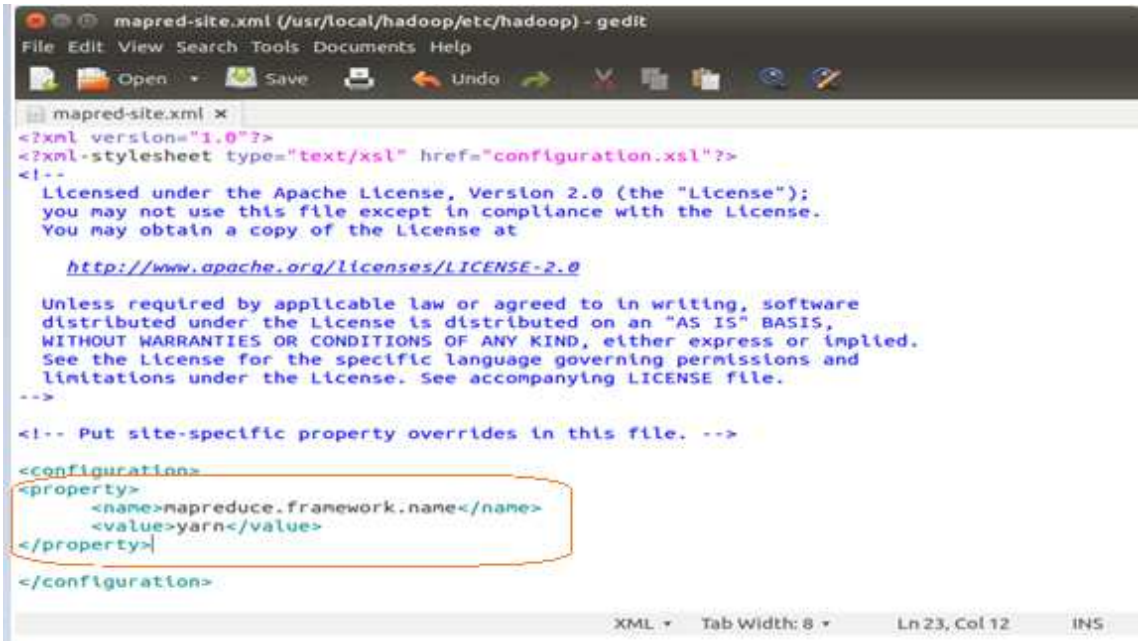
- **yarn-site.xml**



- **mapred-site.xml**

Tout d'abord nous avons copié le modèle du fichier mapred-site.xml.template:

```
hduser@ubuntu-HP-630-Notebook-PC:/usr/local/hadoop/etc/hadoop$ cp /usr/local/hadoop/etc/hadoop/mapred-site.
xml.template  /usr/local/hadoop/etc/hadoop/mapred-site.xml
```



- Par la suite nous avons formaté le nouveau système de fichier HDFS pour Hadoop :

```
hduser@ubuntu-HP-630-Notebook-PC:~$ hdfs namenode -format
16/03/07 12:10:06 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = ubuntu-HP-630-Notebook-PC/127.0.1.1
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 2.6.0
STARTUP_MSG:   classpath = /usr/local/hadoop/etc/hadoop:/usr/local/hadoop/share/hadoop/common/lib/commons-configuration-1.6.jar:/usr/local/hadoop/share/hadoop/common/lib/zookeeper-3.4.6.jar:/usr/local/hadoop/share/hadoop/common/lib/curator-recipes-2.6.0.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-io-2.4.jar:/usr/local/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar:/usr/local/hadoop/share/hadoop/common/lib/java-xmlbuilder-0.4.jar:/usr/local/hadoop/share/hadoop/common/lib/asm-3.2.jar:/usr/local/hadoop/share/hadoop/common/lib/servlet-api-2.5.jar:/usr/local/hadoop/share/hadoop/common/lib/apacheds-kerberos-codec-2.0.0-M15.jar:/usr/local/hadoop/share/hadoop/common/lib/xz-1.0.jar:/usr/local/hadoop/share/hadoop/common/lib/log4j-1.2.17.jar:/usr/local/hadoop/share/hadoop/common/lib/stax-api-1.0-2.jar:/usr/local/hadoop/share/hadoop/common/lib/jettison-1.1.jar:/usr/local/hadoop/share/hadoop/common/lib/xmlenc-0.52.jar:/usr/local/hadoop/share/hadoop/common/lib/jetty-util-6.1.26.jar:/usr/local/hadoop/share/hadoop/common/lib/hadoop-auth-2.6.0.jar:/usr/local/hadoop/share/hadoop/common/lib/junit-4.11.jar:/usr/local/hadoop/share/hadoop/common/lib/jackson-xc-1.9.13.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-beanutils-core-1.8.0.jar:/usr/local/hadoop/share/hadoop/common/lib/jackson-mapper-asl-1.9.13.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-codec-1.4.jar:/usr/local/hadoop/share/hadoop/common/lib/curator-framework-2.6.0.jar:/usr/local/hadoop/share/hadoop/common/lib/jets3t-0.9.0.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-beanutils-1.7.0.jar:/usr/local/hadoop/share/hadoop/common/lib/activation-1.1.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-httpclient-3.1.jar:/usr/local/hadoop/share/hadoop/common/lib/jasper-runtime-5.5.23.jar:/usr/local/hadoop/share/hadoop/common/lib/guava-11.0.2.jar:/usr/local/hadoop/share/hadoop/common/lib/paranamer-2.3.jar:/usr/local/hadoop/share/hadoop/common/lib/jersey-core-1.9.jar:/usr/local/hadoop/share/hadoop/common/lib/hadoop-annotations-2.6.0.jar:/usr/local/hadoop/share/hadoop/common/lib/jasper-compiler-5.5.23.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-cli-1.2.jar:/usr/local/hadoop/share/hadoop/common/lib/api-util-1.0.0-M20.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-logging-1.1.3.jar:/usr/local/hadoop/share/hadoop/common/lib/curator-client-2.6.0.jar:/usr/local/hadoop/share/hadoop/common/lib/slf4j-api-1.7.5.jar:/usr/local/hadoop/share/hadoop/common/lib/api-asn1-api-1.0.0-M20.jar:/usr/local/hadoop/share/hadoop/common/lib/netty-3.6.2.Final.jar:/usr/local/hadoop/share/hadoop/common/lib/jaxb-api-2.2.2.jar:/usr/local/hadoop/share/hadoop/common/lib/hamcrest-core-1.3.jar:/usr/local/hadoop/share/hadoop/common/lib/jetty-6.1.26.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-lang-2.6.jar:/usr/local/hadoop/share/hadoop/common/lib/jackson-jaxrs-1.9.13.jar:/usr/local/hadoop/share/hadoop/common/lib/mockito-all-1.8.5.jar:/usr/local/hadoop/share/hadoop/common/lib/httpcore-4.2.5.jar:/usr/local/hadoop/share/hadoop/common/lib/jsch-0.1.42.jar:/usr/local/hadoop/share/hadoop/common/lib/htrace-core-3.0.4.jar:/usr/local/hadoop/share/hadoop/comm
```

- Et pour démarrer Hadoop il faut taper les deux commandes:

start-dfs.sh

start-yarn.sh

- La commande « jps » permet de visualiser tous les processus actifs et leur ID :

```
hduser@ubuntu-HP-630-Notebook-PC:~$ jps
20807 DataNode
21182 ResourceManager
21314 NodeManager
21598 Jps
20678 NameNode
20983 SecondaryNameNode
hduser@ubuntu-HP-630-Notebook-PC:~$
```

III.6 HBase

Si on réussit Hadoop, HBase est moins compliquée pour qu'elle soit installée. Il faut choisir une version stable conforme à la configuration disponible. Les étapes à suivre pour effectuer l'installation:

-La version stable de HBase a été téléchargée depuis le site :

http://www.apache.org/dyn/closer.cgi/hbase/

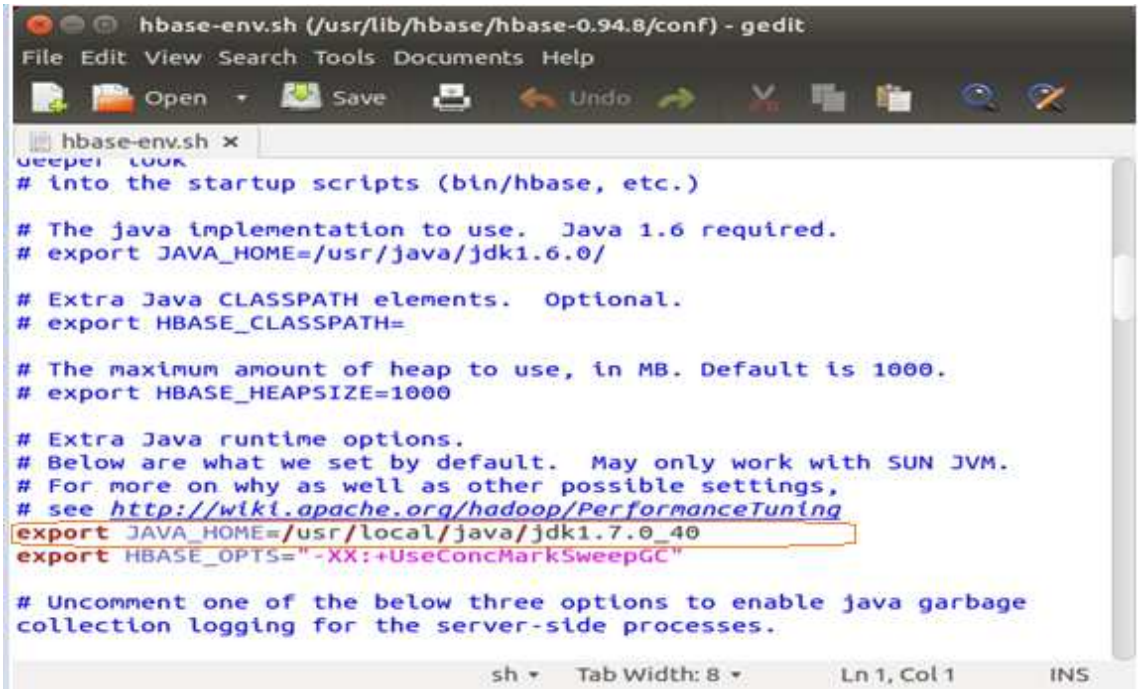
- Nous avons connecté en tant que hduser et décompressé le fichier téléchargé :

```
su hduser
cd Téléchargements
tar -xvf hbase-0.94.8.tar.gz
sudo mkdir /usr/lib/hbase
mv hbase-0.94.8 /usr/lib/hbase/hbase-0.94.8
sudo chown -R hduser:hadoop hbase
```

- Configuration de HBase avec java :

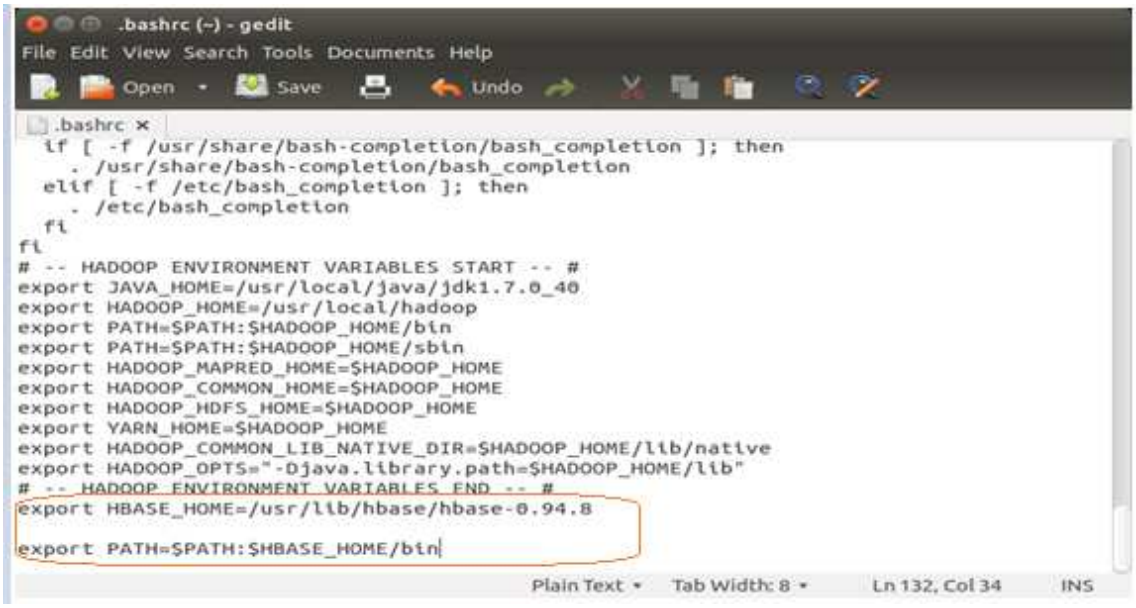
Après l'ouverture de fichier « hbase-env.sh » nous avons défini le chemin de java installé dans notre système :

```
export JAVA_HOME=/usr/local/java/jdk1.7.0_40
```

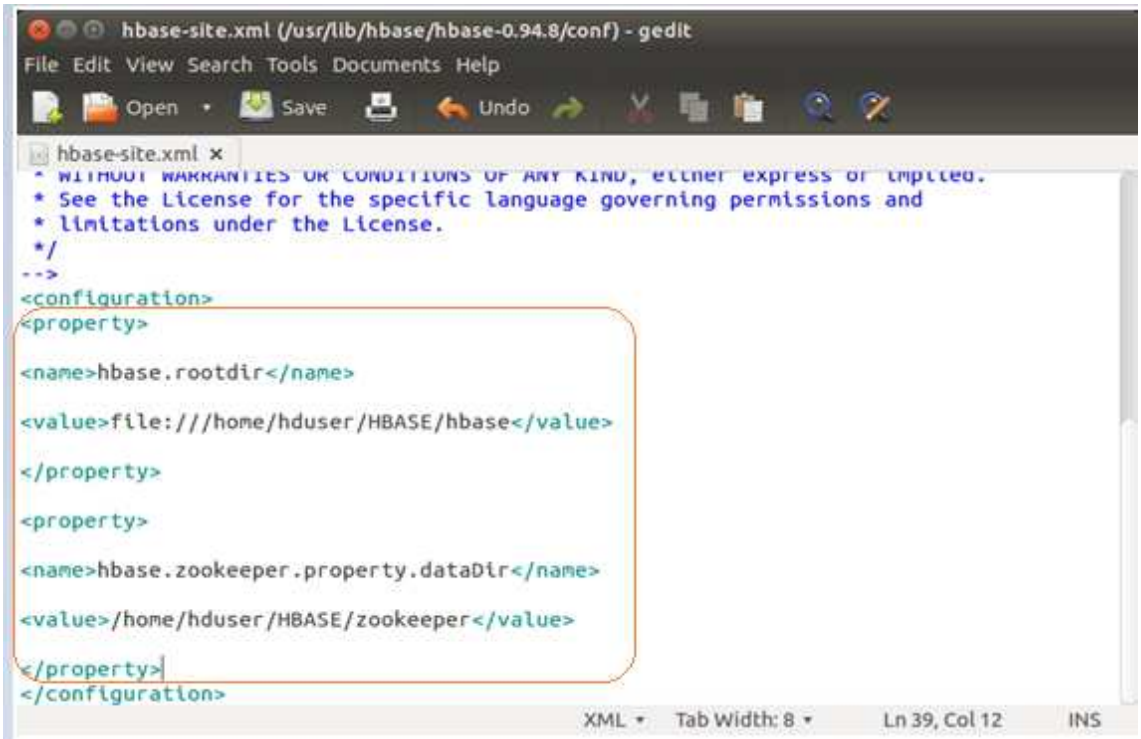


- Définition du chemin de HBASE_HOME dans le fichier « .bashrc » :

gedit ~/.bashrc
Ajouter :
export HBASE_HOME=/usr/lib/hbase/hbase-0.94.8
export PATH=$PATH:$HBASE_HOME/bin



-À ce stade, HBase est installé. Mais avant de commencer, nous avons modifié « HBase-site.xml » :



- Ensuite nous avons changé l'entrée 127.0.1.1dans le répertoire / etc / hosts, par 127.0.0.1 :

```
cd /etc
gedit hosts
```

- Démarrage de HBase :

```
hduser@ubuntu-HP-630-Notebook-PC:/usr/lib/hbase/hbase-0.94.8/bin$ sudo ./start-hbase.sh
starting master, logging to /usr/lib/hbase/hbase-0.94.8/bin/../logs/hbase-root-master-ubuntu-HP-630-Notebook-PC.out
hduser@ubuntu-HP-630-Notebook-PC:/usr/lib/hbase/hbase-0.94.8/bin$
```

- La commande « sudo ./hbase shell» nous a permet de lancer le shell :

```
hduser@ubuntu-HP-630-Notebook-PC:/usr/lib/hbase/hbase-0.94.8/bin$ sudo ./hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.94.8, r1485407, Wed May 22 20:53:13 UTC 2013

hbase(main):001:0>
```

III.7 Redis

- La commande suivante nous a permis de télécharger Redis à partir du terminal :

```
wget http://download.redis.io/releases/redis-3.0.2.tar.gz
```

```
ubuntu@ubuntu-HP-630-Notebook-PC: ~
ubuntu@ubuntu-HP-630-Notebook-PC:~$ wget http://download.redis.io/releases/redis-3.0.2.tar.gz
--2016-03-28 17:21:12--  http://download.redis.io/releases/redis-3.0.2.tar.gz
Resolving download.redis.io (download.redis.io)... 109.74.203.151
Connecting to download.redis.io (download.redis.io)|109.74.203.151|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1360182 (1.3M) [application/x-gzip]
Saving to: 'redis-3.0.2.tar.gz'

63% [================================>                    ] 858,863     26.0KB/s  eta 19s
```

- Pour extraire et compiler Redis, nous avons exécuté les commandes :

```
tar xzf redis-3.0.2.tar.gz
cd redis-3.0.2
make
```

- Nous avons installé ensuite« tcl8.5 », ceci par :

```
sudo apt-get install tcl8.5
```

```
ubuntu@ubuntu-HP-630-Notebook-PC:~$ sudo apt-get install tcl8.5
[sudo] password for ubuntu:
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
Les paquets supplémentaires suivants seront installés :
  libtcl8.5
Paquets suggérés :
  tcl-tclreadline
Les NOUVEAUX paquets suivants seront installés :
  libtcl8.5 tcl8.5
0 mis à jour, 2 nouvellement installés, 0 à enlever et 1207 non mis à jour.
Il est nécessaire de prendre 746 ko dans les archives.
Après cette opération, 3,518 ko d'espace disque supplémentaires seront utilisés.
Souhaitez-vous continuer ? [O/n] O
```

- Puis nous avons exécuté le test recommandé et l'installer :

```
make test
sudo make install
```

- Et finalement nous avons installez le script :

```
cd utils
sudo ./install_server.sh
```

```
ubuntu@ubuntu-HP-630-Notebook-PC:~/redis-3.0.2/utils$  sudo ./install_server.sh
Welcome to the redis service installer
This script will help you easily set up a running redis server

Please select the redis port for this instance: [6379]
Selecting default: 6379
Please select the redis config file name [/etc/redis/6379.conf]
Selected default - /etc/redis/6379.conf
Please select the redis log file name [/var/log/redis_6379.log]
Selected default - /var/log/redis_6379.log
Please select the data directory for this instance [/var/lib/redis/6379]
Selected default - /var/lib/redis/6379
Please select the redis executable path [/usr/local/bin/redis-server]
Selected config:
Port           : 6379
Config file    : /etc/redis/6379.conf
Log file       : /var/log/redis_6379.log
Data dir       : /var/lib/redis/6379
Executable     : /usr/local/bin/redis-server
Cli Executable : /usr/local/bin/redis-cli
Is this ok? Then press ENTER to go on or Ctrl-C to abort.
```

- Ces commandes permettent de démarrer et arrêter Redis :

```
sudo service redis_6379 start
sudo service redis_6379 stop
```

-La vérification de Redis a été faite par :

```
src/redis-cli
```

```
ubuntu@ubuntu-HP-630-Notebook-PC:~/redis-3.0.2$ src/redis-cli
127.0.0.1:6379> set foo bar
OK
127.0.0.1:6379> get foo
"bar"
127.0.0.1:6379>
```

III.8 OrientDB

- En premier lieu, nous avons téléchargé et installé la version 2.1.3 de la communauté OrientDB :

```
wget https://orientdb.com/download.php?file=orientdb-community-2.1.3.tar.gz
```

- Nous avons décompressé et renommé ensuite le fichier téléchargé pour le rendre plus facile à travailler :

```
sudo tar –xf download.php?file=orientdb-community-2.1.3.tar.gz -C /opt
sudo mv /opt/orientdb-community-2.1.3 /opt/orientdb
```

- L'étape suivante consiste à démarrer le serveur et se connecter à la console :

```
cd /opt/orientdb
sudo bin/server.sh
sudo /opt/orientdb/bin/console.sh
```

- Remarque : pour vérifier que le serveur est à l'écoute sur les ports 2424 (pour les connexions binaires) et 2480 (pour les connexions HTTP). Nous avons exécuté dans un autre terminal les commandes :

```
sudo netstat -plunt | grep 2424
sudo netstat -plunt | grep 2480
```

```
hayet@hayet-HP-630-Notebook-PC:~$ sudo netstat -plunt | grep 2424
[sudo] password for hayet:
tcp        0      0 0.0.0.0:2424            0.0.0.0:*               LISTEN      9580/java
hayet@hayet-HP-630-Notebook-PC:~$ sudo netstat -plunt | grep 2480
tcp        0      0 0.0.0.0:2480            0.0.0.0:*               LISTEN      9580/java
hayet@hayet-HP-630-Notebook-PC:~$
```

Et finalement pour se connecter à l'instance de serveur, nous avons exécuté dans la console la commande :

```
orientdb>connect remote:127.0.0.1 root
```

```
OrientDB console v.2.1.3 (build UNKNOWN@r; 2015-10-04 10:56:30+0000) www.orientdb.com
Type 'help' to display all the supported commands.
Installing extensions for GREMLIN language v.2.6.0

orientdb> connect remote:127.0.0.1 root
Enter password:

Connecting to remote Server instance [remote:127.0.0.1] with user 'root'...OK
orientdb {server=remote:127.0.0.1/}>
```

IV. Présentation et analyse des résultats expérimentaux

En préambule, nous soulignons que les résultats retenus pour chaque charge de travail sont les moyennes de plusieurs tests effectués dans au moins trois jours différents.

IV.1 Chargement de données (LoadProcess)

IV.1.1 MongoDB

- La commande est la suivante :

```
./bin/ycsb load mongodb -P workloads/workloada -p
mongodb.url=mongodb://localhost:27017/ycsb?w=0 -s > mongoload.txt
```

- Après le lancement de cette dernière, le résultat de chargement obtenu à partir du terminal est le suivant :

```
Infos: Opened connection [connectionId{localValue:2, serverValue:3}] to localhost:27017
2016-04-23 10:33:52:241 10 sec: 143330 operations; 14333 current ops/sec; est completion in 32 seconds [INSERT: Count=143330, Max=838143, Min=24
, Avg=61,15, 90=89, 99=302, 99.9=733, 99.99=48063]
2016-04-23 10:34:02:242 20 sec: 260155 operations; 11682,5 current ops/sec; est completion in 27 seconds [INSERT: Count=116825, Max=1814527, Min
=24, Avg=79,43, 90=66, 99=73, 99.9=142, 99.99=66431]
2016-04-23 10:34:12:241 30 sec: 361517 operations; 10136,2 current ops/sec; est completion in 20 seconds [INSERT: Count=101373, Max=1131519, Min
=24, Avg=97,64, 90=70, 99=86, 99.9=167, 99.99=122367]
2016-04-23 10:34:22:241 40 sec: 419932 operations; 5841,5 current ops/sec; est completion in 18 seconds [INSERT: Count=58404, Max=2068479, Min=2
4, Avg=166,25, 90=43, 99=72, 99.9=142, 99.99=345343]
2016-04-23 10:34:32:241 50 sec: 436482 operations; 1655 current ops/sec; est completion in 19 seconds [INSERT: Count=16550, Max=2291711, Min=24,
 Avg=520,66, 90=45, 99=78, 99.9=1110, 99.99=686591]
2016-04-23 10:34:42:242 60 sec: 470233 operations; 3375,1 current ops/sec; est completion in 17 seconds [INSERT: Count=33751, Max=2398207, Min=2
4, Avg=324,32, 90=42, 99=71, 99.9=1056, 99.99=483327]
2016-04-23 10:34:52:241 70 sec: 506381 operations; 3614,8 current ops/sec; est completion in 13 seconds [INSERT: Count=36148, Max=2279423, Min=2
4, Avg=278,7, 90=70, 99=77, 99.9=1013, 99.99=392191]
2016-04-23 10:35:02:241 80 sec: 522494 operations; 1611,3 current ops/sec; est completion in 12 seconds [INSERT: Count=16113, Max=1882111, Min=2
4, Avg=617,9, 90=37, 99=71, 99.9=1293, 99.99=942079]
2016-04-23 10:35:12:241 90 sec: 531531 operations; 903,7 current ops/sec; est completion in 12 seconds [INSERT: Count=9037, Max=3092479, Min=24,
 Avg=1089,01, 90=33, 99=77, 99.9=1120, 99.99=1943551]
2016-04-23 10:35:22:242 100 sec: 539643 operations; 811,2 current ops/sec; est completion in 12 seconds [INSERT: Count=8112, Max=2095103, Min=24
, Avg=1172,9, 90=29, 99=65, 99.9=1248, 99.99=2080767]
2016-04-23 10:35:32:241 110 sec: 544596 operations; 495,3 current ops/sec; est completion in 12 seconds [INSERT: Count=4953, Max=2424831, Min=25
, Avg=1788,19, 90=44, 99=71, 99.9=1265, 99.99=2424831]
2016-04-23 10:35:42:241 120 sec: 548407 operations; 381,1 current ops/sec; est completion in 12 seconds [INSERT: Count=3811, Max=3778559, Min=25
, Avg=2247,41, 90=28, 99=69, 99.9=1213, 99.99=3778559]
2016-04-23 10:35:52:241 130 sec: 572967 operations; 2456 current ops/sec; est completion in 7 seconds [INSERT: Count=24566, Max=3303423, Min=24,
 Avg=536,47, 90=70, 99=81, 99.9=983, 99.99=2014207]
avr. 23, 2016 10:36:01 AM com.mongodb.diagnostics.logging.JULLogger log
Infos: Closed connection [connectionId{localValue:2, serverValue:3}] to localhost:27017 because the pool has been closed.
2016-04-23 10:36:01:597 139 sec: 600000 operations; 2889,68 current ops/sec; [CLEANUP: Count=1, Max=367359, Min=367104, Avg=367232, 90=367359, 9
9=367359, 99.9=367359, 99.99=367359] [INSERT: Count=27027, Max=3217407, Min=24, Avg=328,79, 90=69, 99=80, 99.9=879, 99.99=453631]
hayet@hayet-HP-630-Notebook-PC:~/YCSB$
```

- D'autre part le journal de chargement généré par YCSB pour MongoDB est obtenu dans un fichier texte « mongoload.txt » :

mongoload.txt ×

```
mongo client connection created with mongodb://localhost:27017/ycsb?w=0
[OVERALL], RunTime(ms), 139355.0
[OVERALL], Throughput(ops/sec), 4305.5505722794305
[CLEANUP], Operations, 1.0
[CLEANUP], AverageLatency(us), 367232.0
[CLEANUP], MinLatency(us), 367104.0
[CLEANUP], MaxLatency(us), 367359.0
[CLEANUP], 95thPercentileLatency(us), 367359.0
[CLEANUP], 99thPercentileLatency(us), 367359.0
[INSERT], Operations, 600000.0
[INSERT], AverageLatency(us), 226.81303333333332
[INSERT], MinLatency(us), 24.0
[INSERT], MaxLatency(us), 3778559.0
[INSERT], 95thPercentileLatency(us), 72.0
[INSERT], 99thPercentileLatency(us), 172.0
[INSERT], Return=OK, 600000
```

IV.1.2 CouchBase

```
./bin/ycsb load couchbase -s -P workloads/workloada -p couchbase.useJson=false -p couchbase.url=http://127.0.0.1:8091/pools  -p couchbase.bucket=default -p couchbase.password=administrator
```

IV.1.3 Cassandra

```
./bin/ycsb load cassandra2-cql -P workloads/workloada -p hosts=localhost -p columnfamily=data
```

IV.1.4 HBase

```
./bin/ycsb load hbase094 -P workloads/workloada -p columnfamily=family -s
```

IV.1.5 Redis

```
./bin/ycsb load redis -s -P workloads/workloada -p "redis.host=127.0.0.1" -p "redis.port=6379" > redisloadwka1.txt
```

IV.1.6 OrientDb

```
bin/ycsb load orientdb -P workloads/workloada -p orientdb.url=plocal:/tmp/ycsb -p orientdb.user=admin -p orientdb.password=admin
```

Le tableau ci-dessous montre le temps de chargement moyen de chaque base de données:

BASE	MongoDB	Couchbase	Cassandra	HBase	Redis	OrientDb
TEMPS(min)	2,7	1,4	5,6	4	1,7	3,5

Tableau IV.1 : Temps de chargement moyen

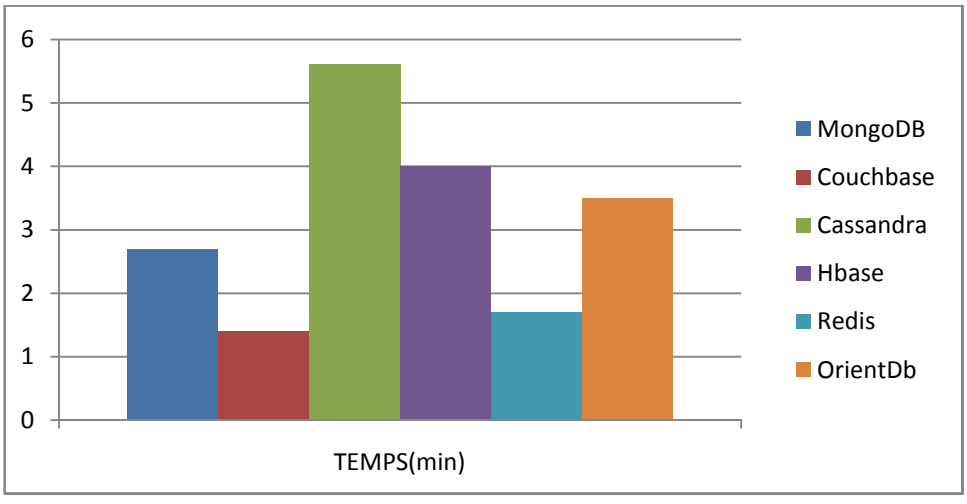


Figure IV.1 : Histogramme de temps de chargement moyen

La figure IV.1 montre les temps expirés pour l'opération de chargement de 600000 enregistrements pour chacune des bases de données testées.

Les bases de données orientées document (CouchBase et MongoDB) ont été plus performantes en moyenne. Nous avons constaté que durant le chargement de 600000 enregistrements, le meilleur temps est obtenu par CouchBase avec un temps de chargement de seulement 1 minute et 4 secondes contre 2 minutes et 7 secondes par MongoDB, ce qui signifie que la première était deux fois plus rapide que la deuxième.

Malgré que Redis était largement plus rapide que MongoDB, les bases de données clé-valeur (Redis et OrientDB) ont démontré une performance moyenne au-dessous de celle des documents, puisqu'OrienDB était deux fois plus lente que Redis et plus lente que les bases orientées document (CouchBase et MongoDB).

Pour le loading des 600000 enregistrements, les deux bases de données orientées colonne ont été moins performantes par rapports aux premières.

D'une manière générale, on peut affirmer pour la phase de chargement, que le meilleur résultat obtenu parmi les six bases testées c'est ce de CouchBase, qui peut être justifié par le fait que cette dernière offre de hautes performances en raison de ses innovations qui lui permettent d'être la première à exploiter une couche de mise en cache RAM intégrée.

IV.2 Exécution des Workloads

Toutes les charges de travail sont constituées d'un ensemble de 1000 opérations variées exécutées sur les 600000 enregistrements déjà chargés dans les bases de données.

IV.2.1 Workload A (50% Read /50% Update)

- **MongoDB**

```
./bin/ycsb run mongodb -s -P workloads/workloada> mongorunwka1.txt
```

mongorunwka1.txt

```
mongo client connection created with mongodb://localhost:27017/ycsb?w=1
[OVERALL], RunTime(ms), 11926.0
[OVERALL], Throughput(ops/sec), 83.85041086701325
[CLEANUP], Operations, 1.0
[CLEANUP], AverageLatency(us), 2471.0
[CLEANUP], MinLatency(us), 2470.0
[CLEANUP], MaxLatency(us), 2471.0
[CLEANUP], 95thPercentileLatency(us), 2471.0
[CLEANUP], 99thPercentileLatency(us), 2471.0
[READ], Operations, 490.0
[READ], AverageLatency(us), 10949.430612244898
[READ], MinLatency(us), 787.0
[READ], MaxLatency(us), 123711.0
[READ], 95thPercentileLatency(us), 23071.0
[READ], 99thPercentileLatency(us), 40127.0
[READ], Return=OK, 490
[UPDATE], Operations, 510.0
[UPDATE], AverageLatency(us), 11206.825490196079
[UPDATE], MinLatency(us), 746.0
[UPDATE], MaxLatency(us), 63615.0
[UPDATE], 95thPercentileLatency(us), 22735.0
[UPDATE], 99thPercentileLatency(us), 38271.0
[UPDATE], Return=OK, 510
```

- **CouchBase**

```
./bin/ycsb run couchbase -s -P workloads/workloada -p couchbase.useJson=false -p couchbase.url=http://127.0.0.1:8091/pools  -p couchbase.bucket=default -p couchbase.password=administrator
```

- **Cassandra**

```
./bin/ycsb run cassandra2-cql -P workloads/workloada -p hosts=localhost -p columnfamily=data-s > cassandrarunwka1.txt
```

- **HBase**

```
./bin/ycsb run hbase094 -P workloads/workloada -p columnfamily=family -s > hbaserunwka1.txt
```

- **Redis**

```
./bin/ycsb run redis -s -P workloads/workloada -p "redis.host=127.0.0.1" -p "redis.port=6379" >redisrunwka1.txt
```

- **OrientDb**

```
bin/ycsb run orientdb -P workloads/workloada -p orientdb.url=plocal:/tmp/ycsb -p orientdb.user=admin -p orientdb.password=admin
```

Le tableau suivant illustre le temps d'exécution moyen de la charge de travail A pour chacune des bases de données:

BASE	MongoDB	Couchbase	Cassandra	HBase	Redis	OrientDb
TEMPS(sec)	10	13	20	28	34	29

Tableau IV.2 : Temps d'exécution de Workload A

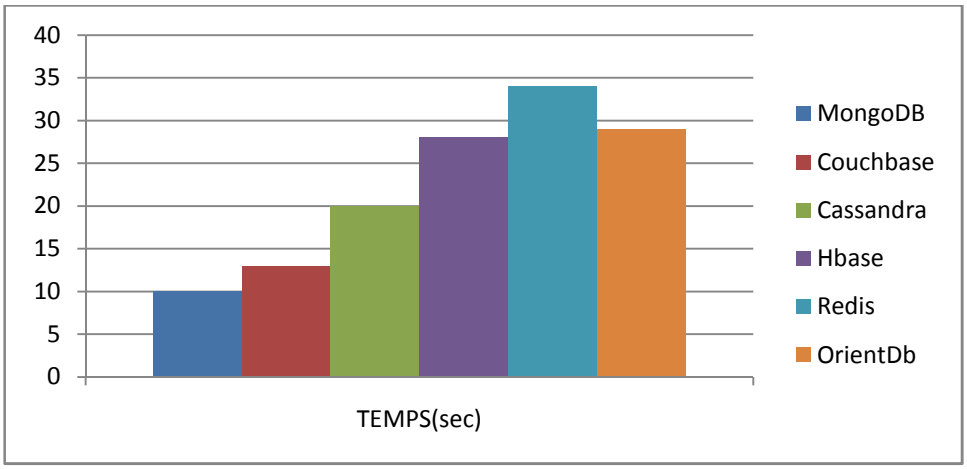


Figure IV.2 : Histogramme de temps d'exécution du Workload A

La Figure IV.2 montre les résultats obtenus durant l'exécution du Workload A composé de 50% d'opérations Read et 50% Update de 1000 opérations, effectuées sur 600000 enregistrements.

Après la lecture des résultats obtenus, nous remarquons que les bonnes performances sont présentées en premier lieu par la catégorie orientée document où MongoDB était la plus rapide suivie par CouchBase. En deuxième lieu on retrouve la catégorie orientée colonne Cassandra avec 20 secondes et HBase avec 28 secondes. Les bases de données clé-valeur sont les moins performantes relativement aux précédentes.

En effet, il faut voir les résultats des charges C et H pour favoriser une solution NoSQL par rapport aux autres.

IV.2.2 Workload B (95% Read, 5% Update)

Le tableau et la figure ci-dessous exposent le temps d'exécution moyen de la charge de travail B pour chacune des bases de données:

BASE	MongoDB	Couchbase	Cassandra	HBase	Redis	OrientDb
TEMPS(sec)	10	8	32	39	24	14

Tableau IV.3 : Temps d'exécution du workload B

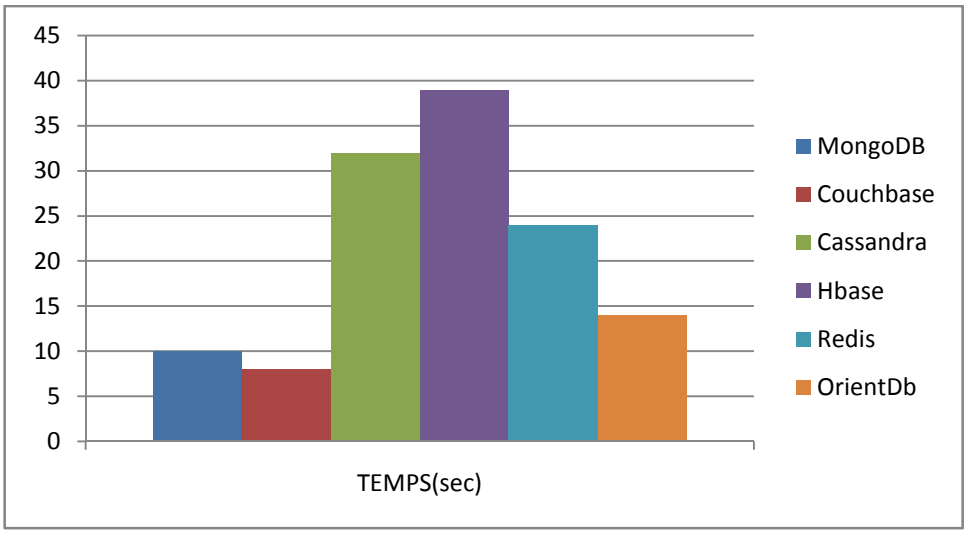


Figure IV.3 : Histogramme de temps d'exécution du Workload B

Les résultats de la figure IV.3 prouve l'ascendance une autre fois des modèles orientés document pour les charges composées principalement d'opérations de lecture. En revanche, ceux des clé-valeur ont prouvé leur performance par rapport aux modèles orientés colonnes. Rappelons aussi que CouchBase s'est imposé une autre fois sur les autres avec une durée de 8 secondes.

IV.2.3 Workload C (100% Read)

Le tableau IV.4 et la figure IV.4 montrent le temps d'exécution moyen de la charge de travail C, composée uniquement d'opérations de lecture, pour chacune des bases de données:

BASE	MongoDB	Couchbase	Cassandra	HBase	Redis	OrientDb
TEMPS(sec)	9	8	28	46	19	11

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

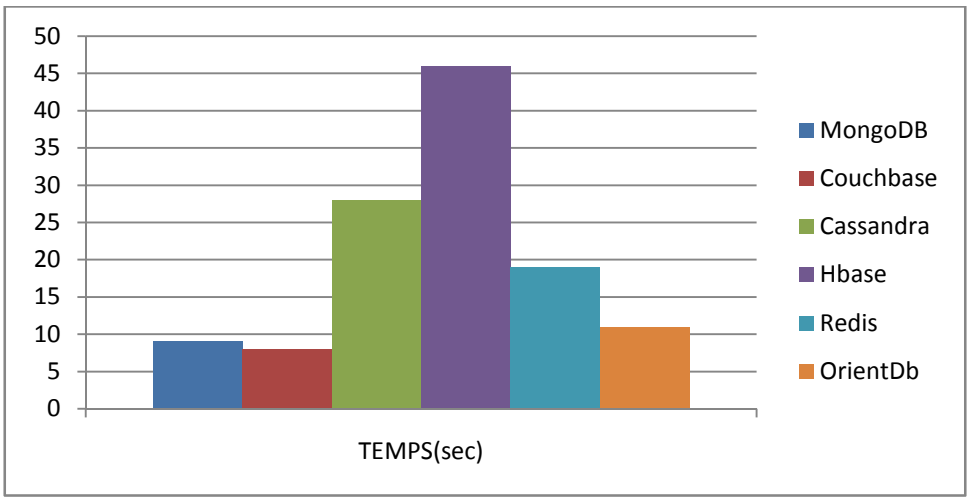


Figure IV.4 : Histogramme de temps d'exécution du Workload C

Pour des opérations purement lecture, les résultats obtenus confirment le classement précédent des bases lors d'exécution de la charge B.

IV.2.4 Workload D (5% Insert, 95% Read)

Le tableau et la figure suivantes affichent le temps d'exécution moyen de la charge de travail D pour chacune des bases de données:

BASE	MongoDB	Couchbase	Cassandra	HBase	Redis	OrientDb
TEMPS(sec)	5	10	16	43	30	8

Tableau IV.5 : Temps d'exécution du Workload D

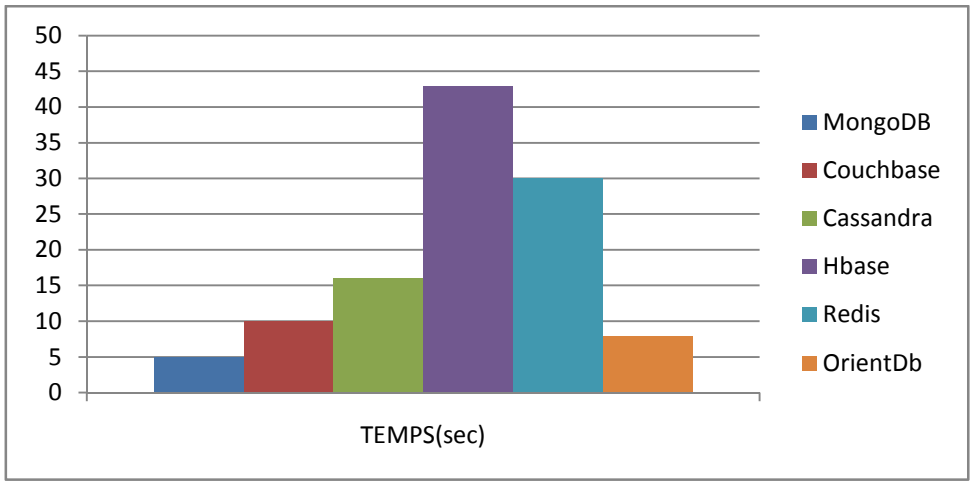


Figure IV.5 : Histogramme de temps d'exécution du Workload D

La charge de travail D est composée principalement d'opérations de lecture (95%) et 5% d'opérations d'insertion de nouveaux enregistrements qui sont insérés puis relus.

En exécutant cette charge de travail, nous confirmons une nouvelle fois la performance des systèmes orientés documents, en particulier MongoDB qui était très rapide en exécutant la tâche en 5 secondes, et la contre-performance des orientés colonnes et surtout celle de HBase avec 43 secondes.

IV.2.5 Workload E (95% Scan, 5% Insert)

Le tableau et la figure ci-dessous présente le temps d'exécution moyen de la charge de travail E pour chacune des bases de données:

BASE	MongoDB	Couchbase	Cassandra	Hbase	Redis	OrientDb
TEMPS(min)	3,75	0,2	0,83	1,12	6,3	16,3

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

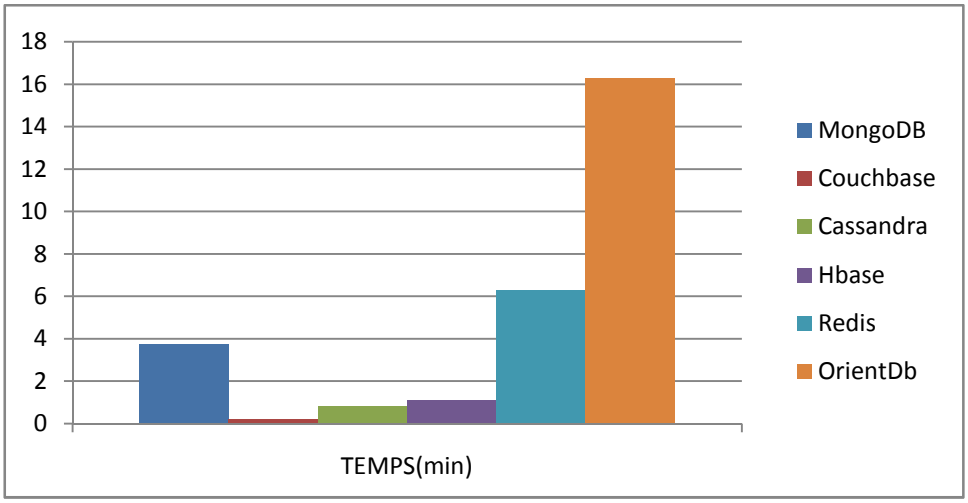


Figure IV.6 : Histogramme de temps d'exécution du Workload E

Cette charge de travail est constituée majoritairement par des opérations de scan rapide 95% et 5% d'insertion de nouveaux enregistrements. Lors de ce test, CouchBase a été la plus performante en ayant le meilleur temps d'exécution (12 secondes) par rapport aux autres bases, ceci est expliqué par le fait qu'elle utilise les vues pour interroger les données.

D'un point de vue globale, les bases de données orientés colonne HBase et Cassandra ont présenté les meilleures performances en moyenne respectivement (67 secondes) et (50 secondes), contrairement au bases clé-valeur qui ont eu le plus faible rendement et en particulier OrientDB qui a été la plus lente en exécutant la charge de travail en 16 minutes et plus.

IV.2.6 Workload F (50% Read, 50% Read-Modify-Write)

Le tableau ci-dessous montre le temps d'exécution moyen de la charge de travail F, moitié lecture et moitié écriture, pour chacune des bases de données:

BASE	MongoDB	Couchbase	Cassandra	HBase	Redis	OrientDb
TEMPS(sec)	11	18	27	36	18	23

Tableau IV.7 : Temps d'exécution du Workload F

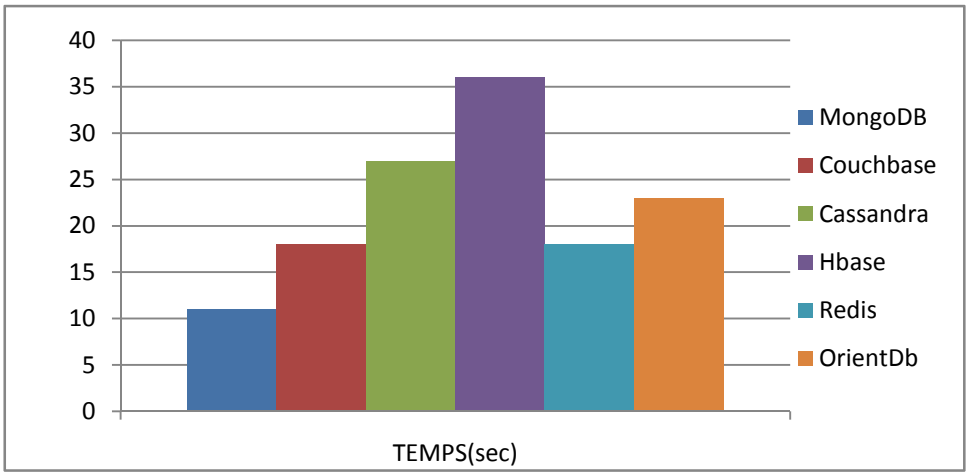


Figure IV.7 : Histogramme de temps d'exécution du Workload F

La Figure IV.7 présente les résultats obtenus après exécution du Workload F composé de 50% de lecture, pour l'autre 50% : les enregistrements sont lus en premier lieu, mis à jour après et ensuite sauvegardés. Nous avons constaté une nouvelle fois la contre-performance des systèmes orientés colonnes à savoir HBase et Cassandra en raison de leurs contraintes de lecture par rapport à la mise à jour. En revanche, Redis et CouchBase ont obtenus les mêmes bons résultats (18 sec).

D'autre part la meilleure performance est celle de MongoDB qui est entrain de prouver son efficacité avec CouchBase pour les opérations de lecture.

IV.2.7 Workload G (5% Read, 95% Update)

Le tableau et la figure suivantes exposent le temps d'exécution moyen de la charge de travail G pour chacune des bases de données:

BASE	MongoDB	Couchbase	Cassandra	HBase	Redis	OrientDb
TEMPS(sec)	12	12	10	7	18	21

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

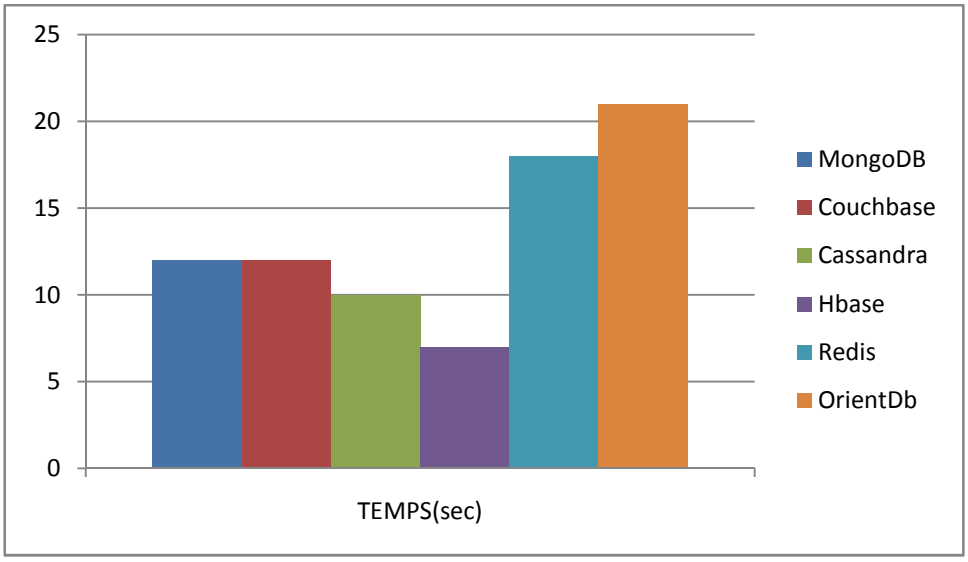


Figure IV.8 : Histogramme de temps d'exécution du Workload G

Les résultats révèlent que pour une charge composée principalement de mise à jour, les bases de données orientées colonne reprennent l'ascendance sur les autres architectures, en revanche celles des clé-valeur ont été largement au-dessous.

Notons aussi les mêmes résultats obtenus par CouchBase et MongoDB.

IV.2.8 Workload H (100% Update)

Le tableau et la figure ci-dessous présentent le temps d'exécution moyen de la charge de travail H pour chacune des bases de données:

BASE	MongoDB	Couchbase	Cassandra	HBase	Redis	OrientDb
TEMPS(sec)	10	15	6	4	25	21

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

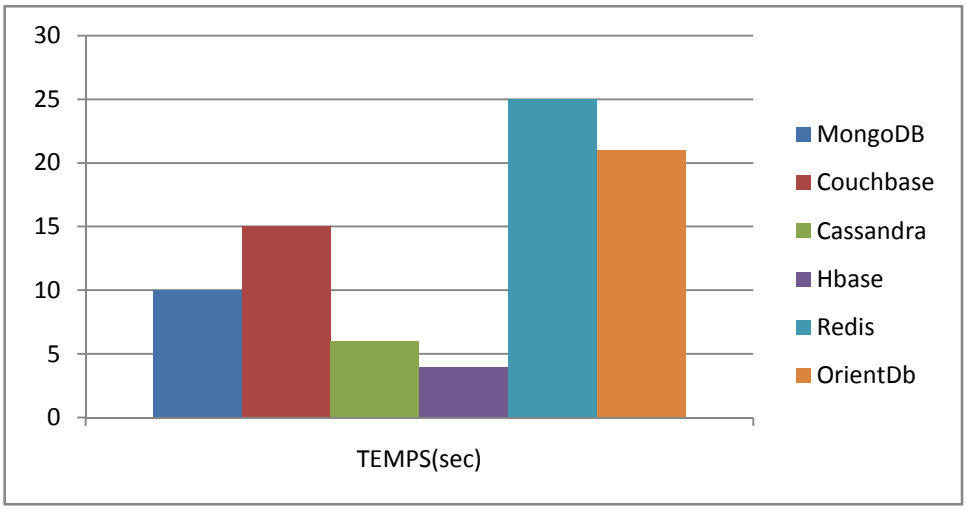


Figure IV.9 : Histogramme de temps d'exécution du Workload H

Pour des opérations purement mises à jour, les bases de données orientées colonne HBase et Cassandra confirment leurs performances atteintes lors d'exécution de la charge G par rapport à toutes les autres systèmes NoSQL.

IV.3 Récapitulatif de performance de l'ensemble des Workloads

Le tableau et la figure IV.10 récapitulent les résultats obtenus par les bases de données NoSQL pour toutes les charges de travail (A + B + C +D+ E+ F + G + H).

BASE	MongoDB	Couchbase	Cassandra	Hbase	Redis	OrientDb
Total(min)	5	2	3	5	9	18

Tableau IV.10 : Temps d'exécution global

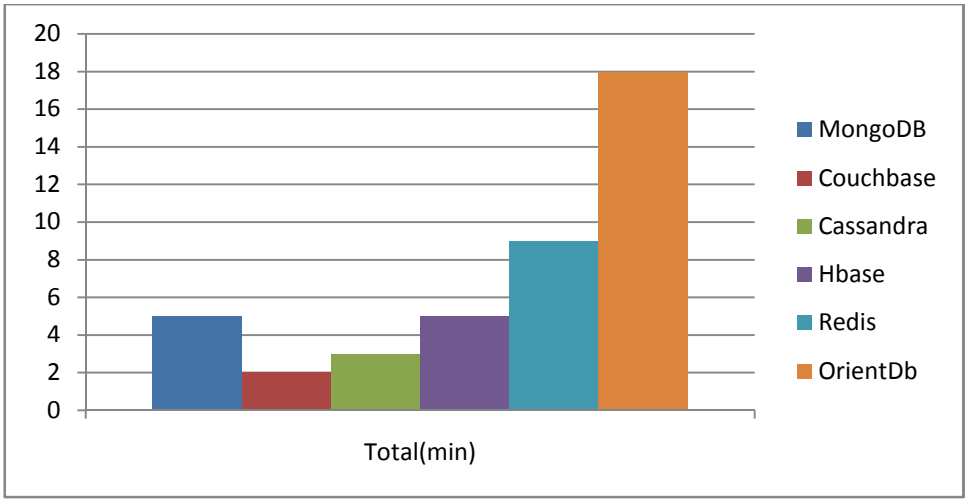


Figure IV.10 : Histogramme de temps d'exécution global

D'un point de vue globale, les modèles orientés documents et orientés colonnes sont largement plus performants par rapport aux modèles clé-valeur.

Le meilleur temps d'exécution est présenté par CouchBase avec 2 minutes seulement contre Cassandra qui se place en seconde position avec 3 minutes seulement.

V. Evaluation globale de MongoDB, CouchBase, HBase, Cassandra, Redis, OrientDB

Les résultats expérimentaux des différents tests effectués ont permis d'évaluer et comparer les trois types de bases de données NoSQL : orientées document, orientées colonne et clé valeur, en s'appuyant sur le temps d'exécution des différentes charges de travail.

Après la lecture des résultats, nous pouvons conclure que les critères de choix dépendent des besoins applicatifs et de la nature d'opérations exécutées sur les données.

Pour une question de performance et des fins d'optimisation, on peut spécialiser les bases de données NoSQL selon le modèle approprié et selon le contexte d'utilisation de ces dernières. Parmi les solutions NoSQL étudiées, on affirme qu'il y a celles optimisées pour les lectures, celles pour les mises à jour et d'autres pour les opérations de scan:

- Pour des opérations purement lecture il faut s'orienter vers les architectures orientées document à savoir CouchBase et MongoDB.
- Pour les opérations de mis à jour lourde, il est très intéressant d'adopter des architectures orientées colonne.
- Pour des opérations de scan, CouchBase, HBase, et Cassandra ont prouvé leur performance.
- Pour les architectures clé-valeur, beaucoup d'efforts restent à fournir par les concepteurs pour améliorer leurs performances.

VI. Conclusion

Ce chapitre nous a permis de tirer plusieurs renseignements concernant la performance.

A ce jour, il n'existe pas encore un seul "gagnant prend tout" parmi les solutions NoSQL disponibles dans le marché. Selon le contexte d'utilisation et les conditions de déploiement, il est presque toujours possible pour une base de données NoSQL de surperformer l'autre et de démontrer d'excellentes performances lorsque les règles changent.

Conclusion générale

Notre projet de fin d'études consistait à une étude comparative entre les différentes familles des solutions NoSQL : orientées document, orientées colonne et clé valeur à savoir MongoDB, CouchBase, Cassandra, Hbase, Redis et OrientDB.

L'objectif fixé de cette étude est d'évaluer les performances de ces bases de données en insérant en premier lieu 600 000 enregistrements, ensuite en lançant un ensemble de tests sous forme de charge de travail composées de 1000 opérations chacune de différentes natures : lecture, scan ou mise à jour.

L'outil employé pour arbitrer les six systèmes est Yahoo! Cloud Serving Benchmark, qui est très recommandé pour ce genre d'études dans le domaine des bases de données NoSQL.

Après la lecture et l'analyse des résultats expérimentaux, on peut affirmer qu'il y a des bases de données très performantes pour des workloads particuliers, contrairement à d'autres qui étaient meilleures dans d'autres charges de travail.

En conclusion, on peut retenir que le choix d'utilisation d'un SGBD dépend d'un ensemble de paramètres en relation avec l'environnement dans lequel les données sont exploitées. En effet le type de données et le type des traitements effectués sur ces données sont des indices importants pour définir la solution à adopter. La fréquence estimée de lecture, d'écriture, de mise à jour ainsi que la taille des données sont les facteurs essentiels déterminants dans le choix d'une alternative parmi d'autres. Actuellement la tendance vers une favorisation d'une solution NoSQL précise est loin d'être indiscutable à cause du nombre important des systèmes existants. Plusieurs solutions Open Source et payantes sont présentées aux différents acteurs concernés.

Perspectives

Enfin, on peut estimer que l'objectif tracé au préalable a été largement atteint, néanmoins ce travail pourrait être complété et prolongé sur plusieurs aspects, ainsi on peut souligner un ensemble de perspectives et de piste de recherches à explorer, citons :

- Etaler notre étude à d'autres solutions NoSQL à savoir : Elasticsearch, Memcached, Amazon DynamoDB, CouchDB, Accumulo et autres.
- Multiplier le nombre d'enregistrements pour atteindre ou dépasser le million.
- Diversifier les charges de travail en créant d'autres.

- Etendre notre travail d'un environnement monoposte à un environnement distribué en augmentant à chaque fois le nombre de machines connectées.

Liste D’abréviations

BDDR : Base De Données Relationnelles.

SQL : Structured Query Language.

SGBD : Système de Gestion de Base de Données.

SGBDR : Système de Gestion de Base de Données Relationnelles.

ACID : Atomicity, Consistancy, Isolation, Durability.

NoSQL : Not only SQL.

CAP : Consistency, Availability, Partition tolerance.

BASE : Basically Available, Soft-state, Eventual consistency.

CRUD : Create, Read, Update, Delete.

JSON : JavaScript Object Notation.

BSON : Binary JSON.

HDFS : Hadoop Distributed File System.

Liste des Figures

Liste des Tableaux

Références Bibliographiques

[1] Robin Bloor, L'échec des Bases de Données Relationnelles, L'essor de la Technologie Objet et le Recours aux Bases de Données Hybrides. Baroudi Bloor International Inc., 2003, 2004.

[2] Antoine Cornuéjols, BASES DE DONNÉES CONCEPTS ET PROGRAMMATION, AgroParisTech, Spécialité Informatique (2009-2010).

[3] Laurent Audibert, Base de Données et Langage SQL, Cours-BD, Institut Universitaire de Technologie de Villetaneuse, Département Informatique.

[4] Andreas Meier, Introduction pratique aux bases de données relationnelles, Ed 2, (Collection IRIS), Springer-Verlag France 2002, 2006.

[5] Christine Parent, et Dr. Aida Boukottaya, cours de Bases de Données Avancées, (chapitre 5) l'école des Hautes Etudes Commerciales (HEC), Université de Lausanne.

[6] Matteo Di Maglie, Adoption d'une solution NoSQL dans l'entreprise, Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES, Carouge, 12 septembre 2012 Haute École de Gestion de Genève (HEG-GE).

[7] Mathieu Roger, Bases NoSQL, synthèse d'étude et projets d'interlogiciels, octera [AT] octera [DOT] info.

[8] Kouedi Emmanuel, Approche de migration d'une base de données relationnelle vers une base de données NoSQL orientée colonne, Mémoire présenté en vue de l'obtention du diplôme de MASTER II RECHERCHE, Option **:** S.I & G.L ; Université de YAOUNDE I. Mai 2012.

[9] Xavier MALETRAS, Le NoSQL- Cassandra, Thèse Professionnelle, université Paris 13, 27/05/2012.

[10] Marie-France Lasalle, Cours Du Relationnel a l'objet : Limites du Relationnel, 24/1/2006. Disponible sur :

http://circe.univ-fcomte.fr/Marie-France Lasalle/bda/COURSHTM/cours/chap01/lec02/page 01.htm

[11] Mark Whitehorn, Quand faut-il envisager d'utiliser une base de données NoSQL (plutôt qu'une base relationnelle) ? , University of Dundee.
Disponible sur : http://www.lemagit.fr/conseil/Quand-envisager-NoSQL

[12] Rudi Bruchez, Les bases de données NOSQL et le BIGDATA comprendre et mettre en œuvre ,2iéme édition : ÉDITIONS EYROLLES , www.editions-eyrolles.com.

[13] Meyer Léonard, L'AVENIR DU NoSQL Quel avenir pour le NoSQL ?, 2014.
Disponible sur :
http://www.leonardmeyer.com/wp-content/uploads/2014/06/avenirDuNoSQL.pdf

[14] Bernard Espinasse, Introduction aux systèmes NoSQL(Not Only SQL), Aix-Marseille Université (AMU) Ecole Polytechnique Universitaire de Marseille Avril 2013.

[15] Solid IT "Classement NoSQL, Solid IT", http://db-engines.com/en/ranking

[16] Adriano Girolamo PIAZZA, NoSQL Etat de l'art et benchmark ;Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES ; Genève, 9 octobre 2013
Haute École de Gestion de Genève (HEG-GE).

[17] Lionel Heinrich, Architecture NoSQL et réponse au Théorème CAP, Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES en Informatique de Gestion,
Genève, le 26 octobre 2012 Haute École de Gestion de Genève (HEG-GE).

[18] Alex Objelean, Introduction to Couchbase-NoSQL Document Database, SEPTEMBER 25, 2015.

[19] http://developer.couchbase.com/documentation/server/4.1/architecture/architecture-intro.html

[20] Henry Potsangbam**,** Learning CouchBase : Design documents and implement real world e-commerce applications with Couchbase, November2015.

[21] Aurélien Foucret, NoSQL : une nouvelle approche du stockage et de la manipulation des données, SMILE open source solutions.

[22] Hadoop: Big data analysis framework, tutorials point. Disponible sur : http://www .tutorialspoint.com/hadoop/

[23] Julien Gerlier et Siman Chen, Prototypage et évaluation de performances d'un service de traçabilité avec une architecture distribuée basée sur Hadoop, 2011.

[24] Cyrille Chausson, Tout savoir sur Hadoop : Vulgarisation de la technologie et les stratégies de certains acteurs,TechTarget, 2014.

[25] Acef Mohamed, UTILISATION DU MODELE MAPREDUCE DANS LES DIFFERENTS SYSTEMES NOSQL: ETUDE COMPARATIVE, Mémoire Pour l'Obtention du Diplôme de Post Graduation Spécialisée, 2015.

[26] Hadrien Furrer, SQL, NoSQL, NewSQL stratégie de choix., Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES, Genève, le 16 Février 2015 Haute École de Gestion de Genève (HEG-GE).

[27] http://orientdb.com/docs/last/Tutorial-Document-and-graph-model.html

[28] http://blog.itpub.net/22664653/viewspace-709925/

[29] https://docs.mongodb.com/v2.6/core/replica-set-secondary/

[30] Armstrong, T., Ponnekanti, V., Dhruba, B., and Callaghan, M.: LinkBench: a database benchmark based on the Facebook social graph, In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13), ACM, New York, NY, USA, 1185-1196.

Résumé

Ce mémoire s'ajoute aux différents travaux de recherche dans le domaine des bases de données NoSQL « Not only SQL ». Ces nouveaux modèles proposent une nouvelle manière d'organisation et de stockage de données conçue principalement pour remédier aux contraintes imposées par les propriétés ACID sur les modèles relationnels.

Notre objectif consistait à développer une étude comparative, entre six solutions NoSQL très utilisées dans le marché, à savoir :MongoDB, CouchBase, Cassandra, HBase, Redis, OrientDB, pour proposer aux décideurs, des éléments d'information pour des éventuels choix de la meilleure solution appropriée pour leurs entreprises.

Le Benchmark utilisé pour départager ces solutions, est le Yahoo! Cloud Serving Benchmark.

Mots-clés : NoSQL,MongoDB, CouchBase, Cassandra,HBase, Redis, OrientDB,YCSB.

Abstract

This memory is in addition to various research projects in the field of NoSQL databases "Not only SQL". These new models offer a new way of organizing and storing data designed primarily to address the constraints imposed by the ACID properties of the relational models.

Our aim was to develop a comparative study between six NoSQL solutions widely used in the market, namely: MongoDB, Couchbase, Cassandra, HBase, Redis, OrientDB, to propose to the decision-makers, the elements of information for possible choices of the best solution suited for their companies.

The Benchmark used to decide these solutions is the Yahoo! Cloud Serving Benchmark.

Keywords: NoSQL, MongoDB, Couchbase, Cassandra, HBase, Redis, OrientDB, YCSB.

ملخص

هذه المذكرة تضاف إلى المشاريع البحثية المختلفة في مجال قواعد البيانات NoSQL "ليس فقط SQL". هذه النماذج الجديدة توفر وسيلة جديدة لتنظيم وتخزين البيانات تهدف في المقام الأول لمعالجة القيود التي تفرضها خصائص ACID من النماذج العلائقية.

كان هدفنا تطوير دراسة المقارنة بين ستة حلول NoSQL تستخدم على نطاق واسع في السوق، وهي: MongoDB، Couchbase، Cassandra، HBase، Redis، OrientDB لإقتراح المعلومات المناسبة من أجل إختيار أفضل حل مناسب.

المعيار المتبع في تحديد هذه الحلول هو YCSB.

الكلمات المفتاحية: NoSQL، MongoDB، Couchbase، Cassandra، HBase، Redis، OrientDB، YCSB.