REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider – BISKRA
Faculté des Sciences Exactes, des Sciences de la Nature et de la Vie
Département d'informatique

N° d'ordre : SIOD /M2/2020

## Mémoire

Présenté pour obtenir le diplôme de master académique en

# Informatique

Parcours : **Informatique décisionnelle et optimisation SIOD**

---

# Etude comparative entre des systèmes NoSQL

---

**Par :**

**BOUDJEMAA SAMIRA**

Soutenu le 15/09/2020, devant le jury composé de :

| Nom et prénom | Grade | Président |
|---|---|---|
| Ben Seghier Nadia | MCA | Rapporteur |
| Nom et prénom | Grade | Examinateur |

# *Remerciements*

Mes premiers remerciements à Dieu Tout-Puissant pour le courage et la volonté qu'il nous a donnée pour mener à bien ce travail.

Je tiens à remercier sincèrement Mm Ben Seghier Nadia, qui en tant qu'encadreur de ce PFE, pour son assistance permanente, pour ces conseils et son suivi durant la réalisation de ce projet.

J'exprime ma gratitude et mes remerciements à « **M. LAbed Yassine** », directeur de la bibliothèque publique de lecture "**Mohamed Essami**" à Biskra, pour m'avoir permis de terminer ma carrière universitaire.

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

Je tiens à remercier

Tous mes enseignants du département d'informatique de la faculté des sciences à l'Université Mohamed KHIDER à Biskra, qui ont contribué à notre formation tout au long de nos années d'études.

# *Dédicace*

***Toutes les lettres ne sauraient trouver les mots qu'il faut ...***

***Tous les mots ne sauraient exprimer la gratitude, l'amour, le***

***respect, la reconnaissance ...***

***Aussi, c'est tout simplement que...***

***Je dédie ce mémoire à***

***A l'esprit de mes chère parents,***

***Je dédie ce modeste travail et ma profonde***

***gratitude à Ma très chère grande sœur kira pour son amour et son soutien***

***A Mes très chère Frère sami et samir et ma chère Sœurs hadda et messaouda,***

***A mon très chère niece hidaya (houdhoud)***

***et chère neveu Mohammed khatib et mouatez***

***et toute Ma chère Famille.***

***A Mes Amis et Mes Collègues de Travail***

***surtout « Razika,dounia,Naiim,Nassir»,***

***Un Spécial remerciement à «M. Sriti Ali » pour son soutien et ses***

***encouragements***.

***et à Tous ceux et celles qui me connaissent.***

***Je Vous Aime.***

***Mlle. Boudjemaa Samira***

## 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.

Le but de ce travail est d'expliquer l'architecture des différentes bases NoSQL disponibles sur le marché. Et de voir d'une façon plus approfondi de trois d'entre elles, à savoir: Redis, MongoDB, Cassandra 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 (YCSB), c'est la meilleure base de données destine aux bases Nosql. On a fait la comparaison dans deux machines de processeur i3 et 64 bits et pour deux systèmes différents : la dernière version de ubuntu18.04 et windows10.

**Mots-clés :** SGBDR**,** NoSQL, Redis, MongoDB, Cassandra, 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.

The purpose of this work is to explain the architecture of the different NoSQL databases available on the market and see a greater depth three of them, namely: Redis, MongoDB, Cassandra 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 (YCSB), this is the best database for Nosql databases. We made the comparison in two i3 and 64-bit processor machines and with two systems: the latest version of ubuntu18.04 and windows10

**Keywords:** DBMS**,** NoSQL, MongoDB, Redis, Cassandra,YCSB

# Table de matières

# Liste des figures

**Chapitre I :**



**Chapitre II :**



**Chapitre III :**



**Chapitre IV :**

# Liste des Tableaux

# 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 trois 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

## Les Bases De Données Relationnelles Et Leurs Limites

# 1. Introduction

Les bases de données ont pris aujourd'hui une place essentielle dans l'informatique, plus particulièrement en gestion. Au cours des trente dernières années, des concepts, méthodes et algorithmes ont été développés pour gérer des données sur mémoires secondaires; ils constituent aujourd'hui l'essentiel de la discipline « Bases de Données » (BD). Cette discipline est utilisée dans de nombreuses applications.

Il existe un grand nombre de Systèmes de Gestion de Bases de Données (SGBD) qui permettent de gérer efficacement de grandes bases de données. De plus, une théorie fondamentale sur les techniques de modélisation des données et les algorithmes de traitement a vu le jour. Vous avez sans doute une idée intuitive des bases de données. Prenez garde cependant, car ce mot est souvent utilisé pour désigner n'importe quel ensemble de données.

Une base de données est un ensemble de données modélisant les objets d'une partie du monde réel et servant de support à une application informatique. Pour mériter le terme de base de données, un ensemble de données non indépendantes doit être interrogeable par le contenu, c'est-à-dire que l'on doit pouvoir retrouver tous les objets qui satisfont à un certain critère.

# 2. Les bases de données « BDD »

## 2.1. Définition d'une BDD

Une base de données peut être vue comme une collection de données persistantes, opérationnelles, enregistrées en mémoire secondaire (disque dure par exemple). Ces données doivent êtres cohérentes, non redondantes (ou de redondance minimale). Accessibles simultanément par plusieurs programmes (ou utilisateurs). Il faut noter qu'une base de données est indépendante des programmes d'application qui l'utilisent.

En d'autres termes, une base de données est un ensemble de données organisé en vue de son utilisation par des programmes correspondant à des applications distinctes et de manière à faciliter l'évolution indépendante des données et des programmes. **[Site 1]**

## 2.2. L'utilité d'une BDD

L'utilité d'une base de données est de regrouper les données communes à une application dont le but est :

- D'éviter les redondances et les incohérences de données qu'entérinerait fatalement une approche ou les données sont différents fichiers sans connexions entre eux.
- D'offrir des langages de haut niveau pour la définition et la manipulation des données.
- De partager les données entre plusieurs utilisateurs.
- De contrôler l'intégrité, la sécurité et la confidentialité des données.

- D'assurer l'indépendance entre les données et les traitements.

## 2.3. La conformité« ACID» [Site 2]

**ACID** est un acronyme résumant les quatre propriétés élémentaires d'une transaction au sein d'une base de données : **l'Atomicité, la Cohérence**, **l'Isolation** et **la Durabilité**, Andreas Reuter et Theo Härder ont inventé l'acronyme " **ACID** " en 1983.

Au sein d'une base de données, le terme de " transaction " désigne les opérations apportant des modifications aux données. Par exemple, un virement bancaire provoquant le débit du compte de l'émetteur et le crédit du compte du bénéficiaire est une transaction. Ces transactions doivent toutefois présenter quatre propriétés visant à garantir leur validité même en cas d'erreur ou de pannes informatiques.

Ces quatre propriétés sont :

- **Atomicité :** L'atomicité des transactions au sein des bases de données signifie que tous les changements apportés aux données sont effectués totalement, ou pas du tout. Soit tous les changements apportés par la transaction sont enregistrés pour la postérité, soit aucun ne l'est. En outre, l'atomicité permet d'éviter que les changements prennent effet en cas de panne de l'application ou du serveur de la base de données en plein milieu de la transaction. Ainsi, la base de données ne risque pas d'être corrompue par des opérations imprévisibles et à moitié complétées.
- **Cohérence:** La cohérence signifie que les transactions doivent respecter les contraintes d'intégrité des données de la base de données. Ainsi, une transaction qui commence avec un ensemble de données cohérent (respectant les contraintes d'intégrité) doit résulter par un ensemble de données cohérent. Pour maintenir la cohérence, un système de base de données DBMS peut abandonner les transactions qui risquent de provoquer une incohérence. Précisons que cette cohérence ne s'applique pas aux étapes intermédiaires de la transaction. Pour assurer la cohérence au fil du temps, il est possible d'utiliser une base de données temporelle. Ceci permet de spécifier les contraintes de données qui doivent traverser la dimension temporelle.
- **Isolation :** L'isolation signifie que les écritures et lectures des transactions réussies ne seront pas affectées par les écritures et lectures d'autres transactions, qu'elles soient ou non réussies. Les transactions isolées peuvent être " sérialisées ", ce qui signifie que l'état final du système peut être atteint en effectuant les transactions une par une.

  Pour décrire comment l'isolation peut être atteinte, on emploie souvent les termes des transactions " optimistes " ou " pessimistes ". Dans le cas des transactions optimistes, le logiciel de gestion des transactions considère de manière générale que les autres

transactions sont réussies et qu'elles ne lisent ou n'écrivent pas deux fois au même endroit. Il permet aussi les lectures et les écritures aux données modifiées par d'autres transactions. Si une transaction est avortée ou si des conflits d'ordre surviennent, les deux transactions sont annulées et réinitialisées.

Dans le cas des transactions pessimistes, les ressources sont verrouillées jusqu'à la fin de la transaction pour empêcher toute interférence. Si une transaction nécessite des ressources utilisées par une autre, elle devra patienter. Bien souvent, cette attente est inutile et entame les performances.

Il existe en outre de nombreuses approches hybrides combinant transactions optimistes et pessimistes. Ceci permet généralement de profiter des performances supérieures des transactions optimistes, et de la progression garantie des transactions pessimistes.

- **Durabilité :** Enfin, la durabilité garantit que les transactions réussies survivront de façon permanente et ne seront pas affectées par d'éventuelles pannes ou problèmes techniques. Les changements apportés aux données doivent être permanents. Plus précisément, ce sont les effets logiques des données modifiées sur les futures transactions qui doivent être permanents. La simple écriture des données sur le disque ne suffit pas pour atteindre la durabilité. Pour cause, le disque peut par exemple tomber en panne. Il est nécessaire que le DBMS écrive des logs sur les changements effectués. Ces logs doivent être permanents, et éventuellement redondants.

  En cas de panne du disque, de l'OS ou du DBMS, la base de données sera redémarrée et le DBMS pourra vérifier si les données sont synchronisées avec les logs. En effet, les logs contiennent les effets de toutes les transactions effectuées. Ce n'est pas forcément le cas des fichiers de données. Si des informations sont manquantes, les opérations enregistrées dans les logs seront à nouveau effectuées.

## 2.4. Les modèles de BDD

- **Le modèle hiérarchique (années 60) :** Dans premier modèle de SGBD, les données sont classées hiérarchiquement. Ce modèle utilise des pointeurs entre les différents enregistrements, organisés dans une structure arborescente de façon à ce que chaque enregistrement n'ait qu'un seul possesseur.
- **Le modèle réseau (années 70) :** Il lève de nombreuses limites du modèle hiérarchique grâce à la possibilité d'établir des liaisons de type n-n, les liens entre objets pouvant exister sans restriction. Pour retrouver une donnée dans ce modèle, il faut connaitre le chemin d'accès (les liens), ce qui rend les programmes dépendants de la structure de données. **[BEN15]**

- **Le modèle relationnel :** 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 rapidement un intérêt considérable en raison de sa simplicité et de ses fondations mathématiques. Une base de données relationnelle est un répertoire d'éléments de données dotés d'une relation prédéfinie entre eux. Ces éléments sont organisés en des tableaux définis, composés de colonnes et de rangées. Les tableaux permettent de répertorier les informations sur les objets qui doivent être représentés dans la base de données. Chaque colonne d'un tableau répertorie un certain type de données et chaque champ indique la valeur réelle d'un attribut. Les rangées d'un tableau représentent un ensemble de valeurs liées à un objet ou à une entité. Chaque rangée d'un tableau peut être marquée avec un identifiant unique, qu'on appelle une clé principale, et les rangées reprises dans plusieurs tableaux peuvent être associées en utilisant des clés étrangères. Ces données sont accessibles de plusieurs manières sans qu'il ne soit nécessaire de réorganiser les tableaux de base de données eux-mêmes. d'autre part ces données sont gérées par un système de gestion de bases de données relationnelle (Relational Data Base Management System, RDBMS), 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. 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 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. **[KEM15]**

  Propose une démarche cohérente et unifiée pour :

  - **la description des données** (LDD : Langage de Description des Données) ;
  - **l'interrogation des données** (LMD : Langage de Manipulation des Données).
- **Le modèle Orienté Objet :** Il permet de voir une base de données comme un ensemble de classe d'objets, ayant des liens d'héritage, d'agrégation, de composition, ou de simple association entre elle.

## 3. Le système de gestion de base de données (SGBD)

Un système de gestion de base de données est un ensemble de programmes qui permet la gestion et l'accès à une base de données. Un SGBD héberge généralement plusieurs bases

de données, qui sont destinées à des logiciels ou des thématiques différentes. Nous distinguons couramment les SGBD classiques, dits SGBD-R ou SGBD relationnels des SGBD-O ou SGBD orientés objet.

## 3.1. Les objectifs d'SGBD

Les bases de données et les systèmes de gestion de base de données ont été Crès pour répondre à un certain nombre de besoins et pour résoudre un certain nombre de problèmes. Ces objectifs sont les suivants : **[Site 3]**

- **Indépendance physique :** La façon dont les données sont définies doit être indépendante des structures de stockage utilisées.
- **Indépendance logique :** Un même ensemble de données peut être vu différemment par des utilisateurs différents. Toutes ces visions personnelles des données doivent être intégrées dans une vision globale.
- **Accès aux données :** L'accès aux données se fait par l'intermédiaire d'un Langage de Manipulation de Données (LMD). Il est crucial que ce langage permette d'obtenir des réponses aux requêtes en un temps « raisonnable ». Le LMD doit donc être optimisé, minimiser le nombre d'accès disques, et tout cela de façon totalement transparente pour l'utilisateur.
- **Administration centralisée des données (intégration) :** Toutes les données doivent être centralisées dans un réservoir unique commun à toutes les applications. En effet, des visions différentes des données (entre autres) se résolvent plus facilement si les données sont administrées de façon centralisée.
- **Non-redondance des données :** Afin d'éviter les problèmes lors des mises à jour, chaque donnée ne doit être présente qu'une seule fois dans la base.
- **Cohérence des données :** Les données sont soumises à un certain nombre de contraintes d'intégrité qui définissent un état cohérent de la base. Elles doivent pouvoir être exprimées simplement et vérifiées automatiquement à chaque insertion, modification ou suppression des données. Les contraintes d'intégrité sont décrites dans le Langage de Description de Données (LDD).

I. **Partage des données :** Il s'agit de permettre à plusieurs utilisateurs d'accéder aux mêmes données au même moment de manière transparente. Si ce problème est simple à résoudre quand il s'agit uniquement d'interrogations, cela ne l'est plus quand il s'agit de modifications dans un contexte multiutilisateur, car il faut : permettre à deux (ou plus) utilisateurs de modifier la même donnée « en même temps » et assurer un

résultat d'interrogation cohérent pour un utilisateur consultant une table pendant qu'un autre la modifie.

- **Sécurité des données :** Les données doivent pouvoir être protégées contre les accès non autorisés. Pour cela, il faut pouvoir associer à chaque utilisateur des droits d'accès aux données.
- **Résistance aux pannes :** Que se passe-t-il si une panne survient au milieu d'une modification, si certains fichiers contenant les données deviennent illisibles ? Il faut pouvoir récupérer une base dans un état « sain ». Ainsi, après une panne intervenant au milieu d'une modification deux solutions sont possibles : soit récupérer les données dans l'état dans lequel elles étaient avant la modification, soit terminé l'opération interrompue.

## 3.2. Principe de fonctionnement de SGBD

Le SGBD est un ensemble de logiciels informatiques qui sert à manipuler les bases de données, à effectuer des opérations ordinaires telles que consulter, modifier, construire, transformer, copier, sauvegarder ou restaurer des bases de données. Le SGBD peut se décomposer en trois sous-systèmes. **[BISA15]**, ou trois niveaux de description des données ont été définis par la norme ANSI/SPARC.

### 3.2.1. Le niveau externe

Correspond à la perception de tout ou partie de la base par un groupe donné d'utilisateurs, indépendamment des autres. On appelle cette description le *schéma externe* ou *vue*. Il peut exister plusieurs schémas externes représentant différentes vues sur la base de données avec des possibilités de recouvrement. Le niveau externe assure l'analyse et l'interprétation des requêtes en primitives de plus bas niveau et se charge également de convertir éventuellement les données brutes, issues de la réponse à la requête, dans un format souhaité par l'utilisateur.

### 3.2.2. Le niveau conceptuel

Décrit la structure de toutes les données de la base, leurs propriétés (*i.e.* les relations qui existent entre elles : leur sémantique inhérente), sans se soucier de l'implémentation physique ni de la façon dont chaque groupe de travail voudra s'en servir. Dans le cas des SGBD relationnels, il s'agit d'une vision tabulaire où la sémantique de l'information est exprimée en utilisant les concepts de relation, attributs et de contraintes d'intégrité. On appelle cette description le schéma conceptuel.

### 3.2.3. Le niveau interne ou physique

S'appuie sur un système de gestion de fichiers pour définir la politique de stockage ainsi que le placement des données. Le niveau physique est donc responsable du choix de l'organisation physique des fichiers ainsi que de l'utilisation de telle ou telle méthode d'accès en fonction de la requête. On appelle cette description le schéma interne. **[Site 4]**

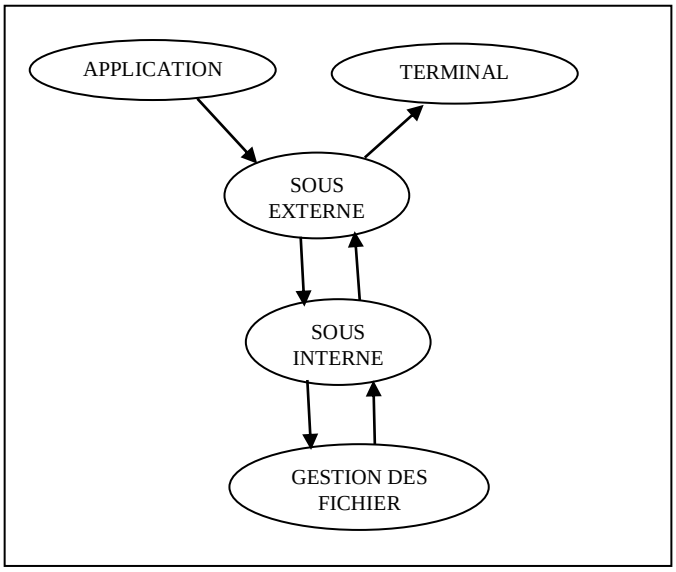



**Figure I.1 : le système de gestion de fichier permet de stocker les informations sur un support physique**

## 3.3. Les langages d'un SGBD

Dans un SGBD les données existent en permanence et doivent être déclarées une fois pour toute contrairement aux variables des programmes classiques qui disparaissent à l'arrêt du programme.

### 3.3.1. Le langage de description des données (LDD)

- spécifie le schéma conceptuel
- descriptif des types d'entités, entités, de leurs attributs et domaines des associations (ou relations) entre ces entités
- utilisé

  - lors de la définition de la on de la BD
  - lors des modifications du schéma conceptuel
  - pour préciser la façon dont les données sont enregistrées et comment y accéder (Correspondance entre schéma conceptuel et schéma physique).

### 3.3.2. Le langage de manipulation des données (LMD)

(LMD) ou langage d'interrogation

- interroger la base
- mettre à jour les données

Généralement un programme d'application (ex : gérer les réservations) est écrit dans un langage de programmation classique, appelé « langage hôte » (C, C++, Cobol, J , Cobol, Java, ...) et la communication avec la BD s'effectue e avec des instructions du LMD activées à partir du langage hôte.

On conserve ainsi la puissance des langages de programmation pour l'application tout en Profitant de la généralité et de la portabilité du LMD**. [Site 1]**

## 3.4. SGBD Relationnel

Puisqu'un langage de requête structuré est le noyau du système, ce type de base de données est également appelé SQL (Structured Query Language) est l'interface principale qui permet de communiquer avec les bases de données relationnelles. SQL est devenue une norme de l'American National Standards Institute (ANSI) en 1986. La norme ANSI SQL est prise en charge par tous les moteurs de bases de données relationnelles courants, certains d'eux disposant même d'une extension à ANSI SQL pour prendre en charge des fonctionnalités qui leur sont spécifiques. SQL permet d'ajouter, de mettre à jour ou de supprimer des rangées de données, de récupérer des sous-ensembles de données pour le traitement des transactions et les applications analytiques et de gérer tous les aspects de la base de données.

Dans les SGBD relationnels, les données apparaissent sous forme de tableaux de lignes et de colonnes avec une structure rigide, avec des dépendances claires. En raison de la structure intégrée et du système de stockage des données, les bases de données SQL ne nécessitent pas beaucoup de travail additionnel pour être bien protégées. Ils constituent un bon choix pour la création et la prise en charge de solutions logicielles complexes, où toute interaction a de nombreuses conséquences. L'un des principes fondamentaux de SQL est la conformité ACID (**A**tomicity, **C**onsistency, **I**solation, **D**urability). La conformité ACID est une option intéressante si vous créez, par exemple, des applications e-commerce ou dans le secteur financier, dans lesquelles l'intégrité de la base de données est essentielle.

Cependant, la croissance peut être un défi avec les bases de données SQL. La mise à l'échelle d'une base de données SQL entre plusieurs serveurs (mise à l'échelle horizontale) nécessite des efforts d'ingénierie supplémentaires. Au lieu de cela, les bases de données SQL

sont généralement mises à l'échelle verticalement, c'est-à-dire en ajoutant plus de puissance de calcul à un serveur.

Il existe de nombreux systèmes de gestion de bases de données, en voici une liste non exhaustive de Quelques SGBD relationnels connus et utilisées : **[Site 2]**

- MySQL
- MariaDB
- Oracle
- PostgreSQL
- MSSQL
- dBase (Borland) : gestionnaire de fichier structurés + langage de programmation
- FoxPro
- Access (Microsoft)
- Paradox (Borland)
- Ingres
- Informix
- Sybase
- DB2 (IBM)

## 3.5. Limites de SGBD Relationnels

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

### 3.5.1. Modélisation des entités réelles

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

### 3.5.2. Surcharge sémantique

Le modèle relationnel s'appuie sur un seul concept (la relation) pour modéliser à la fois les entités et les associations / relations entre ces entités. Il existe donc un décalage entre la réalité et sa représentation abstraite. **[AMW15]**

### 3.5.3. Contraintes d'intégrité

Les SGBDs relationnels sont limités à des contrôles de cohérence simples. Il est donc difficile de modéliser des contraintes réelles correspondant aux données d'une entreprise. Ce

problème n'est que partiellement résolu avec SQL-2 qui permet d'exprimer des contraintes dans la partie langage de définition. **[AMW15]**

### 3.5.4. Langage de manipulation

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

### 3.5.5. Scalabilité limitée

Atteinte par les groupes comme Yahoo, Google, Face book...etc. Cette limite est la plus gênante pour les entreprises. Dans le cas de traitements de données en masse, le constat est simple : les SGBD relationnels ne sont pas adaptés aux environnements distribués requis par les volumes gigantesques de données et par les trafics tout aussi gigantesques générés par ces opérateurs. **[AMW15]**

### 3.5.6. Des propriétés ACID en milieu distribué

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

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

- Le coût en stockage est énorme car chaque donnée est présente sur chaque serveur.
- Le coût d'insertion/modification/suppression est très grand**. [AMW15]**

### 3.5.7. La limite face aux usages

Les structures de données manipulées par les systèmes sont devenues de plus en plus complexes avec en contrepartie des moteurs de stockage peu évoluant. Le principal point faible des modèles relationnels est l'absence de gestion d'objets hétérogènes ainsi que le besoin de déclarer au préalable l'ensemble des champs représentant un objet. **[AMW15]**

## 4. Conclusion

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

# Chapitre II

## Les Bases De Données Nosql

# 1. Introduction

Les bases de données NoSQL, signifiant « Not only SQL » ont commencé à émerger depuis 2009, pour répondre aux nouveaux besoins en performance lors de traitement de gros volumes de données. Le NoSQL ne vient pas remplacer les BD relationnelles mais proposer une alternative ou compléter les fonctionnalités des SGBDs relationnels pour donner des solutions plus intéressantes dans certains contextes.

# 2. Big Data [MH18]

## 2.1. Définition

Littéralement, Big Data signifie données massives ou méga données. C'est un ensemble d'entités de données hétérogènes en extensibilité permanente qui ne peuvent pas pris en charge par les systèmes de gestion de données classiques. Big Data est aussi une architecture distribuée et scalable pour le traitement et le stockage de grands volumes de données. En effet, on crée environ 2,5 milliards de Giga octets de données tous les jours, émanant des différents domaines créés par les divers outils numériques : vidéos publiés, messages envoyés, signaux GPS, enregistrements transactionnels d'achats en ligne et bien d'autres encore.

Ces volumes massifs de données sont baptisés Big Data. Les géants du Web, au premier rang comme Yahoo, Facebook, Amazon et Google, ont été les tous premiers à déployer ce type de technologie pour permettre à tout le monde d'accéder en temps réel à leurs bases de données géantes.

L'émergence du Big Data est considérée comme une nouvelle révolution industrielle semblable à la découverte de la vapeur, de l'électricité, du téléphone et de l'informatique. D'autres, qualifient ce phénomène comme étant le dernier épisode de la troisième révolution industrielle, dite celle de « l'information ».

Cependant, aucune définition universelle ou précise ne peut qualifier le Big Data. Etant un concept polymorphe et complexe, son interprétation varie selon les communautés qui s'y intéressent en tant que fournisseur ou utilisateur de services. Le Big Data est aussi défini par rapport à la manière avec laquelle les grandes masses de données peuvent être traitées et exploitées de façon optimale.

Une autre définition similaire dans ou les auteurs qualifient les Big Data comme n'importe quel type de source de données qui a au moins trois caractéristiques communes :

- Volumes de données extrêmement volumineux ;
- Vitesse de transmission extrêmement élevée ;

- Très grande Variété de données.

La vérité est que le Big Data a bien trois significations et que les éditeurs n'en abordent qu'une seule à la fois. Il est important de connaître leur positionnement pour comprendre le principe. Un quatrième V pour Valeur et un cinquième pour la Véracité sont apparu ultérieurement.

### 2.1.1. Volume

Le Big Data est associé à un volume de données vertigineux, se situant actuellement entre quelques dizaines de téraoctets (1 To=212 octets) et plusieurs pétaoctets (1 Po=215 octets) en un seul jeu de données. Le volume correspond à la masse d'informations produite chaque seconde. Les entreprises issues de tous les secteurs d'activité gérant des données massives, se voient assujetties à trouver des techniques nécessaires et moyens capables pour gérer les volumes de données collectés chaque jour et d'une importance vitale pour leur survie.

### 2.1.2. Vélocité

La vélocité ou la vitesse d'échanges décrit la fréquence à laquelle les informations sont générées, capturées, stockées et partagées. Elle est aussi le traitement des flux continus de données. Les entreprises doivent appréhender la vitesse non seulement en termes de création de données, mais aussi sur le plan de leur traitement, de leur analyse et de leur restitution à l'utilisateur en respectant les exigences des applications en temps réel.

### 2.1.3. Variété

De plus en plus, le taux des données structurées manipulées dans des tables de bases de données relationnelles est en décroissance par rapport à l'expansion des types de données non-structurées. Cela peut être des images, des vidéos, des messages, des voix, et bien d'autres encore. Aujourd'hui, on trouve plusieurs milliers de sources hétérogènes comme les capteurs d'informations aussi bien dans les trains, les automobiles, les avions ou les équipements électroménagers qui émettent une variété d'informations de tout genre. Les technologies Big Data, permettent de faire de la création, l'intégration, l'analyse, la reconnaissance, le classement des données de différents types comme des photos sur différents sites ou les messages échangés sur les réseaux sociaux, etc. Ce sont les différents éléments qui constituent la variété supportée par le Big Data.

### 2.1.4. Valeur

La notion de Valeur correspond à l'intérêt qu'on puisse tirer de l'utilisation de cette technologie. Selon les experts du domaine, les entreprises qui ne s'intéressent pas sérieusement au contenu de leurs volumes de données hébergées risquent d'être pénalisées et

dépassées. Big Data désigne à la fois les grands volumes de données et la difficulté à extraire de cette masse de données celles ayant suffisamment de valeur pour justifier leur analyse. Big Data offre un ensemble d'outils d'analyse de données qui peuvent servir à préserver un privilège concurrentiel.

### 2.1.5. Véracité

L'aptitude à juger la crédibilité et la fiabilité du nombre indéfini de données collectées qualifie la Véracité du Big Data. Il est difficile de justifier l'authenticité et l'exactitude des contenus des différents volumes et variétés de données manipulées comme dans les conversations dans les réseaux sociaux avec les abréviations, le langage familier, les coquilles, les hashtags. La vérité est que le Big Data, a bien trois significations et que les éditeurs n'en abordent qu'une à la fois. Il est important de connaître leur positionnement pour leur poser les bonnes questions.

## 2.2. Architecture Big Data

On distingue principalement les couches suivantes: **[MH18]**

- **Couche matériel (infrastructure Layer):** Peut employer des serveurs virtuels VMware, ou des serveurs physiques ;
- **Couche stockage (Storage layer):** Les données seront stockées soit dans une base NoSQL, ou bien directement dans le système de fichier distribué ou les Datawarehouses;
- **Couche management et traitement (traitement layer):** On trouve dans cette couche les outils de traitement et analyse des données comme MapReduce ou Pig ;
- **Couche visualisation (visualisation layer):** pour la visualisation du résultat du traitement.

Une architecture Big Data doit inclure un ensemble de services qui permettent d'utiliser une multitude de sources de données de manière rapide et efficace. Parmi les composants de cette architecture, citons :

- **Infrastructure physique redondante :** Les Big Data n'auraient probablement pas émergé, sans la disponibilité d'infrastructures physiques robustes différentes des infrastructures traditionnelles. L'infrastructure physique est basée sur une architecture distribuée ou les données peuvent être stockées physiquement dans plusieurs sites connectés par le biais des réseaux, un système de fichiers distribué et divers outils et applications d'analyse de données. La redondance est importante pour traiter des données provenant de sources différentes, et pour assurer un service continu et réduire la latence.

- **Sécurité d'infrastructure :** Plus les entreprises s'intéressent à l'analyse de leurs données contenues dans les Big Data, plus il sera important de sécuriser ces données. Un système Big Data doit authentifier les utilisateurs et prendre en charge l'attribution de privilèges qui permet à ces utilisateurs de disposer légitimement d'un accès aux données. Ces types d'exigences de sécurité doivent faire partie de la conception préalable d'un système Big Data.

## 3. Le NoSQL

Dans n'importe quel débat ou travail sur les Big Data, on est amené à citer explicitement ou implicitement le NoSQL. Ces modèles marquent une rupture assez brutale, avec la manière de conception des schémas de bases de données et manipulation de ces données, que l'on pratiquait depuis déjà quelques décennies puisqu'elles abandonnent complètement ou partiellement la représentation matricielle de l'information et le langage SQL. Cette mouvance est venue pour solutionner les difficultés rencontrées dans la gestion des données classées « Big Data ». Spécifiquement dédiées aux applications orientées Internet, les bases de données NoSQL sont conçues pour pallier aux difficultés rencontrées par les systèmes de gestion de bases de données relationnelles réparties sur un très grand nombre de machines**. [MH18]**

### 3.1. Définition et concepts de base

« NoSQL » est une combinaison de deux mots : No et SQL. Le nom peut sembler comme une opposition aux bases de données SQL, et pourrait être mal interprétée en pensant 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 » « Non seulement », c'est donc une manière de dire qu'il y a autre chose que les bases de données relationnelles et le SQL.

L'idée et la fonction initiale du mouvement n'étant pas de remplacer les systèmes relationnels dits SQL, mais de proposer des alternatives ou compléter les fonctionnalités de ces modèles pour manipuler des grandes masses de données distribués.

Le mouvement NoSQL regroupe de nombreuses solutions de gestion de données qui ne sont plus fondées sur l'architecture classique des bases relationnelles et se distinguent du modèle SQL par une logique de représentation de données non relationnelle. L'unité logique n'y est plus la table, et les données ne sont en général pas manipulées avec SQL. À l'origine servant à manipuler des bases de données géantes pour des sites web de très grande audience tels que Google, Amazon.com, Facebook ou eBay.

Contrairement aux SGBD traditionnels et les entrepôts de données, ces nouveaux modèles de stockage de données sont distribués sur de nombreux serveurs et conçus pour

s'adapter à des milliers ou des millions d'utilisateurs qui font des mises à jour ainsi que les lectures. Ils s'appuient sur les systèmes de fichiers parallèles pour améliorer les performances et remédier aux limitations des ressources en multipliant les équipements pour permettre la parallélisation de stockage et l'accès aux informations. **[MH18]**

## 3.2. Intérêts

Les besoins exigés et le contexte d'utilisation des données sont les éléments de base qui priment sur le choix de la solution convenable de la bonne technologie de gestion de données. Il existe cependant certains critères déterminants pour basculer vers le NoSQL :

### 3.2.1. Scalabilité horizontale au lieu de scalabilité verticale [MH18]

La « scalabilité » est le terme utilisé pour définir l'aptitude d'un système à maintenir un même niveau de performance et avec des temps de réponse plus ou moins constants, face à la montée en de charge de toute nature : volumétrie de données, nombre de clients, etc., par augmentation des ressources matérielles

Il y a deux manières de rendre un système extensible : La scalabilité horizontale (externe) et la scalabilité verticale (interne), comme il est montré dans la Figure II.1

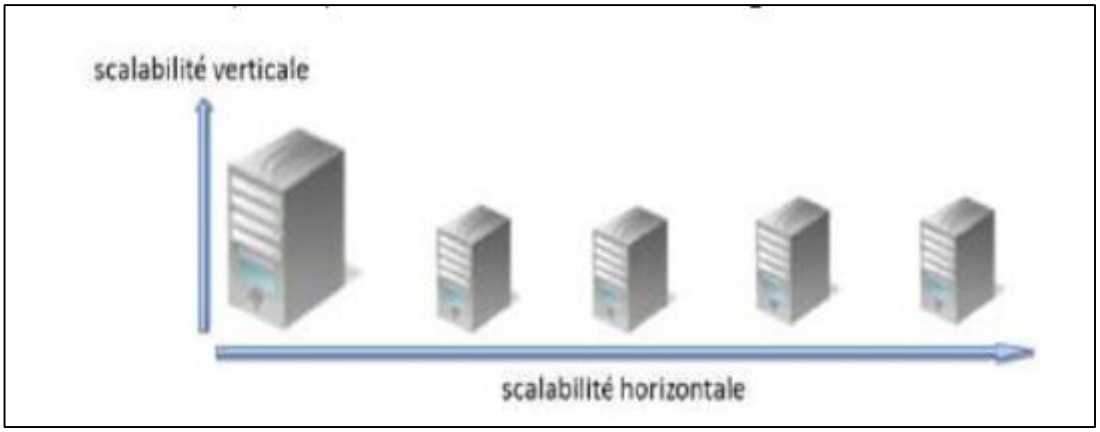



**Figure II.1 : Scalabilite horizontale et verticale**

- **Scalabilité horizontale (Scale-out) :**

  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.

Les avantages :

- Achat de serveurs quand le besoin s'en fait sentir.
- Une panne de serveur ne pénalise pas le système.

- Flexibilité du système.
- Possibilité de mises à jour sans interruption de service.

Les inconvénients :

- Achat de licences pour chaque serveur.
- Difficulté de trouver des serveurs identiques sur le long terme.

- **Scalabilité verticale (Scale-up) :**

  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. C'est ce que l'on appelle la croissance interne. On a une machine et on l'a fait évoluer dans le temps.

Les avantages :

- Achat d'une seule licence.
- Administration simplifiée du serveur.

Les inconvénients :

- Aucune tolérance à la panne.
- Achat d'une machine coûteuse
- Impossibilité de faire des mises à jour sans interrompre le système.

### 3.2.2. Gestion de gros volume de données

Les solutions NoSQL sont des bases de données fortement distribuées crées pour supporter un nombre important d'utilisateurs, d'opérations et de données réparties sur plusieurs machines. Bien que tous les systèmes NoSQL ne soient pas conçus pour répondre à cette problématique, l'aspect d'élasticité de ces bases de données permet la prise en charge de gros volumes de données (Big Data).

### 3.2.3. Performance en écriture

Avoir des meilleures performances en écriture est l'intérêt principal des géants Google, Facebook (135 milliards de messages par mois) et Twitter (7 TB de données par jour). La gestion des masses de données, augmentant considérablement chaque année, exige un temps très important pour le stockage de ces volumes dans les différents serveurs (à 80MB/s, on a besoin d'une journée pour stocker 7TB). En conséquence, l'écriture se doit être distribuée sur un cluster de machines, ce qui nécessite d'autres outils et techniques comme le MapReduce, la réplication, la tolérance aux pannes et un certain niveau de cohérence. **[Léo14]**

### 3.2.4. Types de données flexibles

Parmi les innovations majeures caractérisant les solutions NoSQL, est le fait que celles-ci supportent de nouveaux types de données. 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 oustructure, et changer de format en production. Les objets complexes peuvent être mappés sans trop de complications avec le modèle adopté **[Léo14]**

### 3.2.5. Structure dynamique

Une base de données NoSQL ne contient pas de schéma prédéfinie (schema-less ou schemafree) : Les schémas de la base ne sont pas figés, ce qui donne plus de flexibilité aux données ainsi la structure et le type de données peuvent changer à tout moment sans nécessairement affecter l'application, ce qui permet de lire et écrire les données plus rapidement ;

### 3.2.6. Economie

Les bases de données NoSQL ont tendance à utiliser des serveurs bas de gammes à moindre coûts afin d'équiper les « clusters ». Les serveurs destinés aux bases de données NoSQL sont généralement de bon marché et de moyenne qualité, contrairement à ceux qui sont utilisés par les bases de données relationnelles. De plus, la très grande majorité des solutions NoSQL sont Open–Source, ce qui reflète d'une part une économie importante sur le prix des licences.

## 3.3. Théorème de CAP [BZTH16]

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 conjecture qui définit qu'il est impossible, sur un système informatique de calcul ribué, de garantir en même temps les trois contraintes suivantes :

- **Cohérence (Consistency)** Tous les nœuds (serveurs) du système voient exactement les mêmes données au même moment.
- **Disponibilité (Availability)** Garantie que toute requête reçoive une réponse même si elle n'est pas actualisée.
- **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.

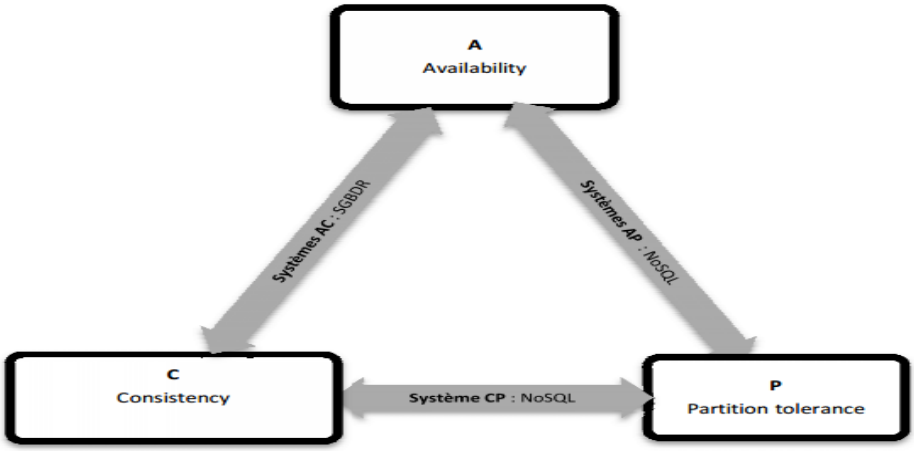



**Figure II.2: Théorème CAP**

Dans un système distribué, il est impossible d'obtenir ces 3 propriétés en même temps, il faut 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 (Disponible et Résistant au partitionnement)

**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.

**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.

**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.

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.

## 3.4. Caractéristiques [MH18]

Voici les caractéristiques principales d'une base de données NoSQL:

- Supporte le stockage en masse en multipliant le nombre d'enregistrements dans les tables (more than rows in tables) : Une table peut contenir des millions de lignes ;

- On n'est pas obligé d'appliquer des jointures entre les tables (free of joins) : Les données sont généralement intégrées dans le même « conteneur ». Cette forme de stockage peut être considérée comme des jointures déjà exécutées qui permet une lecture rapide de données ;
- Elle est conçue pour fonctionner sur plusieurs processeurs : Décomposez le problème en multiples threads avec de nombreuses CPU travaillant ensemble ;
- S'appuie sur la notion "ne rien partager" entre les nœuds (shared-nothing architecture) : Chaque nœud du cluster a sa propre CPU, RAM et disque ;
- Supporte l'élasticité linéaire (linear scalability) ou horizontale (scale-out) : les systèmes NoSQL sont conçus pour augmenter la charge de travail en distribuant automatiquement les données entre plusieurs nœuds et en formant un cluster pour maintenir une performance linéaire à haute évolutivité. Une base NoSQL peut se répartir sur plusieurs serveurs sans l'aide de l'application. De plus des serveurs peuvent être ajoutés ou retirés à la volée. Logiquement, un système NoSQL ne devrait jamais avoir à redémarrer ;
- Supporte les modèles de traitements parallèles, en particulier MapReduce, qui est souvent embarqué dans les solutions NoSQL, ce qui améliore et facilite les calculs parallèles dans des architectures distribuées ;
- Fournit des temps de réponse assez rapide en raison de la simplicité des schémas et l'absence de jointures.
- Gestion des données hétérogènes issues des différentes sources de données.
- Faibles coûts de gestion, d'exploitation et de stockage de gros volumes de données.

Il faut retenir aussi que le NoSQL n'est pas :

- Un langage opposé au SQL ;
- Toujours open source, il y a des produits NoSQL commerciaux ;
- Toujours lié au Big Data ou Cloud Computing : Il peut être utilisé dans d'autres contextes (beaucoup de systèmes relationnels non Big Data ont migré ou cherche à migrer vers le NoSQL).

En revanche, la plupart des solutions NoSQL, créées ces dernières années, présentent certaines faiblesses ou insuffisances comme :

- Non-existence de langages de requête normalisés tels que SQL ;
- Cohérence de données abandonnée relativement au profit de la haute disponibilité ;
- Sécurité des données et manque d'autres fonctionnalités supplémentaires insuffisamment développées : Les systèmes NoSQL se base sur l'application pour la protection des données.

- Maturité et stabilité : Les bases relationnelles ont une longueur d'avance sur ce point. Les utilisateurs sont familiers avec leur fonctionnement et ont confiance en elles.
- Manque d'architectures et d'interfaces normalisées ;
- Moins de documentation et d'outils disponibles.

## 3.5. Différents modèles NoSQL

Il existe une diversité de technologies et solutions toutes libellées en tant que NoSQL, se distinguant par leurs manières de représentation des données. Ces différents systèmes NoSQL utilisent des approches distinctes classées en quatre catégories:

Orientées clé-valeur ; Orientées colonnes ; Orientées documents et Orientées graphes.

### 3.4.1. Bases de données clé-valeur (Key-value store) [BABM16]

Les bases de données associatives ou clé-valeur sont les formes les plus simples de bases NoSQL, qui se présentent comme de grosses tables de hachage. Il s'agit d'un système, sous forme de tableau associatif, qui stocke des valeurs indexées par des clés uniques, dont le principe est très simple : chaque valeur, nombre ou texte est associé à une clé unique, qui sera le seul moyen d'y accéder.

Une entrée de tableau est un couple (Clé, Valeur) assez large en général Get(clé, valeur), Put(clé,valeur), Delete(clé). C'est uniquement par cette clé qu'il sera possible d'exécuter des requêtes sur la valeur. Ce modèle peut être comparable à un dictionnaire où chaque mot (clé) possède une ou plusieurs définitions (valeur).

Du moment où le dictionnaire est ordonné (indexé), il n'est pas nécessaire de parcourir tout le dictionnaire pour retrouver un mot (Figure II.3).

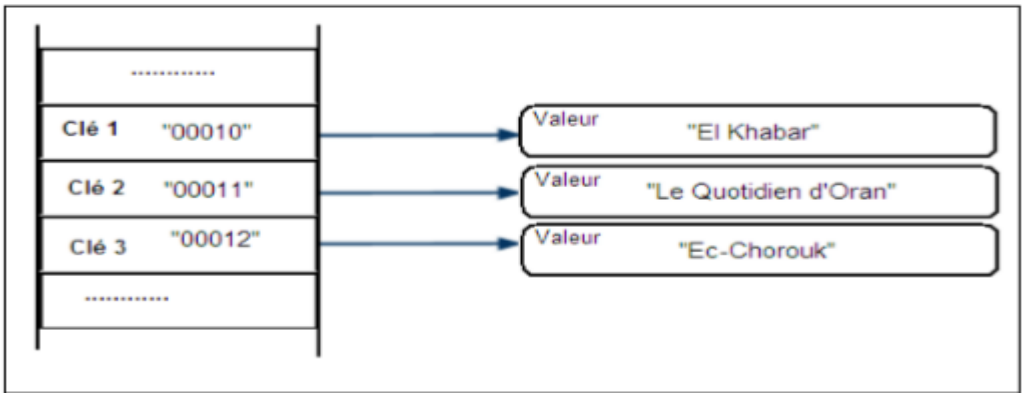



**Figure II.3 : Illustration d'une base de données clé-valeur**

De même, le type NoSQL clé-valeur est indexé par clés qui pointent directement à leurs valeurs correspondantes **[GKH et all13].** La structure de l'objet stocké est libre et donc à la charge du développeur de l'application. Les données, à la différence d'une table de

hachage, sont répliquées. Un avantage considérable de ce type est qu'il est très facilement extensible et donc facilite la scalabilité horizontale. Une simplicité du modèle est échangée par une bonne performance et surtout une scalabilité accrue. En effet dans le cas où le volume de données augmente, la charge est facilement répartissable entre les différents nœuds en redéfinissant tout simplement les intervalles des clés entre chaque nœud. Cette conception a permis de monter en charge à des péta-octets de données et des milliers de requêtes simultanées. Ces systèmes sont donc principalement utilisés comme dépôts de données se limitant à des requêtes simples exécutées uniquement sur les clés. L'interaction avec la base de données se résume aux opérations basiques CRUD (Create, Read, Update, Delete). Les solutions les plus connues ayant adoptées le système de couple clé-valeur sont : Redis, Memcached, Amazon DynamoDB, Riak, Voldemort Ehcache, Hazelcast (utilisé par le site LinkedIn), OrientDB, Berkeley DB, Oracle NoSQL, etc.

### 3.4.2. Bases de données orientées colonnes (Column family)

Les bases de données orientées colonnes sont grossièrement une évolution du modèle associatif. Elles constituent la poupe du mouvement NoSQL à cause de leur récurrence dans l'actualité avec des candidats comme HBase ou Cassandra **[Léo14]**. Elles ont été conçues par les géants du web afin de faire face à la gestion et au traitement de gros volumes de données s'amplifiant rapidement. De par leur structure, les bases orientées colonnes se rapprochent des bases relationnelles. Le principe d'une base de données colonnes consiste dans leur stockage par colonne et non par ligne. Contrairement d'une base de données orientée ligne classique ou les données sont stockées de façon à favoriser les lignes en regroupant toutes les colonnes d'une même ligne ensemble, les bases de données orientée colonnes quant à elles vont stocker les données de façon à ce que toutes 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. Cet ype de structure permet d'être plus évolutif et flexible ; cela vient du fait qu'on peut ajouter à tout moment une colonne. Autrement dit, et contrairement à une base de données relationnelle où les colonnes sont statiques, fixées dès la création du schéma de la table et ce nombre reste le même pour tous les enregistrements dans cette table, celles des bases orientées colonnes sont dites dynamiques et présentes donc uniquement en cas de nécessité, ce qui évite le stockage des valeurs Null. **[CaR10]**

Par exemple, l'enregistrement d'un étudiant nécessitant son nom, prénom et adresse. Dans le relationnel il faut au préalable créer le schéma (la table) qui permettra de stocker ces informations. Si un étudiant n'a pas d'adresse, la rigidité du modèle relationnel nécessite un marqueur Null, signifiant une absence de valeur qui coûtera toutefois de la place en mémoire.

Dans une base NoSQL orientée colonne, la colonne adresse n'existera tout simplement pas. Les bases de données colonnes ont été conçues pour pouvoir stocker plusieurs millions de colonnes (Figure II.4). Les principales solutions NoSQLorientées colonnes sont: Cassandra (Apache), HBase (Apache), Bigtable (Google), Accumulo (Apache), Hypertable, etc.

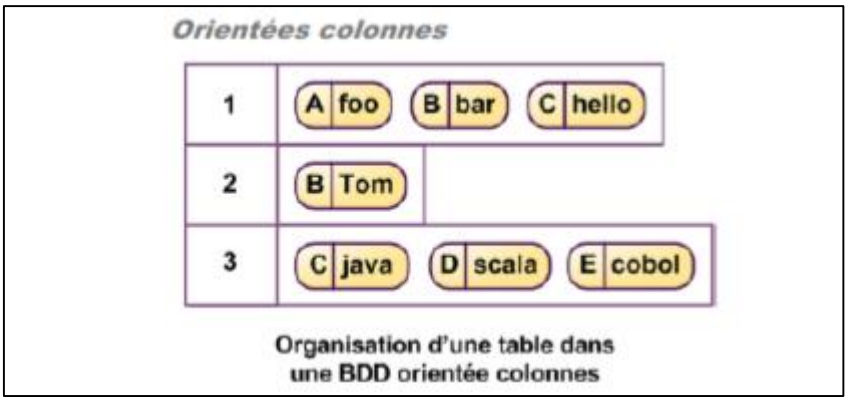



**Figure II.4: Base de données orientée colonnes**

## 3.4.3. Bases de données orientées documents (Document store)

Les bases de données orientées documents sont une alternative aux bases orientées colonnes. Elles fonctionnent donc sur le même principe associatif clé-valeur (ici clé-document), mais avec un ajout de fonctionnalités qui passe par la prise en compte de la structure des données stockées sous forme de document. Ces fonctionnalités comprennent :

- Ajout, modification, lecture ou suppression de seulement certains champs dans un document ;
- Indexation de champs de document permettant ainsi un accès rapide sans avoir recours uniquement à la clé ;
- Requêtes élaborées pouvant inclure des prédicats sur les champs.

Dans ces modèles, les clés ne sont plus associées à des valeurs sous forme de bloc binaire mais à un document de format non imposé.**[Han et all 11]** Les bases de données documentaires sont constituées de collections de documents (Figure II.5). Un document est composé d'un ensemble de champs et des valeurs associées qui peuvent être requêtées. Bien que les documents soient structurés, ce type de base est appelée « Schema-Less » puisqu'il n'est pas nécessaire de définir au préalable les champs d'un document, ce qui permet de stocker dans la base des documents très hétérogènes. Cependant bien que ces documents ne doivent pas se conformer à un modèle, cela ne veut pas pour autant qu'il n'y ait aucune contrainte sur les données. Le stockage structuré des documents leur confère des fonctionnalités dont ne disposent pas les bases clés-valeurs simples dont la plus évidente est la capacité à effectuer des requêtes sur le contenu des objets. La valeur, dans ce cas, est un

document de type JSON, BSON ou XML. Cette caractéristique les rapproche donc des bases SQL.

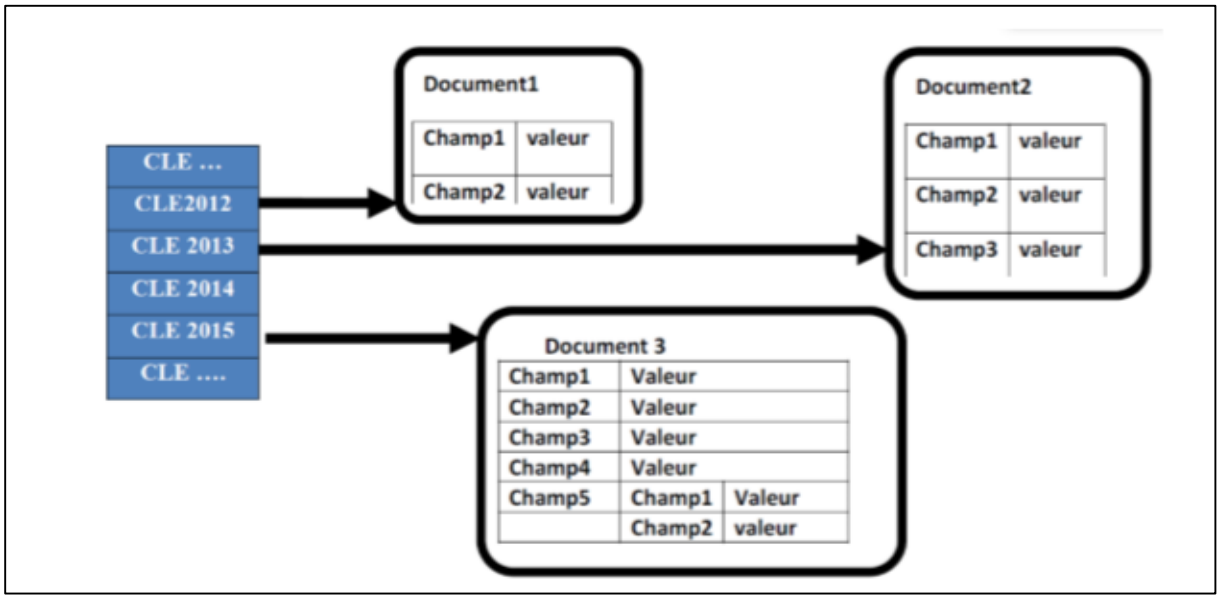



**Figure II.5: Base de données orientée documents**

L'avantage considérable d'une base de données orientée document est de pouvoir récupérer un ensemble d'informations structurées hiérarchiquement depuis une clé. Une opération similaire dans le monde relationnel équivaudrait à plusieurs jointures de tables **[PAG 13]**. Les principales bases de données NoSQL orientées documents sont : MongoDB (10gen), CouchDB, CoucHBase, Amazon DynamoDB, MarkLogic, RavenDB, Cloudant, OrientDB, GemFire, RethinkDB, Datameer, Microsoft Azure DocumentDB, ArangoDB, PouchDB, etc. **[MH18]**

### 3.4.4. Bases de données orientées graphes (Graph store)

Les bases de données orientées graphes sont des systèmes bien différents des autres bases de données. 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 NoSQL orientées graphe n'ont pas pour but de résoudre des problèmes de performances mais plutôt de palier à des problèmes très complexes de liens de connectivité entre les données, qu'une base de données relationnelle serait incapable de le faire.**[PAG 13]** Certains diront qu'il est possible d'exprimer de la complexité avec des bases relationnelles, et ils auront raison, à la différence près que les relations des graphes sont définies au niveau des enregistrements de la base et non au niveau du modèle de données. Le stockage de relations au niveau des données n'a de sens que si ces relations sont toutes très différentes, sinon cela revient à dupliquer toujours la même chose. **[Léo14]**. Les réseaux sociaux comme Facebook et Twitter, où des millions d'utilisateurs sont reliés de différentes manières, constituent un

bon exemple : amis, fans, famille, etc. Dans ce contexte, le défi n'est pas le nombre d'éléments 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 des applications comme les réseaux sociaux. **[DIM 12]** Elles permettent également l'élaboration de liens entre les divers intérêts que pourraient avoir un internaute, afin de pouvoir lui proposer des produits susceptibles de l'intéresser. Ainsi, les pubs s'affichant sur Facebook sont très souvent en relation avec les recherches effectuées sur Google, et les propositions d'achats de sites de vente en ligne tels que Ebay et Amazon sont en relation avec des achats déjà effectués **[GKH et all13]**.

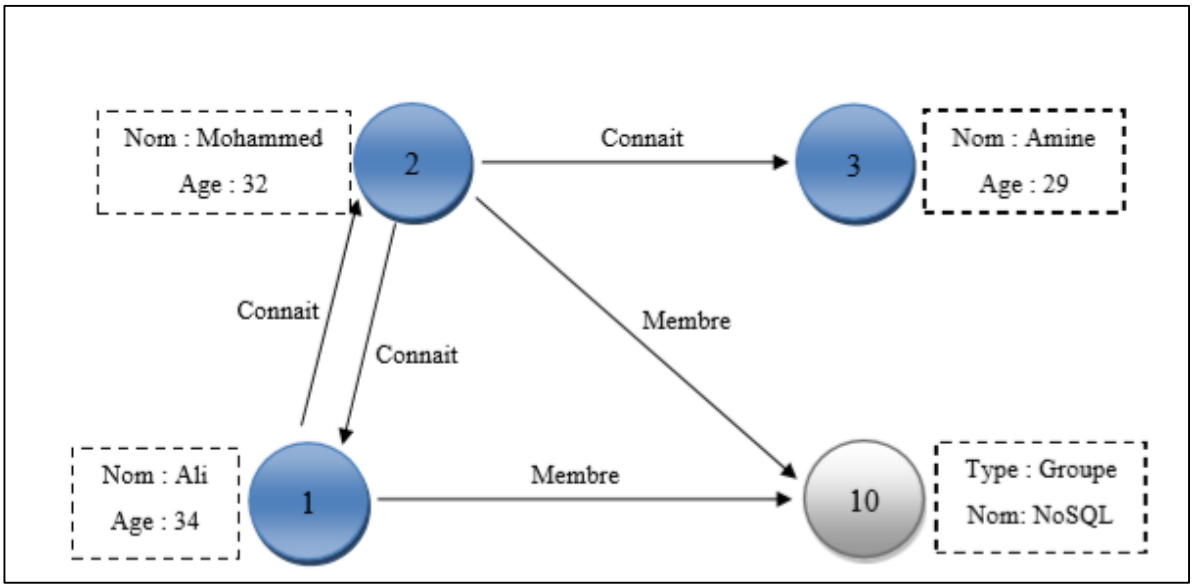



**Figure II.6: Exemple de base de données orientée Graphe**

Les données sont donc stockées sur chaque nœud, lui-même organisé par des relations. A partir de là, il sera nettement plus aisé d'effectuer des opérations qui auraient été très complexes et lourdes dans un univers relationnel. Le point positif de ce type de bases est qu'elles sont parfaitement adaptées à la gestion des données relationnelles même dans un contexte Big Data. Les solutions les plus connues ayant adoptées le NoSQL orienté graphe sont : Neo4j, Orient DB, Titan, ArangoDB, Giraph, InfiniteGraph, Sqrrl, Sparksee, InfoGrid, HyperGraphDB, FlockDB, VelocityGraph, GlobalsDB, GraphDB, etc. **[MH18]**

## 4. Conclusion

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

# Chapitre III

## Redis, MongoDB Et Cassandra

## 1. Introduction

Nous nous intéresserons particulièrement dans ce chapitre aux technologies **Redis, MongoDB , Cassandra c**e sont des technologies récentes, encore relativement peu connues du grand public mais auxquelles nous associons déjà des grands noms parmi lesquels: Facebook, Yahoo, Twitter et autres. Nous présentons en fin de chapitre l'outil Yahoo ! Cloud Serving Benchmark (YCSB) qui est un benchmark et générateur de banque d'essais pour les bases de données NoSQL.

Ces nouveaux systèmes sont distingués à base de plusieurs critères : **la performance**, **le modèle de données utilisé**, **la cohérence**, **les méthodes de stockage**, **les garanties de durabilité**, **la disponibilité**, **le support de requête**, et d'autres dimensions. Ces systèmes sacrifient généralement quelques-unes de ces dimensions au profit d'autres.

Pour évaluer et comparer les solutions NoSQL disponibles, plusieurs benchmarks ont été conçus, le plus couramment utilisé est le YCSB de Yahoo. Plusieurs études comparatives dans le même contexte, utilisant YCSB, ont été développées récemment.

L'étude avec laquelle on va comparer nos résultats a été menée dans une seule machine

## 2. Classement de popularité des systèmes NoSQL

Des études très récentes montrent que le type NoSQL orientée documents, en tant que modèle, reste le plus utilisé, comme il est montré dans la figure III.1. **[Léo14]**

La cause est probablement liée au fait qu'il soit simple à aborder (type JSON). D'un autre coté les orientées colonnes sont au coude à coude avec les clé-valeurs. Les orientées graphes sont en dernière position puisqu'elles sont principalement réservées à des applications très spécifiques.

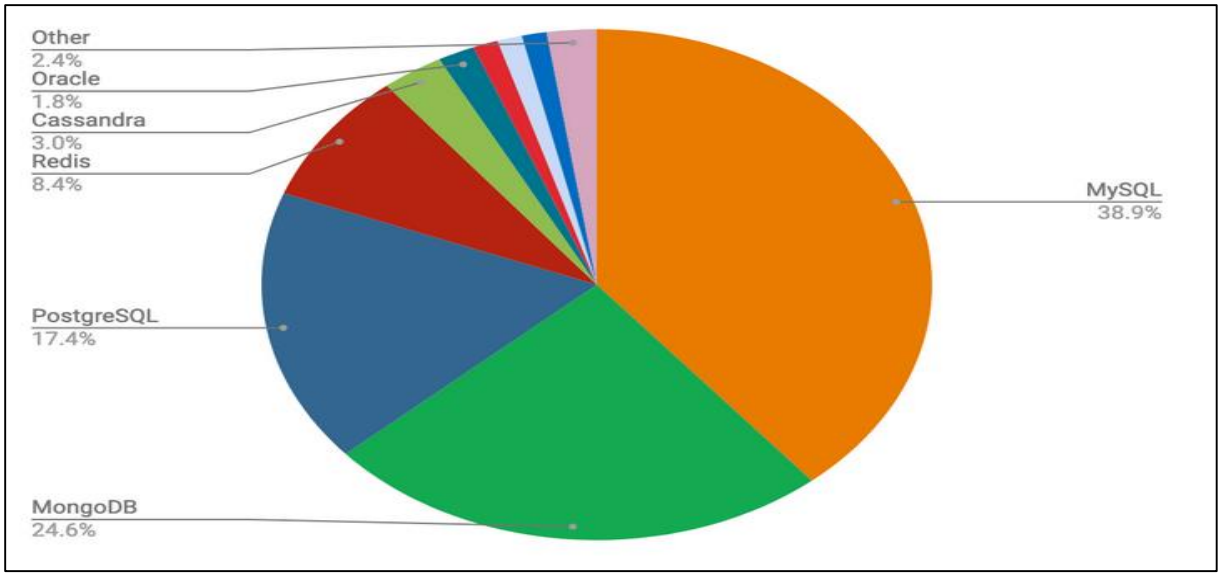



**Figure III.1 : Type de NoSQL, majoritairement utilisé**

Dans la section suivante, nous allons nous intéresser à la description proprement dite des bases NoSQL en ciblant des solutions très populaires, selon un classement de Solid IT **[Site 5].**

Ce DB-Engines Ranking est une liste actualisée de 354 systèmes de gestion de bases de données classés par degré de popularité. Ils mesurent la popularité d'un système en utilisant les paramètres suivants :

1. Nombre de citations du système dans les sites Web.
2. Intérêt général au système.
3. Fréquence des discussions techniques sur le système.
4. Nombre d'offres d'emploi dans lesquelles le système est mentionné.
5. Nombre de profils dans les réseaux professionnels, dans lesquels le système est mentionné.
6. Pertinence dans les réseaux sociaux.

Pour le mois de mars de l'année 2020, le classement des Top 50 est affiché dans le tableau III.1suivant : **[Site 5]**

354 systems in ranking, March 2020

| Rank Mar 2020 | Rank Feb 2020 | Rank Mar 2019 | DBMS | Database Model | Score Mar 2020 | Score Feb 2020 | Score Mar 2019 |
|---|---|---|---|---|---|---|---|
| 1. | 1. | 1. | Oracle | Relational, Multi-model | 1340.64 | -4.11 | +61.50 |
| 2. | 2. | 2. | MySQL | Relational, Multi-model | 1259.73 | -7.92 | +61.48 |
| 3. | 3. | 3. | Microsoft SQL Server | Relational, Multi-model | 1097.86 | +4.11 | +50.01 |
| 4. | 4. | 4. | PostgreSQL | Relational, Multi-model | 513.92 | +6.98 | +44.11 |
| 5. | 5. | 5. | MongoDB | Document, Multi-model | 437.61 | +4.28 | +36.27 |
| 6. | 6. | 6. | IBM Db2 | Relational, Multi-model | 162.56 | -2.99 | -14.64 |
| 7. | 7. | ↑ 9. | Elasticsearch | Search engine, Multi-model | 149.17 | -2.98 | +6.38 |
| 8. | 8. | 8. | Redis | Key-value, Multi-model | 147.58 | -3.84 | +1.46 |
| 9. | 9. | ↓ 7. | Microsoft Access | Relational | 125.14 | -2.92 | -21.07 |
| 10. | 10. | 10. | SQLite | Relational | 121.95 | -1.41 | -2.92 |
| 11. | 11. | 11. | Cassandra | Wide column | 120.95 | +0.60 | -1.84 |
| 12. | 12. | ↑ 13. | Splunk | Search engine | 88.52 | -0.26 | +5.42 |
| 13. | 13. | ↓ 12. | MariaDB | Relational, Multi-model | 88.35 | +1.01 | +4.04 |
| 14. | 14. | ↑ 15. | Hive | Relational | 85.38 | +1.85 | +12.38 |
| 15. | 15. | ↓ 14. | Teradata | Relational, Multi-model | 77.84 | +1.03 | +2.63 |
| 16. | 16. | ↑ 21. | Amazon DynamoDB | Multi-model | 62.51 | +0.38 | +8.02 |
| 17. | 17. | ↓ 16. | Solr | Search engine | 55.09 | -1.07 | -4.92 |
| 18. | 18. | ↑ 20. | SAP HANA | Relational, Multi-model | 54.27 | -0.70 | -1.24 |
| 19. | 19. | ↓ 18. | FileMaker | Relational | 54.16 | -0.72 | -3.97 |
| 20. | ↑ 21. | ↓ 19. | SAP Adaptive Server | Relational | 52.77 | +0.04 | -3.27 |
| 21. | ↑ 22. | ↑ 22. | Neo4j | Graph | 51.78 | +0.57 | +3.20 |
| 22. | ↓ 20. | ↓ 17. | HBase | Wide column | 51.15 | -1.80 | -7.64 |
| 23. | ↑ 25. | ↑ 25. | Microsoft Azure SQL Database | Relational, Multi-model | 35.44 | +4.04 | +7.51 |
| 24. | ↓ 23. | ↓ 23. | Couchbase | Document, Multi-model | 32.08 | -0.08 | -1.72 |
| 25. | ↓ 24. | ↑ 27. | Microsoft Azure Cosmos DB | Multi-model | 31.63 | -0.32 | +6.81 |
| 26. | 26. | ↑ 32. | Google BigQuery | Relational | 27.76 | +0.20 | +8.19 |
| 27. | 27. | ↓ 24. | Memcached | Key-value | 24.80 | -0.50 | -3.93 |
| 28. | 28. | ↓ 26. | Informix | Relational, Multi-model | 24.42 | -0.47 | -2.48 |
| 29. | ↑ 31. | ↑ 35. | InfluxDB | Time Series | 22.43 | +0.85 | +6.25 |
| 30. | ↓ 29. | ↓ 28. | Vertica | Relational, Multi-model | 22.20 | -0.43 | +0.13 |
| 31. | ↑ 32. | ↓ 29. | Amazon Redshift | Relational | 20.96 | -0.01 | +0.07 |
| 32. | ↓ 30. | ↓ 30. | Firebird | Relational | 20.58 | -1.24 | +0.41 |
| 33. | 33. | ↓ 31. | Netezza | Relational | 18.36 | -0.38 | -1.26 |
| 34. | 34. | ↓ 33. | CouchDB | Document | 18.14 | +0.00 | -0.50 |
| 35. | 35. | ↓ 34. | Spark SQL | Relational | 17.15 | +0.21 | +0.84 |
| 36. | 36. | 36. | Impala | Relational, Multi-model | 15.70 | -0.01 | +0.88 |
| 37. | 37. | ↑ 39. | dBASE | Relational | 14.11 | -0.10 | +2.61 |
| 38. | 38. | ↑ 41. | Firebase Realtime Database | Document | 12.60 | +0.25 | +2.30 |
| 39. | ↑ 40. | ↓ 38. | Greenplum | Relational, Multi-model | 12.00 | +0.07 | -0.35 |
| 40. | ↓ 39. | ↓ 37. | MarkLogic | Multi-model | 11.93 | -0.32 | -1.81 |
| 41. | 41. | ↓ 40. | Oracle Essbase | Relational | 10.47 | -0.17 | -0.32 |
| 42. | ↑ 43. | ↑ 71. | Presto | Relational | 9.93 | +0.40 | +6.15 |
| 43. | ↓ 42. | 43. | Microsoft Azure SQL Data Warehouse | Relational | 9.64 | +0.05 | +1.16 |
| 44. | 44. | ↑ 46. | Realm | Document | 9.47 | +0.62 | +2.50 |
| 45. | ↑ 46. | ↓ 44. | Hazelcast | Key-value, Multi-model | 9.07 | +0.79 | +1.05 |
| 46. | ↓ 45. | ↓ 42. | Datastax Enterprise | Wide column, Multi-model | 8.48 | 0.00 | -0.85 |
| 47. | 47. | ↑ 53. | Amazon Aurora | Relational, Multi-model | 8.15 | -0.03 | +2.34 |
| 48. | ↑ 51. | ↓ 47. | Aerospike | Key-value, Multi-model | 7.19 | +0.32 | +0.46 |
| 49. | 49. | | etcd | Key-value | 7.18 | -0.06 | |
| 50. | 50. | ↓ 48. | H2 | Relational | 7.12 | -0.01 | +0.55 |

**Tableau III.1 : Top 50 des SGBD**

Les 17 systèmes NoSQL qui se trouvent dans le Top 50, sont de différents modèles et sont employés dans divers projets de développement et exploitation.

Les trois solutions choisies, sur lesquelles notre étude va s'accentuer, sont rappelées dans le tableau suivant, selon le classement précédent :

| Système | Classement parmi les NoSQL | Classement général |
|---|---|---|
| MongoDB | 1 | 5 |
| Redis | 2 | 8 |
| Cassandra | 3 | 11 |

**Tableau III.2 : Classement des solutions étudiées**

## 2.1. Redis

### 2.1.1. Description

Redis est une base NoSQL de type clé/valeur, acronyme de **RE**mote **DI**ctionary**S**erver, écrit avec le langage de programmation C et distribué sous licence BSD. Il fait partie de la mouvance NoSQL ayant pour objectif fondamental d'avoir les performances les plus élevées possibles. Redis permet de manipuler des types de données simples : chaînes de caractères, tableaux associatifs, ensembles, ensembles ordonnés et listes.

Après avoir connu une grosse croissance en 2010, Redis continue tranquillement son chemin avec l'ajout du « sentinelfailover » qui s'occupe de monitorer, notifier et basculer automatiquement des instances en cas de problèmes.

L'intégration du langage Lua, car tout le monde utilise du Javascript, permettra d'étendre encore les possibilités de cette base orientée clé-valeur.

Actuellement elle est utilisée par des entreprises comme The Guardian, GitHub, StackOverflow, Craigslist ou encore Blizzard Entertainment**. [Hei12]**

### 2.1.2. Stockage en mémoire vive

Redis est une base de données clé/valeur fonctionnant intégralement en mémoire vive qui permet d'obtenir d'excellentes performances en évitant les accès disques, particulièrement coûteux. Redis peut également utiliser la mémoire virtuelle au cas où la taille des données est trop importante pour être tenu en mémoire.

Bien entendu une première difficulté se pose, c'est le risque de perte de données en cas d'incident. Pour y remédier,

- Redis offre la possibilité de sauvegarder l'état de la base dans un fichier. Cette technique ne permettant pas de garantir la conservation des opérations effectuées entre deux sauvegardes.
- La deuxième technique consiste à conserver une trace de toutes les opérations, et les restaurer en les ré-appliquant dans l'ordre, en cas d'incident.

Voici les deux fonctions employées par Redis **: [Hei12]**

− **RDB :** Consiste à copier toute la base de données se trouvant dans la RAM, dans un fichier sur le disque sous forme de backups complets périodiques. Cependant, cela n'est pas adapté pour enregistrer des transactions entre deux copies, sauf si on va procéder à un enregistrement de la base pour sauver la transaction désirée. **[Fur15]**

− **AOF :** fonctionne à la manière des "redo-log d'Oracle". C'est un journal dans lequel les transactions sont conservées dans des temps répétés et déterminés. Il permet à la base de garder une trace de toutes les manipulations de données, ainsi en cas de crash de la base, il permet de rejouer les modifications apportées à la base.

### 2.1.3. Architecture

Redis supporte la réplication via une architecture modèle maître-esclave à des fins de tolérance aux pannes et de répartitions de charges. Toutes les écritures doivent s'effectuer via l'instance maîtresse, mais des lectures sur les instances esclaves, peuvent être faites aussi.

#### 2.1.3.1. Maître / Esclave

Redis se base sur une architecture maître / esclave comme beaucoup de bases NoSQL.

Elle 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 clients.

Redis éprouve 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 (Figure III.2**). [Hei12]**

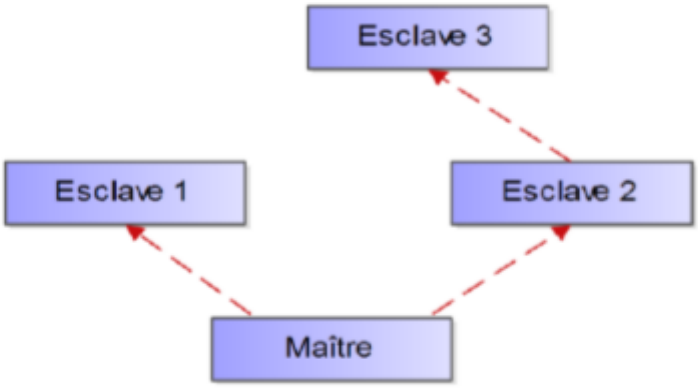



**Figure III.2 : Redis Maître / Esclave**

### 2.1.3.2. Sentinel

Redis propose un service indépendant pour palier au problème de récupération en cas de panne. Pour cela, en plus des divers serveurs s'occupant de la base de données, des serveurs Redis Sentinel ont été conçus, récemment, pour servir et gérer le cluster des serveurs (Figure III.3).

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 un serveur maître échoue, Sentinel s'occupe de le remplacer en promouvant un serveur esclave**.[Hei12]** .

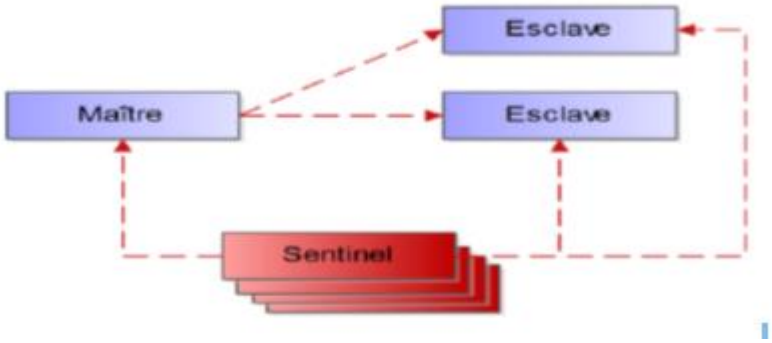



**Figure III.3 : Redis Sentinel**

### 2.1.3.3. Cluster

Pour configurer un cluster, Redis procède à la répartition des données 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 (Figure III.4) **.[Hei12]**

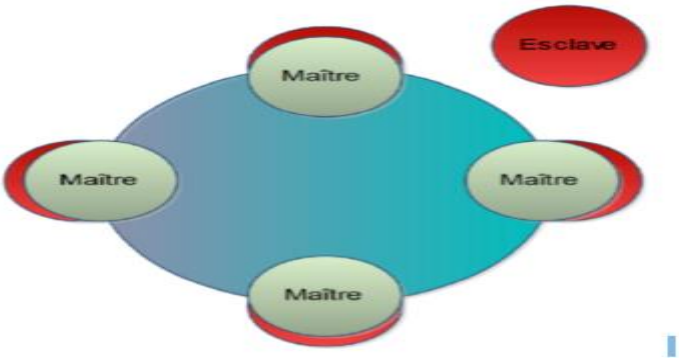



**Figure III.4 : Redis Cluster**

#### 2.1.3.4. Réplication

Chaque fois qu'une transaction est effectuée sur le maître, ce dernier la reproduit directement sur les différents esclaves qui lui sont liés, ce qui permet de les maintenir à jour. A leur tour, ces derniers vont ensuite la répercuter sur les esclaves liés à eux s'il y en a. **[Hei12]**

## 2.2. MongoDB

### 2.2.1. Description

MongoDB est une base de données orientée documents open-source, distribué sous licence AGPL (licence libre), fournissant de hautes performances, une haute disponibilité et une scalabilité automatique.

MongoDB a été développé en C++ depuis 2007 par la compagnie 10gen qui travaillait sur un système de Cloud Computing à données largement réparties, semblable au service App Engine de Google. La première version est apparue en 2009, mais c'était en 2010 que la version 1.4 a été considérée comme industriellement viable.

MongoDB est évolutif, très performant, simple d'utilisation, et bien conçu pour gérer le stockage adapté aux applications à grande échelle, et peut s'exécuter dans un environnement distribué et multi plateformes. De ce fait, MongoDB a été adopté par plusieurs grandes compagnies telles que Four square, SAP, et GitHub.

MongoDB permet de manipuler des objets structurés (documents) au format BSON (Binary JSON), un dérivé de JSON binaire plus axé sur la performance, qui a été pensé pour faciliter le scan des données. Fonctionnant comme une architecture distribuée-centralisée, il réplique les données sur plusieurs serveurs avec le principe de maître-esclave ou primaire/secondaire, permettant ainsi une plus grande tolérance aux pannes (Fail over)et la répartition de la charge de travail entre les nœuds (load balancing).

L'architecture distribuée de MongoDB se base sur deux principes :

1- la répartition des données (sharding).
2- la réplication (réplica set).

La répartition et la duplication de documents est faite de sorte que les documents les plus demandés soient sur le même serveur et que celui-ci soit dupliqué un nombre de fois suffisant. Par sa simplicité d'utilisation du point de vue de développement client, ainsi que ces performances remarquables, MongoDB est la base de données orientée document la plus utilisée. **[Site 6]**

### 2.2.2. Modèle de données

Le modèle de données de MongoDB est de type orienté documents.

- **Un document :** l'unité basique de MongoDB, est une structure de données composée de paires de champs et de valeur, et qui est identifié par son nom.
- **Les valeurs des champs :** peuvent inclure d'autres documents, des tableaux et des tableaux de documents **[Site 6].** Les données prennent la forme de documents enregistrés dans des collections.
- **Une collection :** contenant un nombre quelconque de documents. Les collections sont comparables aux tables, et les documents aux enregistrements dans les bases relationnelles. Afin d'augmenter les performances, tout en manipulant des documents, MongoDB utilise l'indexation similaire aux bases de données relationnelles**.[Abr et all 13]** Néanmoins et contrairement aux bases relationnelles, MongoDB est sans schéma prédéterminé. MongoDB ne pose aucune restriction quant aux documents contenus dans une même collection. A la différence d'une table SQL, le nombre de champs des documents d'une même collection peut varier d'un document à l'autre.
- **Une base de données MongoDB :** est un conteneur de collections, tout comme une base de données SQL contenant des tables. Par ailleurs, MongoDB ne permet pas les requêtes très complexes standardisées, mais il permet de programmer des requêtes spécifiques en JavaScript.

### 2.2.3. Architecture

MongoDB possède quatre différents modes de fonctionnement **:**

#### 2.2.3.1. Single

Le mode Single (Figure III.5) : sert à faire fonctionner une base de données sur un seul serveur. Dans ce mode, un seul processus nommé **mongod** est utilisé pour traiter directement les données issues des requêtes du client. [**Hei12]**

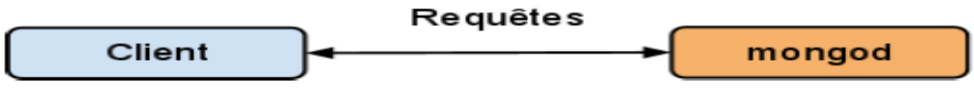



**Figure III.5 : MongoDB en mode serveur seul**

#### 2.2.3.2. Réplication Master / Slave

Dans ce mode Master/Slave (Figure III.6), le serveur fonctionne en tant que maître et s'occupe des demandes clients. En plus il s'occupe de répliquer les données sur le serveur esclave de façon asynchrone. L'esclave est présent pour prendre la place du maître si ce dernier tombe en

panne. Le premier avantage de cette structure est de garantir une forte cohérence des données, en centralisant les opérations clients sur le maître. Aucune opération n'est faite sur l'esclave, hormis les mises à jour envoyées par le maître vers l'esclave**. [Hei12]**

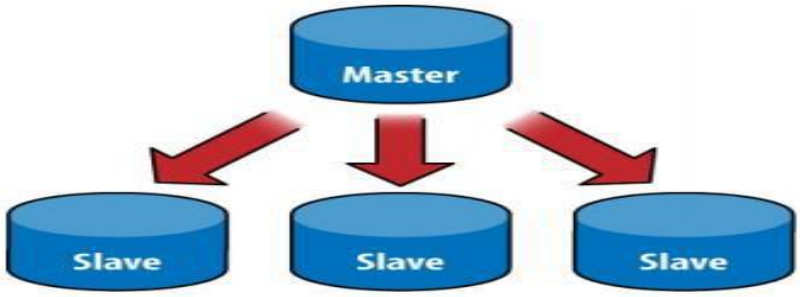



**Figure III.6 : MongoDB Maître / Esclave**

### 2.2.3.3. Réplica Set

Il s'agit d'un mode de réplication maître / esclave plus avancé.

Le Réplica Sets fonctionne avec plusieurs nœuds possédants chacun la totalité des données (Figure III.7) :

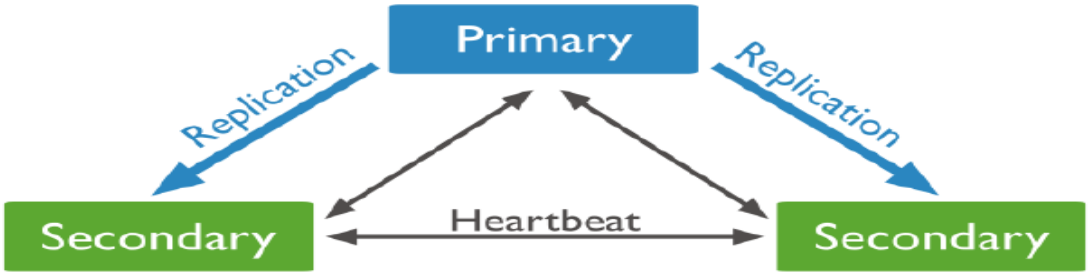



**Figure III.7 :MongoDB Réplica Sets**

Ces différents nœuds vont alors élire un nœud primaire, qui peut s'apparenter à un maître. Pour qu'il soit élu, Il faut qu'un nœud obtienne la majorité absolue. 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 insérer dans le système pour participer, seulement, aux élections afin de pouvoir garantir la majorité absolue. Le rôle de nœud primaire est vu comme celui du serveur maître dans le modèle maître/esclave. En cas de défaillance du nœud primaire, un nouveau nœud au sein du Réplica Set est choisi pour devenir nœud primaire. **[Hei12]**

### 2.2.3.4. Sharding

Le Sharding est une surcouche qui est basée sur du Master / Slave ou du Replica Sets (Figure III.8):

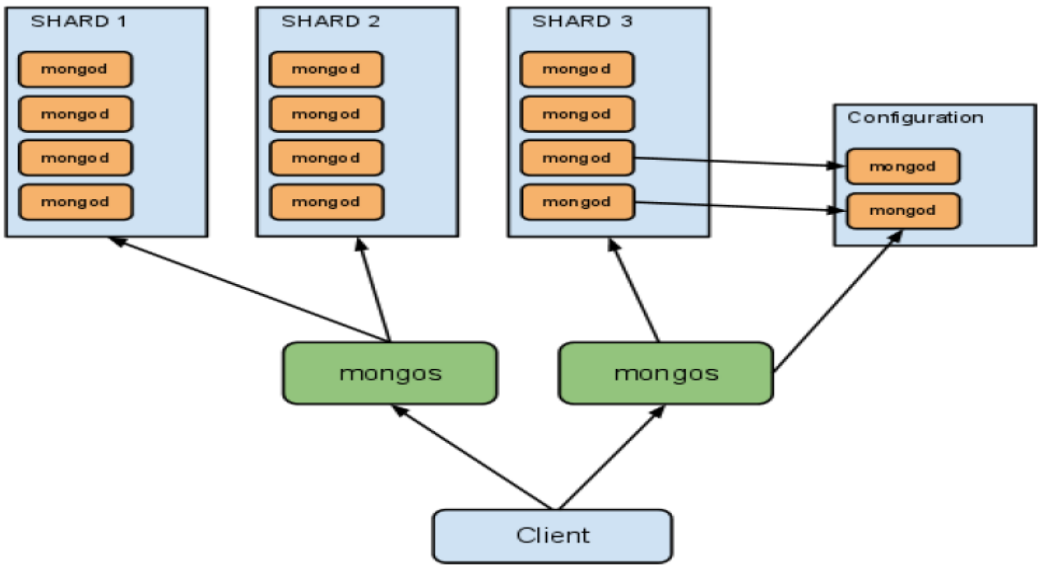



**Figure III.8 : Partitionnement des donnés via Sharding**

Le Sharding sert à partager les données entre plusieurs **Shards**, chaque **Shard** devant stocker une partie des données. La définition de la manière dont les données seront découpées se fait grâce à une **Hash Shard Key**qui permet de définir sur quel critère seront partagées les données entre les **Shards. [AMW15]**

− **Shards**: Un ou plusieurs serveurs en mode Master / Slave ou Réplica Sets. Via son processus **mongod**, ils prennent en charge le stockage des données.

− **Mongos** : Des processus qui redirigent les requêtes des applications clientes vers les **shards** adéquats et regroupent les résultats avant de les transmettre à l'application cliente. Toute modification dans les serveurs de configuration est propagée immédiate mentaux processus **mongos**.

− **Config Servers** : Les serveurs de configurations stockent les métadonnées du système relatives à chaque **shard** et des informations sur le placement des données dans les serveurs respectifs (**shards**).

Ce service est indispensable, il est donc nécessaire de l'assurer en lui consacrant deux, voire trois instances. L'écriture sur ces services utilise le protocole de validation en deux phases pour assurer un stockage fiable. En cas de défaillance d'un service, tous les autres serveurs passent en mode de lecture seule.

### 2.2.4. Manipulation des données

MongoDB propose un langage assez riche (sans doute parmi les plus riches du NoSQL). Le langage natif de la base de données MongoDB est le JavaScript, par lequel les documents Json sont créés.

Cependant, MongoDB est livré aussi avec des liaisons pour les principaux langages deprogrammation : Perl, PHP, Python, Ruby, Scala, Go, C, C++, Dart, Erlang, Haskell, Java, JavaScript, et .NET (C# F#, PowerShell, etc).

Les pilotes intégrés permettent de manipuler la base et ses données directement depuis ces langages.

MongoDB possède également un outil **mongo** en ligne de commande interactif qui permet d'accéder et manipuler directement la base de données. Voici dans la tableau III.3, une liste non-exhaustive de commandes MongoDB :

| Type de commandes | Quelques Opérations ou fonctions |
|---|---|
| Agrégation | count, distinct, group, aggregate, ... |
| Géo-spatiales | geoNear, geoSearch, geoWalk, ... |
| Lecture/écriture | delete, insert, update, eval,find, findAndModify, text, getLastError, parallel Collection Scan, ... |
| Base de données | authenticate, logout, createUser, dropUser, usersInfo, ... |
| Gestion des droits | createRole, grantRolesToUser, dropRole, grantPrivilegesToR, updateUserole, rolesInfo, updateRole,... |
| Répartition et réplication | isMaster, replSetGetStatus, resync, addShard, enableSharding, getShardMap, listShards, movePrimary, split, moveChunk, ... |
| Administration | clean, cloneCollection, copydb, create, drop, reIndex, shutdown, setParameter, repairDatabase, compact,... |

**Tableau III.3: Commandes utiles de MongoDB**

## 2.3. Cassandra

### 2.3.1. Description

Cassandra est une base de données NoSQL orientée colonne. Elle a été conçue pour pallier au problème de performances des bases de données relationnelles, ainsi qu'aux problèmes de gestion de grands volumes de données. Par ailleurs, elle pallie à la complexité de déploiement et au maintien de l'intégrité lors de déploiement en cluster et dans différents Datacenter. Cassandra est conçue pour gérer des quantités massives de données réparties sur plusieurs serveurs (Cluster), en assurant tout particulièrement une disponibilité maximale des données et en éliminant les points individuels de défaillance. **[MH18]**

Initialement, elle était développée par Facebook , afin de répondre à des besoins concernant son service de messagerie, puis elle a été libéré en Open-source et a été adopté par d'autres grands acteurs 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.

Des efforts considérables se consacrent actuellement à Cassandra, pour ajouter de nouvelles fonctionnalités au fur et à mesure comme l'amélioration de ses requêtes. Cependant les

discussions ne s'arrêtent pas quant au fait que la configuration d'un cluster Cassandra n'est pas une partie de plaisir. Le point le plus notable reste DataStax, l'entreprise derrière Cassandra, qui s'efforce d'intégrer des fonctionnalités de Data Mining. Depuis le début Cassandra n'était pas considérée comme un puissant système de requêtes, mais depuis peu et avec l'ajout de nouvelles fonctionnalités, il devient envisageable de l'exploiter dans un contexte d'analyse des données.

### 2.3.2. Caractéristiques

Les principales caractéristiques de la base de données NoSQL Cassandra sont **[MH18]**:

#### 2.3.2.1. Tolérance aux pannes

Les données d'un nœud sont automatiquement répliquées sur différents nœuds. Ainsi, si un nœud est hors service les données présentes sont disponibles à travers d'autres nœuds. Le terme facteur de réplication désigne le nombre de nœuds où la donnée est répliquée. Conçu sur le principe qu'une panne n'est pas une exception mais une normalité, il est simple de remplacer un nœud défaillant en assurant la continuité et la disponibilité du service. Par ailleurs, l'architecture de Cassandra définit le terme de cluster comme étant un groupe d'au moins deux nœuds et un data center comme étant des clusters délocalisés. Cassandra permet d'assurer la réplication à travers différents data center.

#### 2.3.2.2. Décentralisé

Tous les nœuds, dans un cluster Cassandra, sont identiques. Il n'y pas de notion de maitre, ni d'esclave, ni de processus qui aurait à sa charge la gestion globale du système, ni même de goulot d'étranglement au niveau de la partie réseau.

#### 2.3.2.3. 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.

#### 2.3.2.4. Elastique

La scalabilité est linéaire : le débit d'écriture et de lecture augmente automatiquement de façon linéaire lorsqu'un nouveau serveur est ajouté dans le cluster.

#### 2.3.2.5. Haute disponibilité

Cassandra assure qu'il n'y aura pas d'indisponibilité du système ni d'interruption au niveau des applications. Elle a la possibilité de spécifier le niveau de cohérence concernant la lecture et l'écriture. L'écriture des données est très rapide comparée au monde des bases de données relationnelles, puisque Apache Cassandra n'utilise pas le principe de transaction.

### 2.3.3. Architecture

Cassandra est basée sur une architecture distribuée, tous les nœuds du Cluster ont le même rôle et permettent donc la lecture et l'écriture des données au sein de la base NoSQL. Le

client peut alors se connecter à n'importe quel nœud pour accéder aux données. Il n'y a donc plus de SPOF (Single Point Of Failure). La communication entre les différents nœuds du Cluster se fait en « peer to peer ». Un utilisateur se connecte sur l'un des nœuds lors de l'ouverture de sa session sur le serveur, si le nœud sur lequel s'est connecté l'utilisateur, dispose de l'information que celui-ci recherche, le nœud répondra à la requête de l'utilisateur. Si toutefois ce n'est pas le cas, le nœud va communiquer avec ceux qui ont l'information pour ainsi restituer les données à l'utilisateur. Cassandra reprend les concepts de 2 bases de données très réussies : La première est BigTable, créé par Google, pour son modèle de données orienté colonne et son mécanisme de persistance sur disque, et la seconde est Dynamo, créé par Amazon, pour son architecture distribuée sans nœud maître. **[Mal12]**

### 2.3.4. Modèle de données

Le modèle de données de Cassandra s'appuie sur un schéma dynamique, avec un modèle de données orienté colonne. Les colonnes et leurs métadonnées peuvent être ajoutées par l'application lorsque cela s'avère nécessaire. Dans Cassandra, le Keyspace est le conteneur des données de l'application (un peu comme une base de données ou un schéma pour une base de données relationnelle) **[Site 7].**

Dans ces Keyspaces se trouvent une ou plusieurs familles de colonnes (qui correspondent aux tables en base de données relationnelle). Ces familles de colonnes contiennent des colonnes ainsi qu'un ensemble de colonnes connexes qui sont identifiées par une clé de ligne. En outre, chaque ligne d'une famille de colonnes ne dispose pas nécessairement des mêmes colonnes qu'une autre ligne (Figure III.9). Enfin, Cassandra n'impose pas de relations entre les familles de colonnes au sens base de données relationnelles: il n'y a pas de clés étrangères et les jointures entre familles de colonnes ne sont pas supportées.

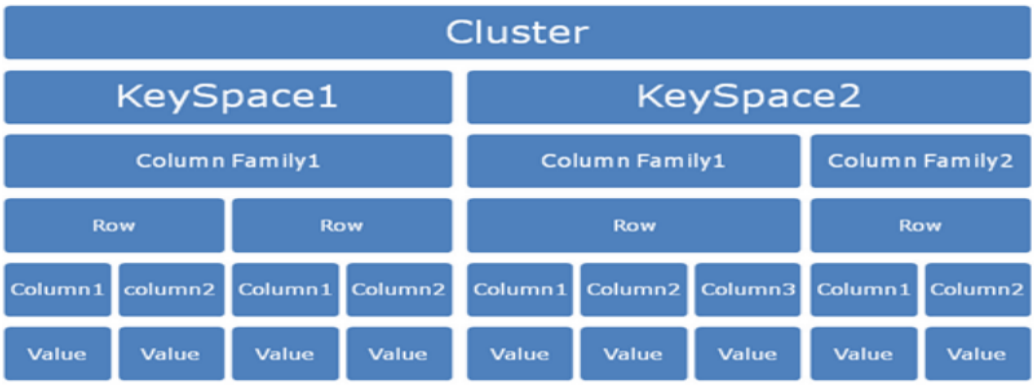



**Figure III.9 : Structure d'un keyspace Cassandra**

De plus, chaque ligne peut disposer d'un ensemble de colonnes différentes. Par contre, bien que les familles de colonnes puissent être flexibles, dans la pratique, il est conseillé d'y

associer une sorte de schéma (à une famille de colonnes, il est préférable de n'y mettre qu'un même type de données). Pour toutes les familles de colonnes, chaque ligne est identifiée de manière unique par sa clé (un peu comme la notion de clé primaire pour les bases de données relationnelles). En fait, dans Cassandra, une colonne est le plus petit élément de données. Elle est modélisée par un tuple qui contient le nom, la valeur et un timestamp (Tableau III.4). Ce timestamp est utilisé par Cassandra pour déterminer la mise à jour la plus récente dans la colonne, et dans ce cas, c'est la plus récente qui est prioritaire lors d'une requête.

| Coloumn_name |
| --- |
| Value |
| Timestamp |

**Tableau III.4: Structure d'une colonne Cassandra**

Une colonne doit avoir un nom qui peut être un label statique (comme "nom" ou "email") ou peut être définit dynamiquement lorsque la colonne est créée par l'application. Les colonnes peuvent être indexées par leur nom (en utilisant un second index).

### 2.3.5. Partitionnement des données dans un cluster Cassandra

Le partitionnement détermine la façon dont les données sont réparties à travers les nœuds du cluster (y compris les répliques). Fondamentalement, un partitionnement est une fonction de hachage pour calculer le jeton (c'est le hash) de la clé d'une ligne. Chaque ligne de données est identifiée de manière unique par une clé distribuée à travers le cluster par la valeur du jeton **[Site 8].**

Cassandra emploie un ensemble de nœuds équivalents ou l'ajout d'un nouveau nœud est très simple puisqu'il suffit 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 (Figure III.10). L'anneau est divisé en plages égales au nombre de nœuds, chaque nœud étant responsable d'un ou plusieurs plages de données. Avant qu'un nœud puisse rejoindre l'anneau, on doit lui a assigné un jeton. La valeur de jeton détermine la position du nœud dans l'anneau et sa plage de données. Les données d'une table (column family) sont partitionnées sur les nœuds en se basant sur la valeur du hash de la clé de chaque ligne.


**Figure III.10 : Ajout d'un nœud dans un cluster Cassandra**

## 3. Travaux voisins

Vu la nouveauté du sujet, plusieurs études ont été menées sur les bases de données NoSQL durant les cinq dernières années.

- En 2015, un article **[Kum et all 15]** a étudié le remplacement d'un SGBD relationnel par un NoSQL, en occurrence MySQL par MongoDB, et en conclusion, il a mis en cause l'extensibilité du MySQL.
- En 2016, une thèse de projet de fin d'études en Master **[GDE 16]** a proposé une Etude critique des contextes ACID et BASE des bases de données NoSQL ; la conclusion du travail était une comparaison entre quatre bases sur les quatre critères ACID.
- Dans la même année, un autre article **[DEV et all 16]** a mené une étude similaire mais en ajoutant d'autres critères tels que l'évolutivité, la performance théorique, la fiabilité, la flexibilité et la complexité.
- En 2018, une étude théorique comparative a été menée sur 10 systèmes Big Data **[OH et all 18]** et leurs critères techniques servant à aider les professionnels à prendre la décision par rapport au choix à faire.
- Plusieurs autres études et articles sont apparus traitant du même sujet ; répartis entre étude comparative SQL NoSQL, ou bien une étude comparative inter-NoSQL. Notre étude sera différente car elle permettra la mise sur un banc d'essai pratique, le YCSB, en vue de tester des données réelles et avoir des résultats de comparaison entre trois fameuses bases de données NoSQL, Redis, MongoDB et Cassandra dans des configurations différentes et mesurer leurs performances respectives.

## 4. 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 IV

## Etude Comparative Et Résultats Expérimentaux

## 1. Introduction

Notre étude consiste à développer une étude comparative sur les performances de trois solutions très répandues dans le marché **Redis, MongoDB, Cassandra** à l'aide de l'outil **YCSB**. Ce dernier étant un outil très connu pour sa puissance de test et utilisé dans plusieurs travaux d'évaluation des bases de données NoSQL.

Plusieurs études comparatives, dans le même domaine, à base **d'YCSB**, ont été récemment réalisées. L'étude à laquelle nos résultats seront comparés, a été réalisée dans deux machines, On tient à signaler que notre étude n'a pas inclus les bases de données orientées graphes, du fait que celles-ci ne doivent pas être évalués selon les mêmes scénarios utilisés dans l'analyse des autres types de bases de données NoSQL (orientées clé/valeur, orientées colonne et orientées document), puisque l'utilisation des liens entre les enregistrements nécessitent une approche différente basée sur d'autres critères pour évaluer les performances des bases de données de graphes.

Ce chapitre sera consacré, à la description du benchmark utilisé et les différentes charges de travail employées, et aussi à la présentation et l'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.

## 2. Benchmark utilisé et charges de travail

Yahoo propose un outil très puissant appelé YCSB ( Yahoo ! Cloud Serving Benchmark) : pour comparer les performances des systèmes NoSQL. Il s'agit d'une nouvelle méthodologie de bancs d'essais (benchmark) open-source où les utilisateurs peuvent développer leurs propres paquets soit en définissant de nouveaux paramètres pour les charges de travail, soit en les écrivant en code Java.

YCSB a été annoncé dans un article de Yahoo! dans lequel ils ont présenté des résultats de comparaison pour quatre systèmes largement utilisés : Apache HBase, Apache Cassandra, Yahoo! PNUTS et une implémentation MySQL distribué, à base de leurs caractéristiques de performance et d'élasticité. **[CBF et all 10]**

Le benchmark mesure les performances brutes, affiche les caractéristiques de latence au fur et à mesure qu'il y a une montée en charge d'un serveur, mesure l'extensibilité, et montre comment les systèmes comparés s'adaptent rapidement à un éventuel passage à l'échelle **[Cat et all 10].** Il est devenu un outil de référence standard pour les systèmes NoSQL. Cet outil est multiplateforme et prend en charge la majorité des systèmes NoSQL et s'adapte facilement à ces solutions.

YCSB se compose de deux composants :

- un générateur de données.
- un ensemble de tests de performance pour évaluer les opérations d'insertion, de mise à jour et de suppression des enregistrements. Chaque test est une charge de travail, ou on peut configurer le nombre d'enregistrements à charger, le nombre d'opérations à exécuter et la proportion de lecture et d'écriture.

Les opérations prises en charge comprennent :

- l'insertion,
- la mise à jour (modifier un des champs),
- la lecture (un champ aléatoire ou tous les champs d'un enregistrement) ainsi que
- le scan (lire les enregistrements dans l'ordre du démarrage à partir d'une clé d'un enregistrement sélectionné au hasard).

Chaque charge de travail (Workload) utilise des paramètres de référence différents, qui peut être modifiée et personnalisée en fonction du type de résultats attendus. La liste des charges de travail, fournis dans ce mémoire, inclut :

- **Workload A** (Update Heavy : 50% Read - 50% Update) : La charge de travail « mise à jour lourde » est constituée de 50% d'opérations de lecture et 50% de mise à jour.
- **Workload B** (Read Mostly : 95% Read - 5% Update) : La charge de travail « lire principalement » contient majoritairement des opérations de lecture 95% et 5% d'opérations de mise à jour.
- **Workload C** (Read Only : 100% Read) : Le test « Lecture seulement » est composée de 100% d'opérations de lecture.
- **Workload D** (Read Latest : 95% Read - 5% Insert) : Un test ou on insère 5% d'enregistrements, avec 95% de lecture de données récemment insérées.
- **Workload E** (Short Ranges : 95% Scan - 5% Insert) : Cette charge de travail consiste à faire 95% d'opérations de scan de la base de données et 5% d'opérations d'insertion.
- **Workload F** (Read-Modify-Write : 50% Read - 50% Read-Modify-Write): Un test de 50% de lecture, dans l'autre moitié, les enregistrements sont lus, modifiés puis sauvegardés.

## 3. Environnement de développement

Notre étude a été menée dans deux machines physique avec la configuration suivante :

**1ere machine :**

- Processeur : Intel® Core(TM) i3-233OM CPU @ 2.20 GHz
- RAM : 4.00 Go
- Système : Microsoft Windows10 Professionnel 64 bits

- Disque dur: 450 GB

**2eme machine:**

- Processeur : Core i3 Intel® Core™ i3-3210M CPU @ 2.50 GHz x 2.
- RAM : 8.00 GB.
- Système :Ubuntu 18.04 , PC 64-bit.
- Disque dur: 370 GB.

L'étude comparative menée a employé les différents outils et systèmes NoSQL, récapitulés dans le tableau suivant :

| Système | Ubuntu 18.04 | Windows 10 |
|---|---|---|
| le benchmark YCSB | Version 0.18.0 | Version 0.17.0 |
| Redis (clé/valeur) | Version 4.0.9 | Version 3.0.504 |
| Mongodb(oriente document) | Version 3.6.3 | Version 4.4.0 |
| Cassandra(oriente colonne) | Version 3.11.6 | Version 3.11.1 |

**Tableau IV.1: Versions des systèmes et outils employés**

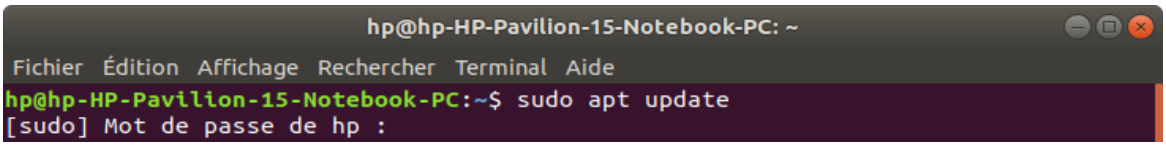
## 4. Configuration et Analyse des BDs NoSql sous Ubuntu 18.04

### 4.1. Première partie : Configuration et Installation

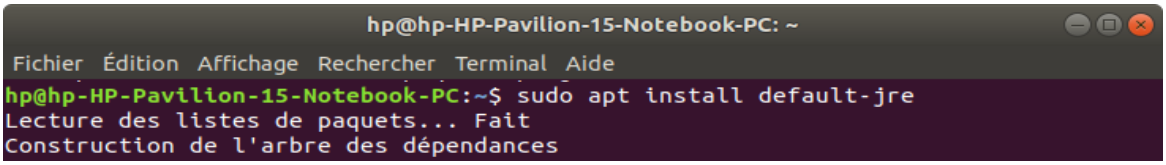
Dans cette partie, nous allons sur Ubuntu 18.04 Pour cela, on va suivre les étapes suivantes:

#### 4.1.1. La mise à jour du système d'exploitation Ubuntu

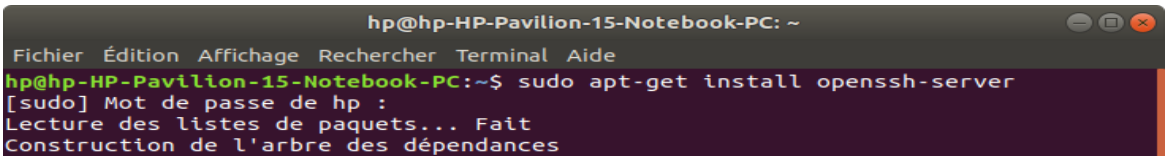
- **Etape 1** : avec la commande **sudo apt-get update**
  On doit fournir le mot de passe de l'utilisateur



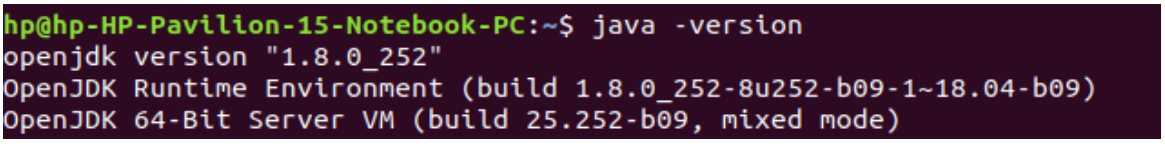

- **Etape 2 :** Installation la version récente de java



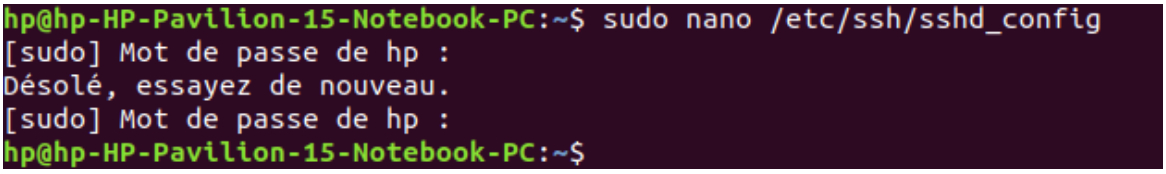
- **Etape 3** : Vérification de la version de java



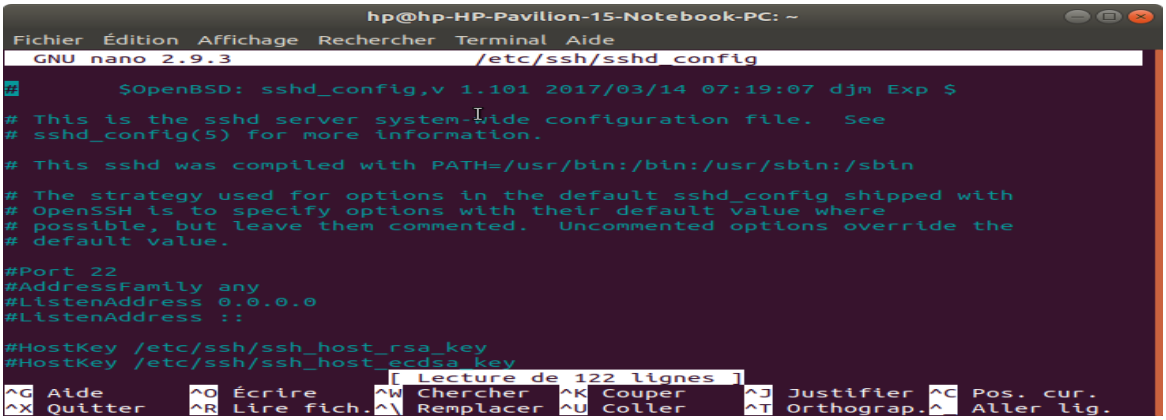

- **Etape 4 :** Installation du openssh-server



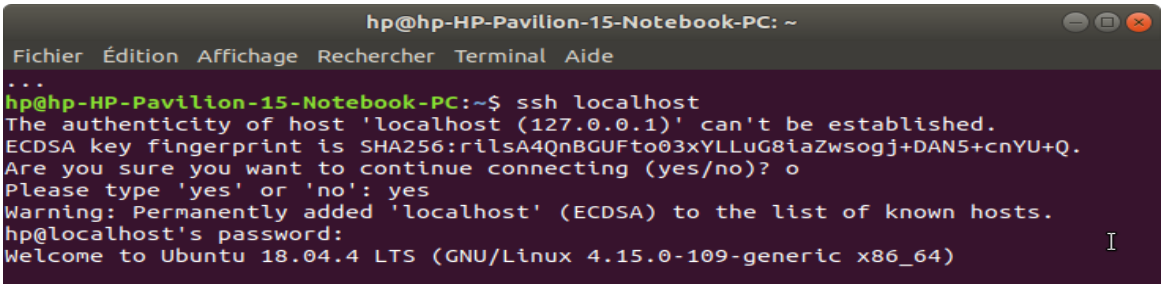

- **Etape 5** : Ouvrir le fichier sshd_config pour modifier l'adresse localhost




- **Etape 6** : Modifier ListenAddress 0.0.0.0 au 127.0.0.1




**Etape 7** : Vérifier si open ssh-server est bien installer



- **Etape 8** : Démarrer ssh-server avec les 2 commandes suivantes :

```
hp@hp-HP-Pavilion-15-Notebook-PC:~$ sudo systemctl restart ssh
```

```
hp@hp-HP-Pavilion-15-Notebook-PC:~$ ssh @
usage: ssh [-46AaCfGgKkMNnqsTtVvXxYy] [-b bind_address] [-c cipher_spec]
           [-D [bind_address:]port] [-E log_file] [-e escape_char]
           [-F configfile] [-I pkcs11] [-i identity_file]
           [-J [user@]host[:port]] [-L address] [-l login_name] [-m mac_spec]
           [-O ctl_cmd] [-o option] [-p port] [-Q query_option] [-R address]
           [-S ctl_path] [-W host:port] [-w local_tun[:remote_tun]]
           [user@]hostname [command]
hp@hp-HP-Pavilion-15-Notebook-PC:~$
```

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

Par les 2 commandes :

```
sudo adduser hp sudo
su - hduser
```

ou bien utiliser l'administrateur **root** directement par la commande :

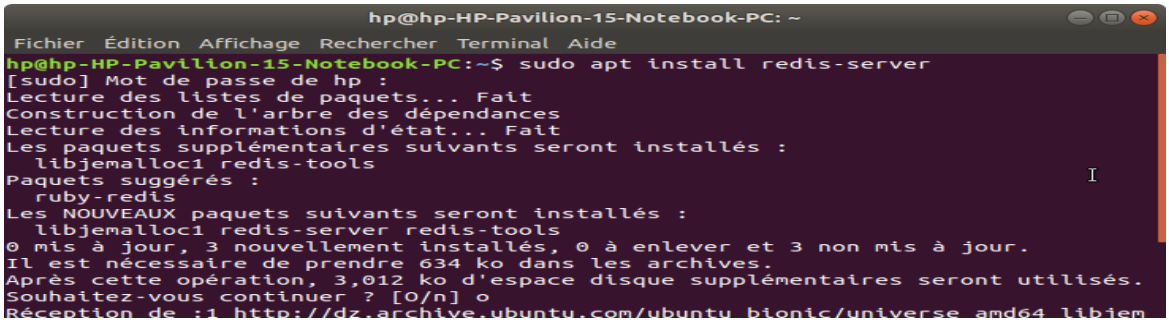
```
Sudo su -
```

### 4.1.2. Configuration de Redis V 4.0.9

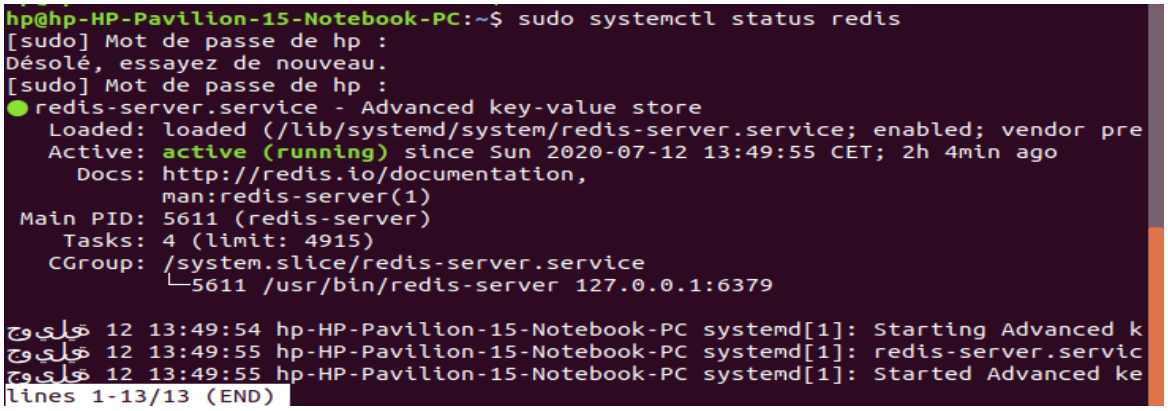
- **Etape1** : Commencez par mettre à jour la liste des packages pour avoir la version la plus récente des listes référentiels par **sudo apt update**

```
hp@hp-HP-Pavilion-15-Notebook-PC: ~
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
hp@hp-HP-Pavilion-15-Notebook-PC:~$ sudo apt update
[sudo] Mot de passe de hp :
Réception de :1 http://ppa.launchpad.net/openjdk-r/ppa/ubuntu bionic InRelease [
15.4 kB]
Réception de :2 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.
7 kB]
Atteint :3 http://dz.archive.ubuntu.com/ubuntu bionic InRelease
Réception de :4 http://dz.archive.ubuntu.com/ubuntu bionic-updates InRelease [88
.7 kB]
Atteint :6 http://ppa.launchpad.net/webupd8team/java/ubuntu bionic InRelease
Réception de :8 http://dz.archive.ubuntu.com/ubuntu bionic-backports InRelease [
74.6 kB]
Réception de :9 http://ppa.launchpad.net/openjdk-r/ppa/ubuntu bionic/main amd64
Packages [9,568 B]
Réception de :10 http://security.ubuntu.com/ubuntu bionic-security/main i386 Pac
kages [510 kB]
Réception de :11 http://dz.archive.ubuntu.com/ubuntu bionic-updates/main i386 Pa
ckages [719 kB]
Réception de :5 https://dl.bintray.com/apache/cassandra 311x InRelease [3,183 B]
Réception de :12 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Pa
ckages [796 kB]
Réception de :7 https://dl.bintray.com/apache/cassandra 39x InRelease [3,168 B]
Réception de :13 http://ppa.launchpad.net/openjdk-r/ppa/ubuntu bionic/main i386
Packages [9,540 B]
```

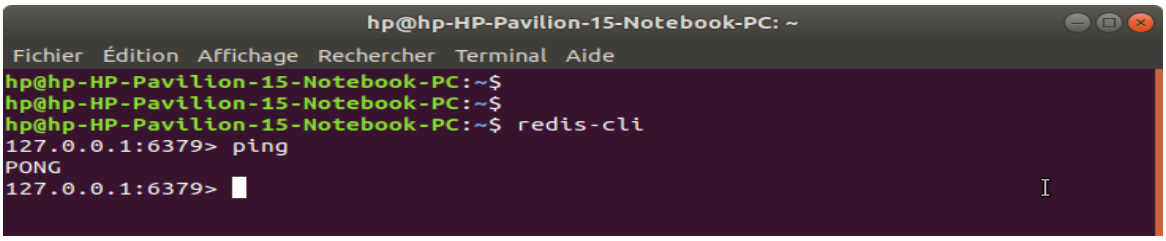
- **Etape 2 :** Installer Redis avec la commande **sudo apt install redis-server**




- **Etape 3 :** Il faut activer Redis-server avec la commande **sudo systemctl status redis**

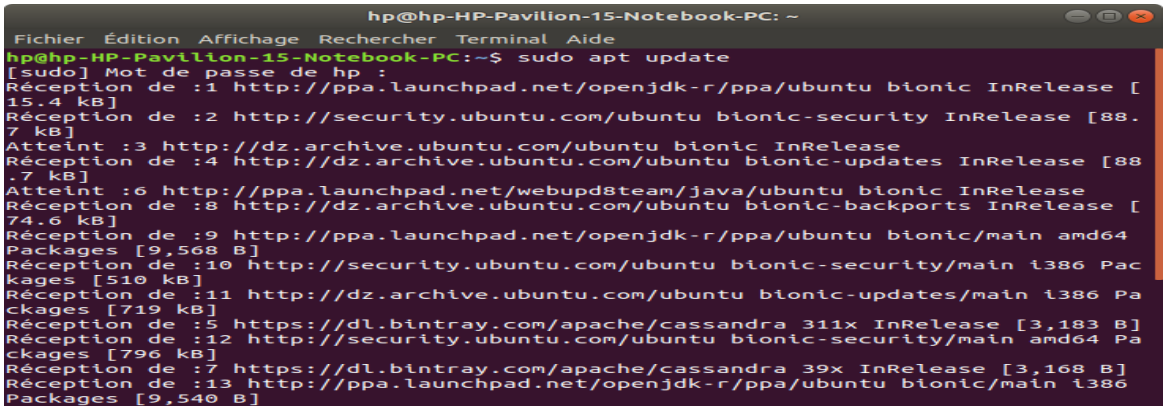



- **Etape 4 :** Il faut activer Redis-client **redis-cli**

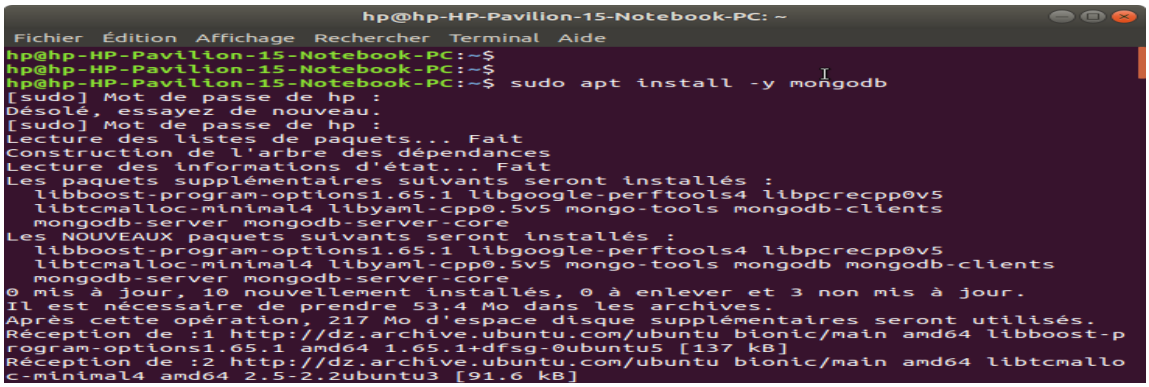



### 4.1.3. Configuration de Mongodb v3.6.3

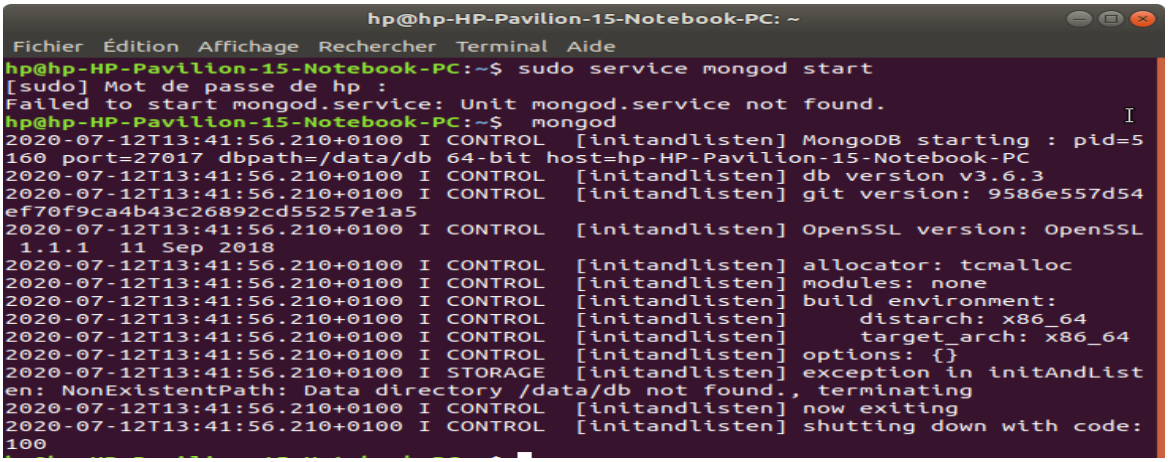
- **Etape 1 :** commencez par mettre à jour la liste des packages pour avoir la version la plus récente des listes référentiels par **sudo apt update**



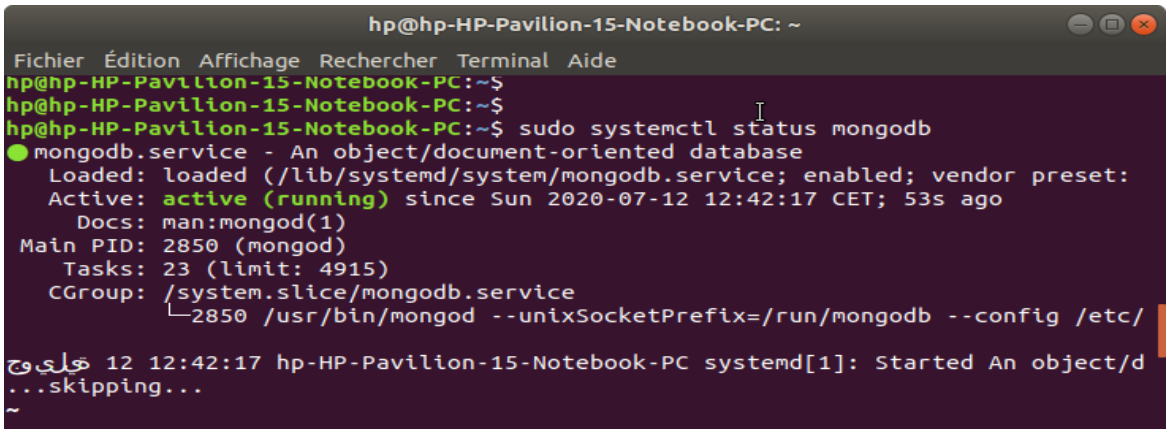
- **Etape 2 :** installez le paquet mongodb **sudo apt install –y mongodb**



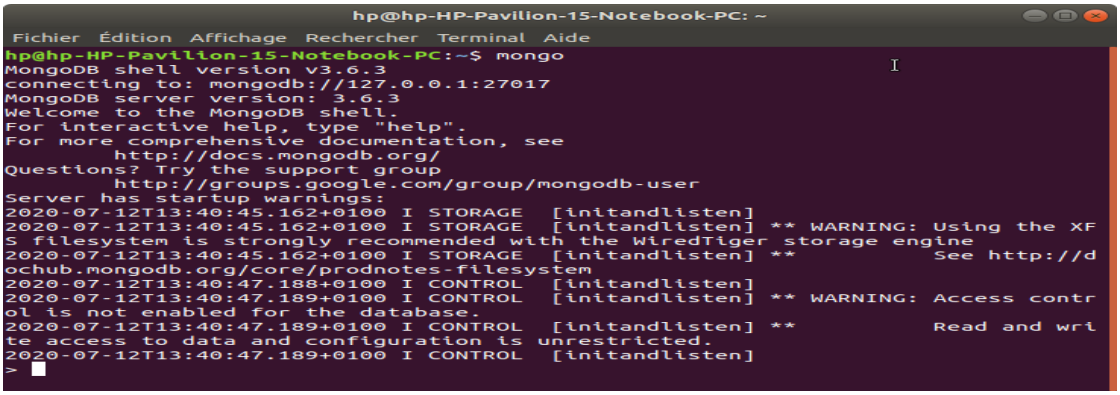

- **Etape 3: activez** mongodb service par **sudo systemclt status mongodb**




- **Etape 4 :** démarrage de serveur mongodb par simple commande **mongod**

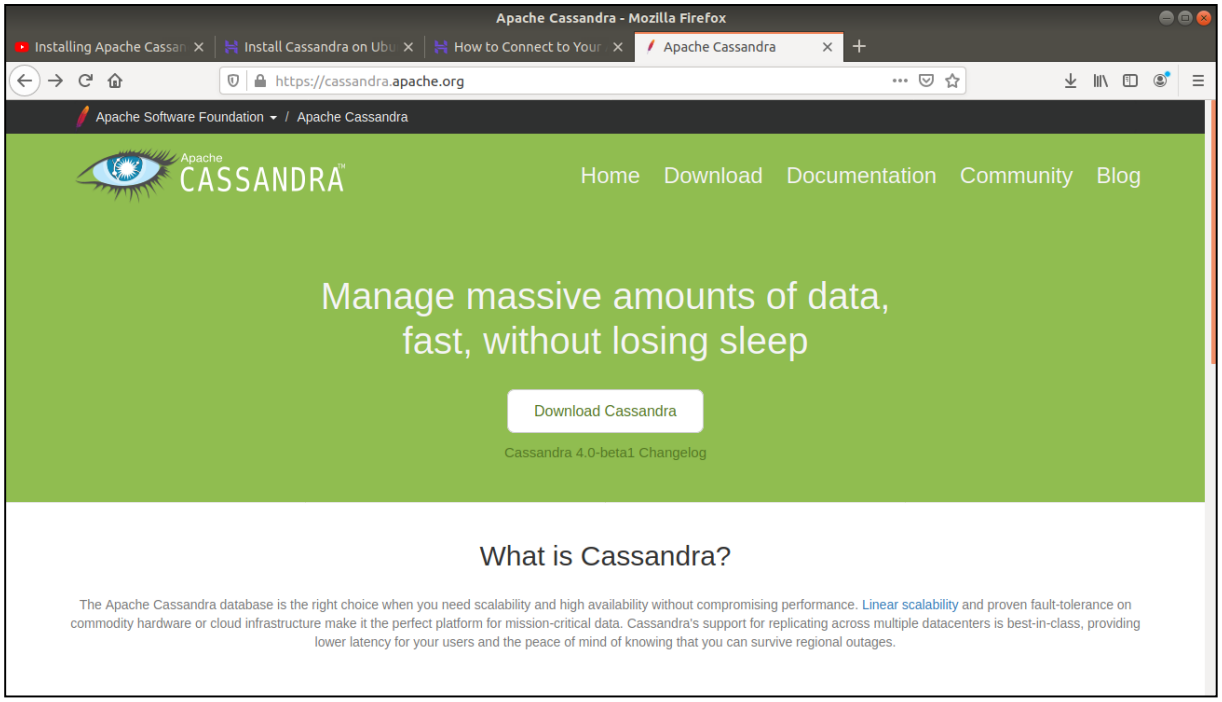



- **Etape 5 :** démarrage du mongodb client dans le même terminal ou autre avec la simple commande **mongo**

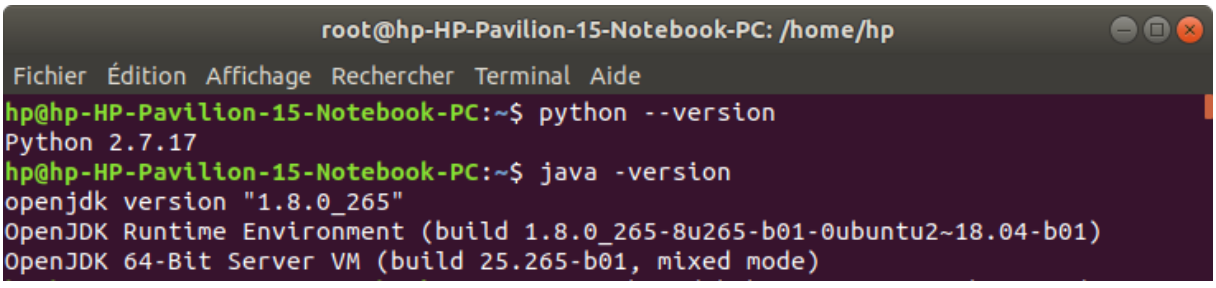



### 4.1.4. Configuration de Cassandra v3.11.7

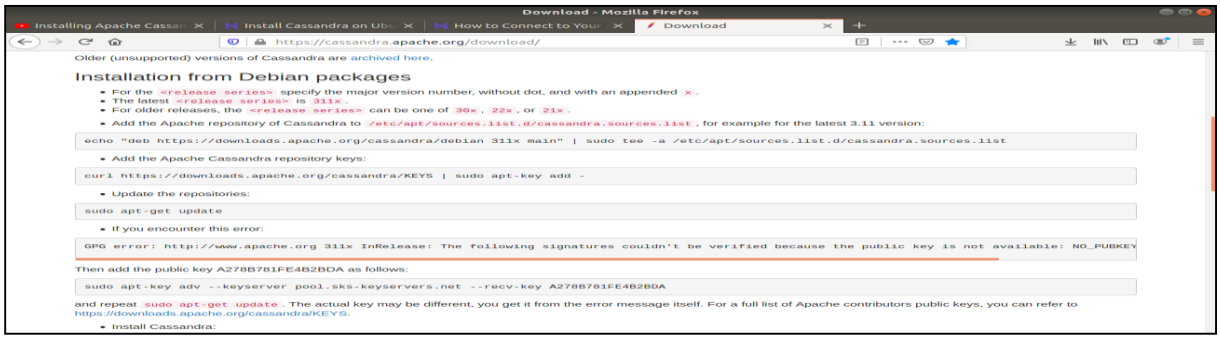
- **Etape 1 :** accéder au **[Site 9]** et cliquer sur **Download Cassandra**




- **Etape 2 :** vérifier la version de **python** et **java** s'ils ont installés dans le système ou non ( l'installation de **python** dans un nouveau cmd on tape **sudo apt install python** se fera automatiquement)



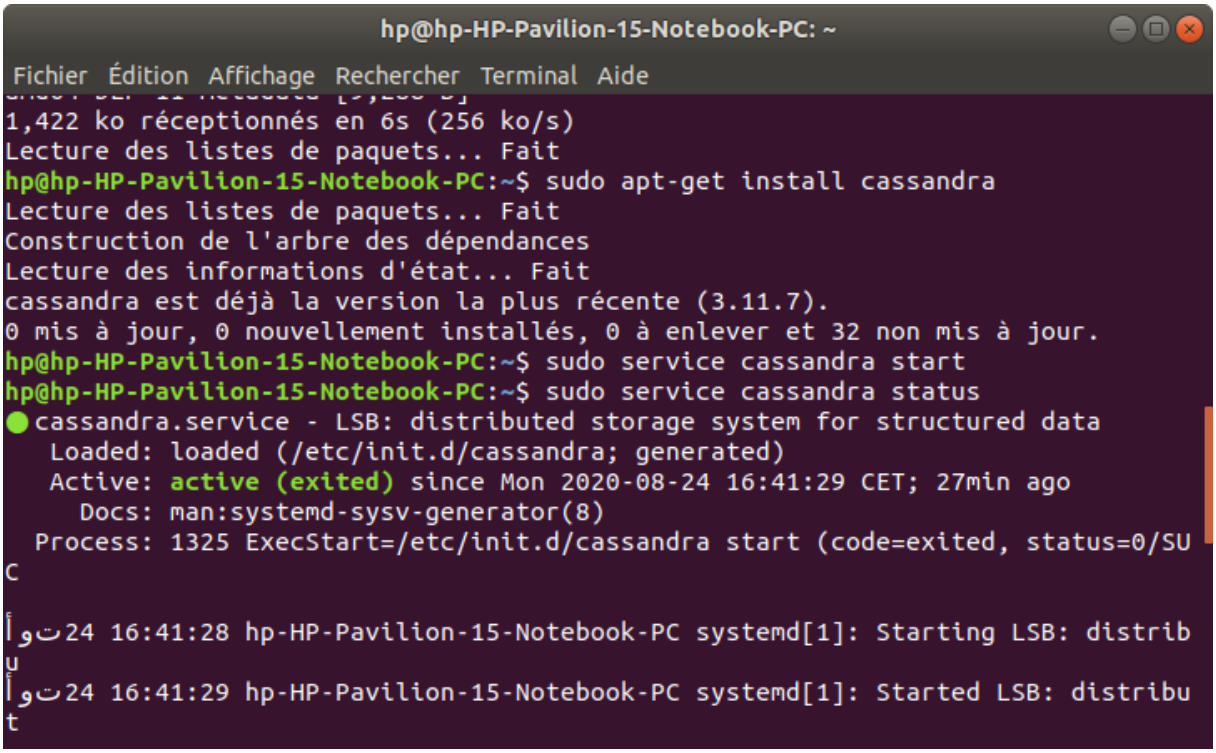
- **Etape 3 :** exécuter les étapes de l'installation à partir de « **Installation from Debian packages »**



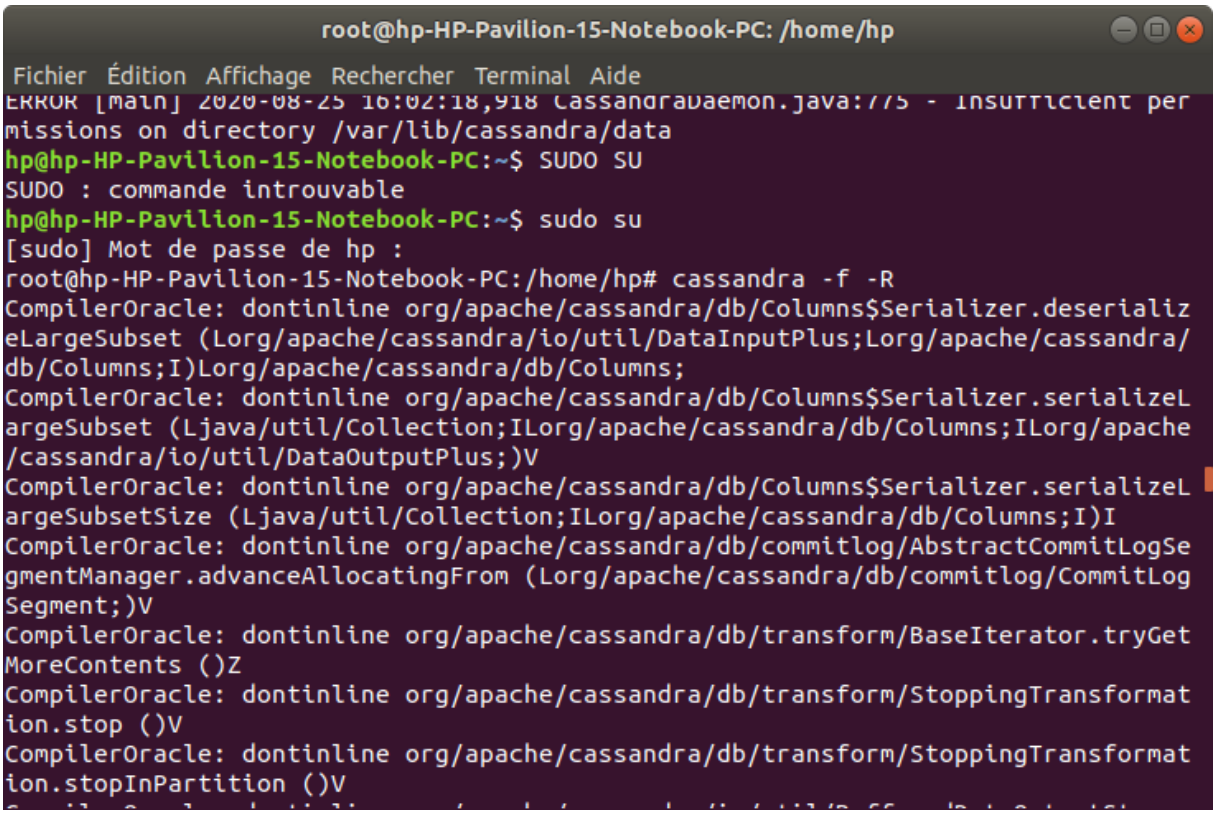

```
hp@hp-HP-Pavilion-15-Notebook-PC:~$     echo "deb http://www.apache.org/dist/cas
sandra/debian 36x main" | sudo tee -a /etc/apt/sources.list.d/cassandra.list
[sudo] Mot de passe de hp :
deb http://www.apache.org/dist/cassandra/debian 36x main
hp@hp-HP-Pavilion-15-Notebook-PC:~$     echo "deb http://www.apache.org/dist/cas
sandra/debian 39x main" | sudo tee -a /etc/apt/sources.list.d/cassandra.list
deb http://www.apache.org/dist/cassandra/debian 39x main
hp@hp-HP-Pavilion-15-Notebook-PC:~$ echo "deb https://downloads.apache.org/cassa
ndra/debian 311x main" | sudo tee -a /etc/apt/sources.list.d/cassandra.sources.l
ist
deb https://downloads.apache.org/cassandra/debian 311x main
```

```
root@hp-HP-Pavilion-15-Notebook-PC: /home/hp
 Fichier  Édition  Affichage  Rechercher  Terminal  Aide
deb https://downloads.apache.org/cassandra/debian 311x main
hp@hp-HP-Pavilion-15-Notebook-PC:~$ curl https://downloads.apache.org/cassandra/
KEYS | sudo apt-key add -
[sudo] Mot de passe de hp :   % Total    % Received % Xferd  Average Speed   Tim
e    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
 25  252k   25 65536    0     0   3200      0  0:01:20  0:00:20  0:01:00 13551
Désolé, essayez de nouveau.
[sudo] Mot de passe de hp :
100  252k  100  252k    0     0    678      0  0:06:21  0:06:20  0:00:01   711
OK
hp@hp-HP-Pavilion-15-Notebook-PC:~$ sudo apt-get update
Atteint :1 http://dz.archive.ubuntu.com/ubuntu bionic InRelease
Réception de :3 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.
7 kB]
Réception de :6 http://dz.archive.ubuntu.com/ubuntu bionic-updates InRelease [88
.7 kB]
Atteint :7 http://ppa.launchpad.net/openjdk-r/ppa/ubuntu bionic InRelease
Atteint :8 http://ppa.launchpad.net/webupd8team/java/ubuntu bionic InRelease
Réception de :9 http://dz.archive.ubuntu.com/ubuntu bionic-backports InRelease [
74.6 kB]
Réception de :10 http://dz.archive.ubuntu.com/ubuntu bionic-updates/main amd64 P
ackages [1,038 kB]
Réception de :11 http://security.ubuntu.com/ubuntu bionic-security/main i386 Pac
```

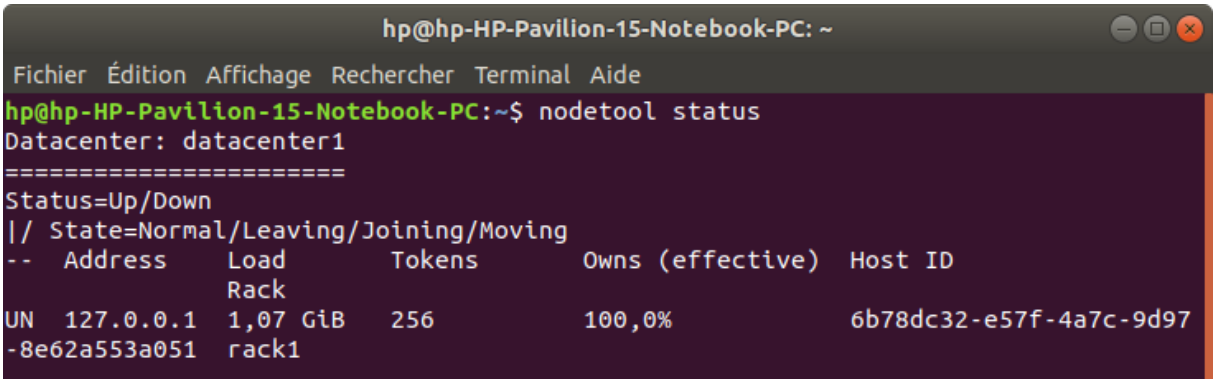
- **Etape 4 :** installer le paquet **Cassandra** et lancer **le service Cassandra**



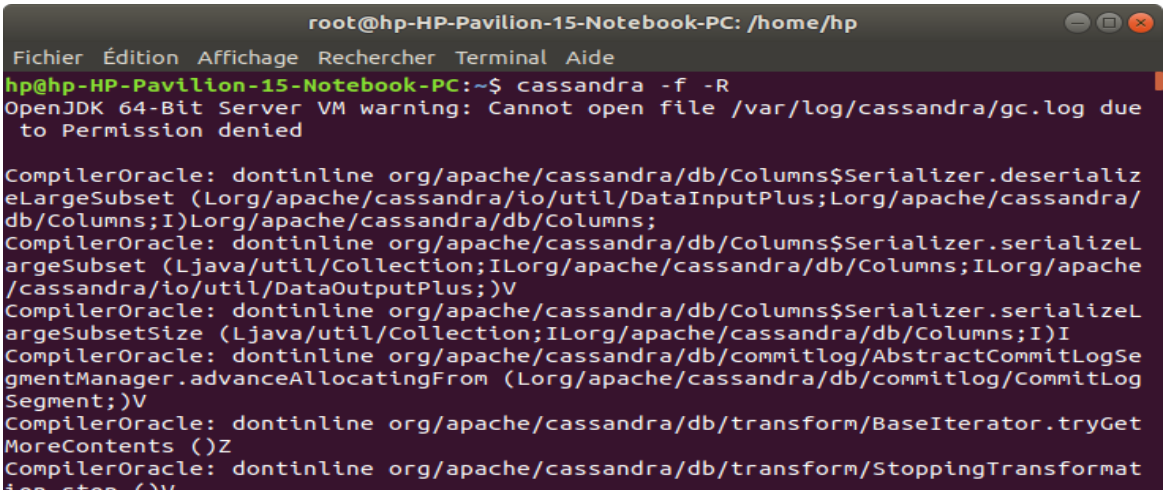

- **Etape 5 :** il faut modifier : l'administrateur (**root**) par la commande **sudo su** pour lancer **le serveur Cassandra** et on tape la commande **cassandra –f –R**



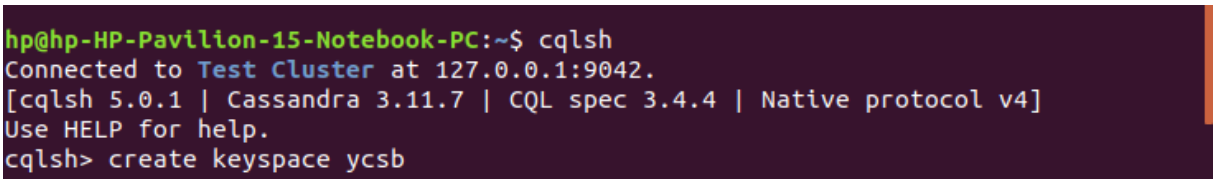

- **Etape 6 :** dans un autre cmd on Exécuter la commande **nodetool status**




- **Etape 7 :** Activer le server cassandra par la commande : **cassandra –f –R**




- **Etape 8 :** Activer le cassandra cqlsh par la commande : **cqlsh**




### 4.1.5. Configuration de CYSB v0.18.0

- **Etape 1 :** commencez par mettre à jour la liste des packages pour avoir la version la plus récente des listes référentiels par **sudo apt update**

```
hp@hp-HP-Pavilion-15-Notebook-PC: ~
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
hp@hp-HP-Pavilion-15-Notebook-PC:~$ sudo apt update
[sudo] Mot de passe de hp :
Réception de :1 http://ppa.launchpad.net/openjdk-r/ppa/ubuntu bionic InRelease [
15.4 kB]
Réception de :2 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.
7 kB]
Atteint :3 http://dz.archive.ubuntu.com/ubuntu bionic InRelease
Réception de :4 http://dz.archive.ubuntu.com/ubuntu bionic-updates InRelease [88
.7 kB]
Atteint :6 http://ppa.launchpad.net/webupd8team/java/ubuntu bionic InRelease
Réception de :8 http://dz.archive.ubuntu.com/ubuntu bionic-backports InRelease [
74.6 kB]
Réception de :9 http://ppa.launchpad.net/openjdk-r/ppa/ubuntu bionic/main amd64
Packages [9,568 B]
Réception de :10 http://security.ubuntu.com/ubuntu bionic-security/main i386 Pac
kages [510 kB]
Réception de :11 http://dz.archive.ubuntu.com/ubuntu bionic-updates/main i386 Pa
ckages [719 kB]
Réception de :5 https://dl.bintray.com/apache/cassandra 311x InRelease [3,183 B]
Réception de :12 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Pa
ckages [796 kB]
Réception de :7 https://dl.bintray.com/apache/cassandra 39x InRelease [3,168 B]
Réception de :13 http://ppa.launchpad.net/openjdk-r/ppa/ubuntu bionic/main i386
Packages [9,540 B]
```

- **Etape 2 :** installez le paquet **git**

```
hp@hp-HP-Pavilion-15-Notebook-PC: ~/YCSB
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
hp@hp-HP-Pavilion-15-Notebook-PC:~$ gitclone http://github.com/brianfrankcooper/YCSB.git
gitclone : commande introuvable
hp@hp-HP-Pavilion-15-Notebook-PC:~$ sudo apt install git
[sudo] Mot de passe de hp :
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
Les paquets supplémentaires suivants seront installés :
  git-man liberror-perl
Paquets suggérés :
  git-daemon-run | git-daemon-sysvinit git-doc git-el git-email git-gui gitk
  gitweb git-cvs git-mediawiki git-svn
Les NOUVEAUX paquets suivants seront installés :
  git git-man liberror-perl
0 mis à jour, 3 nouvellement installés, 0 à enlever et 3 non mis à jour.
Il est nécessaire de prendre 4,741 ko dans les archives.
Après cette opération, 34.0 Mo d'espace disque supplémentaires seront utilisés.
Souhaitez-vous continuer ? [O/n] o
Réception de :1 http://dz.archive.ubuntu.com/ubuntu bionic/main amd64 liberror-perl all 0.17025-1 [22.8 kB]
Réception de :2 http://dz.archive.ubuntu.com/ubuntu bionic-updates/main amd64 git-man all 1:2.17.1-1ubuntu0.7 [804 kB]
Réception de :3 http://dz.archive.ubuntu.com/ubuntu bionic-updates/main amd64 git amd64 1:2.17.1-1ubuntu0.7 [3,915 kB]
4,741 ko réceptionnés en 19s (245 ko/s)
Sélection du paquet liberror-perl précédemment désélectionné.
(Lecture de la base de données... 171924 fichiers et répertoires déjà installés.)
Préparation du dépaquetage de .../liberror-perl_0.17025-1_all.deb ...
Dépaquetage de liberror-perl (0.17025-1) ...
Sélection du paquet git-man précédemment désélectionné.
Préparation du dépaquetage de .../git-man_1%3a2.17.1-1ubuntu0.7_all.deb ...
```

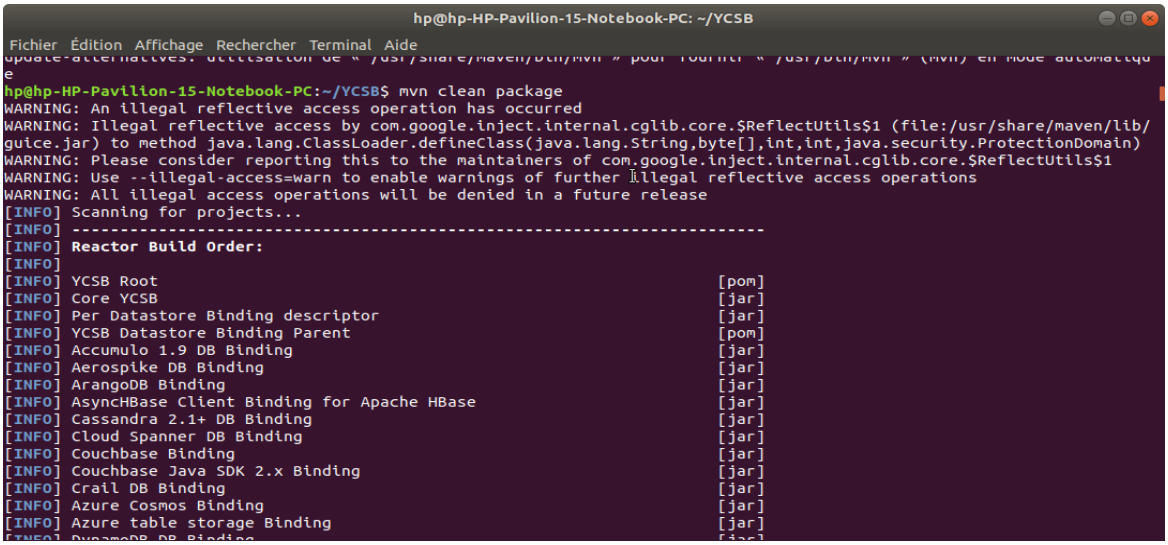
- **Etape 3 :** télécharger la bd **YCSB** par la commande suivant :

```
hp@hp-HP-Pavilion-15-Notebook-PC:~$ git clone http://github.com/brianfrankcooper/YCSB.git
Clonage dans 'YCSB'...
warning: redirection vers https://github.com/brianfrankcooper/YCSB.git/
remote: Enumerating objects: 20365, done.
remote: Total 20365 (delta 0), reused 0 (delta 0), pack-reused 20365
Réception d'objets: 100% (20365/20365), 31.52 MiB | 128.00 KiB/s, fait.
Résolution des deltas: 100% (7950/7950), fait.
```

- **Etape 4 :** installez le paquet **maven** dans le répertoire YCSB

```
hp@hp-HP-Pavilion-15-Notebook-PC:~/YCSB$ sudo apt install maven
[sudo] Mot de passe de hp :
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
Les paquets supplémentaires suivants seront installés :
  libaopalliance-java libapache-pom-java libatinject-jsr330-api-java
  libcdi-api-java libcommons-cli-java libcommons-io-java libcommons-lang3-java
  libcommons-parent-java libgeronimo-annotation-1.3-spec-java
  libgeronimo-interceptor-3.0-spec-java libguava-java libguice-java
  libhawtjni-runtime-java libjansi-java libjansi-native-java libjsr305-java
  libmaven-parent-java libmaven-resolver-java libmaven-shared-utils-java
  libmaven3-core-java libplexus-cipher-java libplexus-classworlds-java
  libplexus-component-annotations-java libplexus-interpolation-java
```

- **Etape 5 :** accéder au YCSB et compiler avec la commande : **mvn clean package**




## 4.2. Deuxième partie : Tests et analyse des résultats

### 4.2.1. Evaluation de performance

Dans ce chapitre, nous allons présenté et analysé les résultats de chargement de 800 000 enregistrements générés par YCSB puis on va effectuer différents tests sous forme de charges de travail (Workloads) composées de 100 000 opérations. Le critère de performance principal sur lequel les comparaisons vont être faites, est le temps d'exécution, qui sera estimé par la moyenne des temps d'exécutions des Workloads obtenus pendant 3 journées différentes. Les 100 000 opérations seront réparties entre lecture, écriture, et mises à jour effectuées sur ces enregistrements.

### 4.2.2. Initialisation et présentation des Wokloads

Nous avons commencé par initialiser l'outil YCSB avec 6 Workloads afin de rendre les scénarios plus significatifs :

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

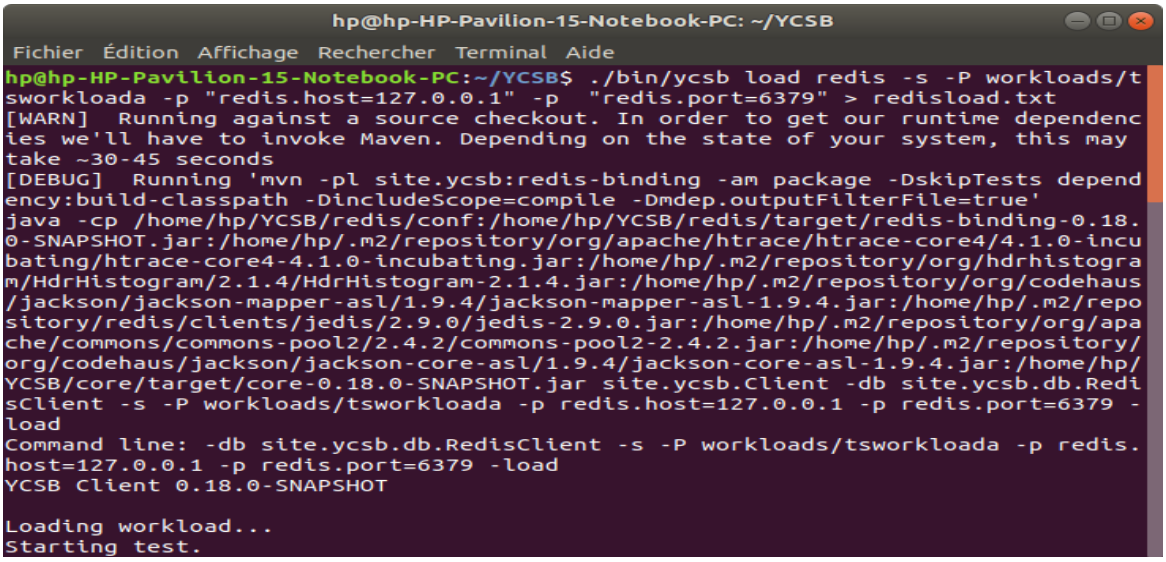
### 4.2.3. Chargement de données (LoadProcess)

Nous commençons par l'étape de chargement de 800 000 enregistrements générés par YCSB (Load Process), on modifier le nombre d'enregistrements et le nombre d'opérations dans le fichier **workload A** dans le répertoire **YCSB.**

- **Redis**

Taper la commande suivante après ouvrir le répertoire YCSB dans le terminal :

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

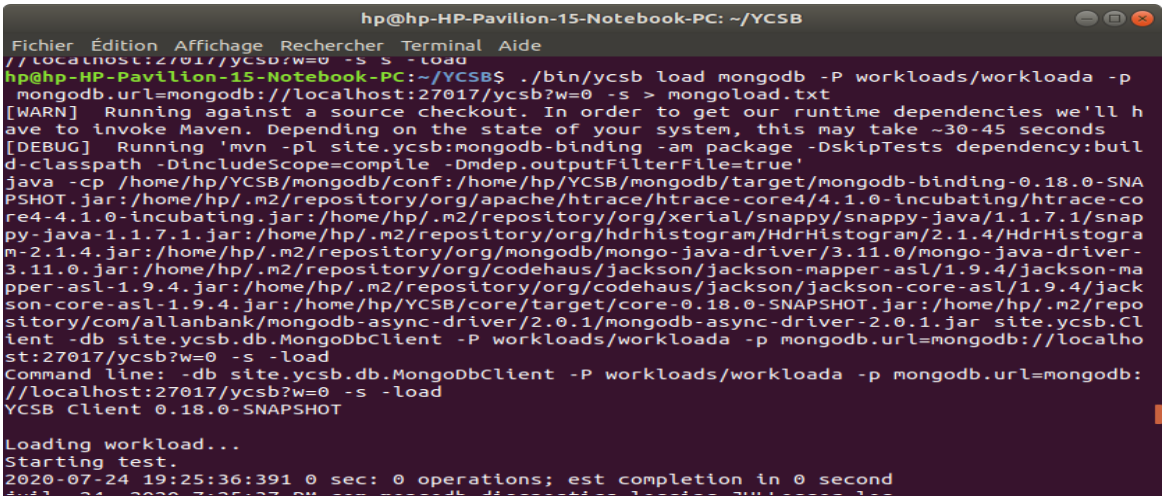



Le résultat du test est le suivant dans le fichier text **redisload.txt**:


- **Mongodb**

Taper la commande suivante après ouvrir le répertoire YCSB dans le terminal :

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

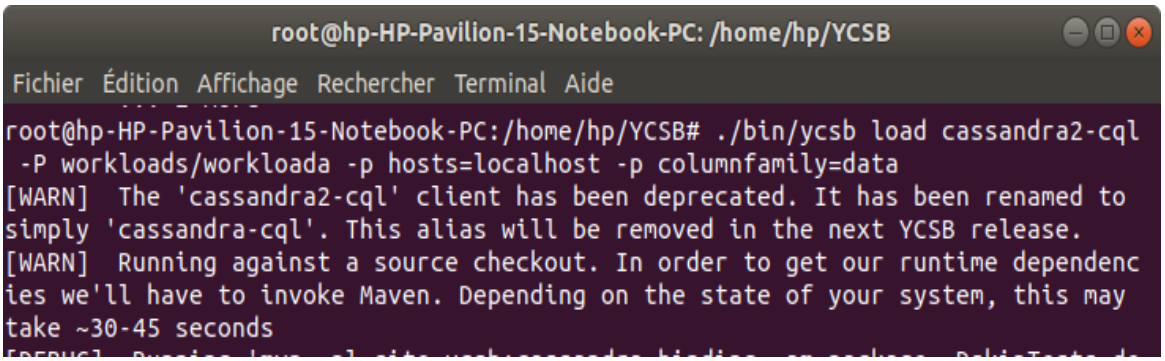



Le résultat du test est le suivant dans le fichier text **mongoload.txt**:


## ➢ Cassandra

Taper la commande suivante après ouvrir le répertoire YCSB dans le terminal :

| .bin/ycsb load cassandra2-cql -P workloads/workloada –p hosts=’127.0.0.1’ –p culumnfamily=data |
|---|




Le résultat du test est le suivant dans le terminal:

```
hp@hp-HP-Pavilion-15-Notebook-PC: ~/YCSB
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
Command line: -db site.ycsb.db.CassandraCQLClient -P workloads/workloada -p hosts=127.0.0.1 -p columnfamily=data-s -load
YCSB Client 0.18.0-SNAPSHOT

Loading workload...
Starting test.
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
DBWrapper: report latency for each error is false and specific error codes to track for latency are: []


[OVERALL], RunTime(ms), 376137
[OVERALL], Throughput(ops/sec), 2126.884619168016
[TOTAL_GCS_PS_Scavenge], Count, 775
[TOTAL_GC_TIME_PS_Scavenge], Time(ms), 1292
[TOTAL_GC_TIME_%_PS_Scavenge], Time(%), 0.3434918659956346
[TOTAL_GCS_PS_MarkSweep], Count, 0
[TOTAL_GC_TIME_PS_MarkSweep], Time(ms), 0
[TOTAL_GC_TIME_%_PS_MarkSweep], Time(%), 0.0
[TOTAL_GCs], Count, 775
[TOTAL_GC_TIME], Time(ms), 1292
[TOTAL_GC_TIME_%], Time(%), 0.3434918659956346
[CLEANUP], Operations, 1
[CLEANUP], AverageLatency(us), 2321408.0
[CLEANUP], MinLatency(us), 2320384
[CLEANUP], MaxLatency(us), 2322431
[CLEANUP], 95thPercentileLatency(us), 2322431
[CLEANUP], 99thPercentileLatency(us), 2322431
[INSERT], Operations, 800000
[INSERT], AverageLatency(us), 458.5980025
[INSERT], MinLatency(us), 208
[INSERT], MaxLatency(us), 170367
[INSERT], 95thPercentileLatency(us), 556
[INSERT], 99thPercentileLatency(us), 737
[INSERT], Return=OK, 800000
hp@hp-HP-Pavilion-15-Notebook-PC:~/YCSB$
hp@hp-HP-Pavilion-15-Notebook-PC:~/YCSB$
hp@hp-HP-Pavilion-15-Notebook-PC:~/YCSB$ ./bin/ycsb load  cassandra-cql -P workloads/workloada -p hosts='127.0.0.1' -p columnfamily=data-s
```

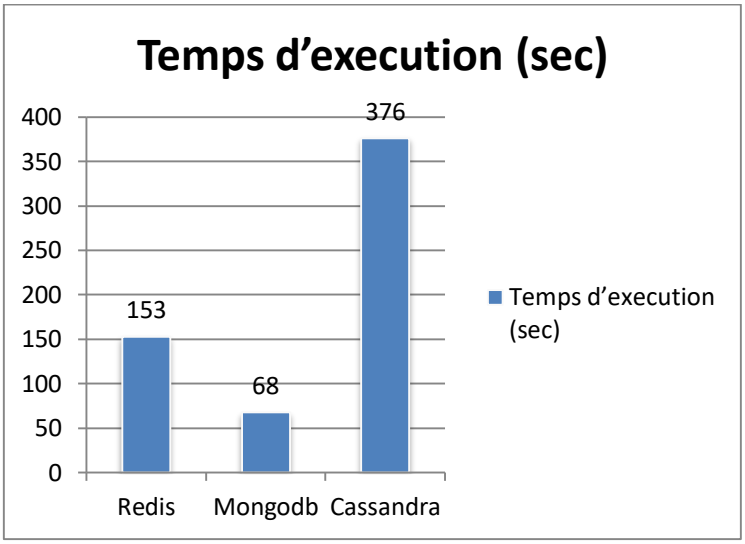



**Figure IV.1 : Histogramme de Temps de chargement**

| SGBDs Nosql | Temps d'execution (sec) |
|---|---|
| Redis | 153 |
| Mongodb | 68 |
| Cassandra | 376 |

**Tableau IV.2 : Temps de chargement**

La Figure IV.1 montre le temps d'exécution pendant le chargement de 800 000 enregistrements dans les trois bases de données. Nous avons constaté que durant le chargement de 800 000 enregistrements, le meilleurs temps d'insertion a été fournit par MongoDB avec un temps de chargement de **68 sec**. Cela signifie que MongoDB était plus rapide que Cassandra et Redis pendant la phase de chargement. La cause est le fait que MongoDB ne nécessite pas de grande quantité de mémoire pendant l'exécution des opérations de chargement initial.

### 4.2.4. Exécution des Workloads

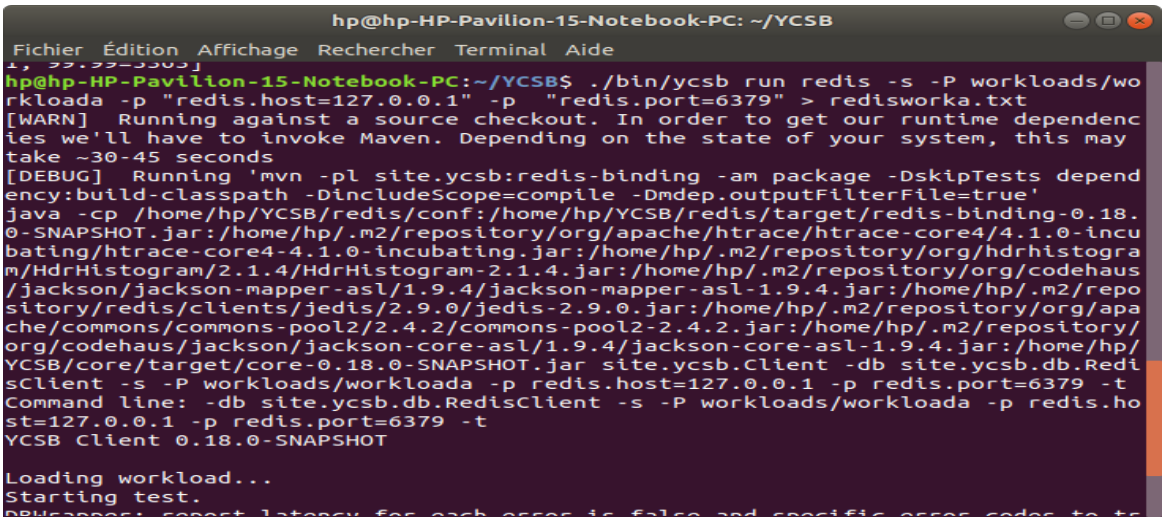
Dans la section suivante, on va présenter et analyser les résultats des tests. Un rapprochement de temps d'exécution des Workloads effectuées sur 800 000 enregistrements, sera établi. Le nombre d'opérations est limité à 100 000 opérations, on modifier le nombre d'enregistrements et le nombre d'opérations dans le fichier **workload A, workload B, workload C ; workload D, workload E, workload F** dans le répertoire **YCSB.**

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

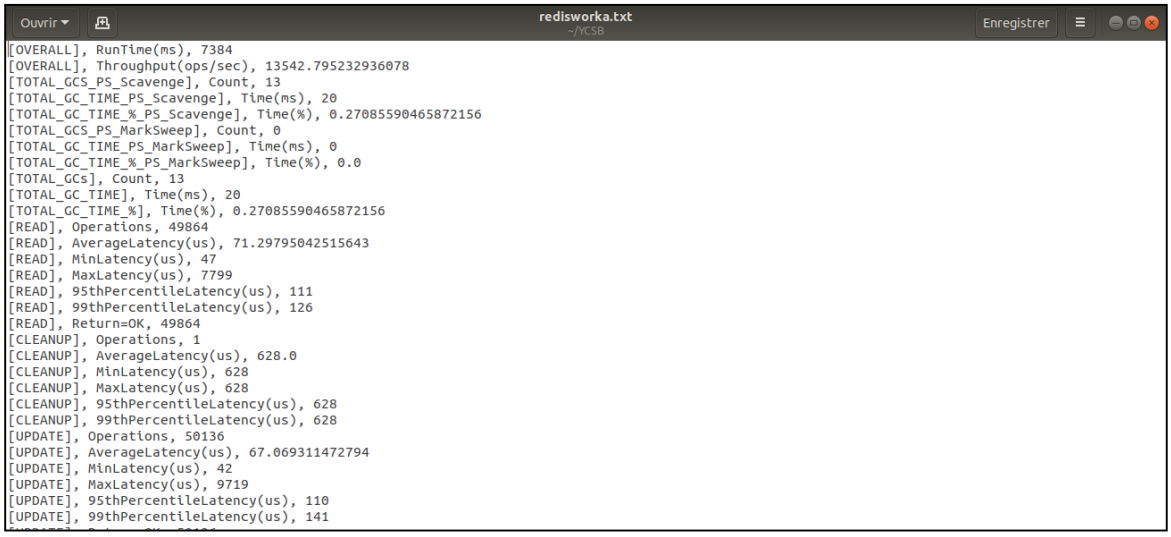
- **Test de Redis : Taper la commande suivante :**

```
. bin/ycsb run  redis –s  -P  workloads/workloada  –p  "redis.port=6379" > redisworka.txt
```

Le résultat du test est le suivant :



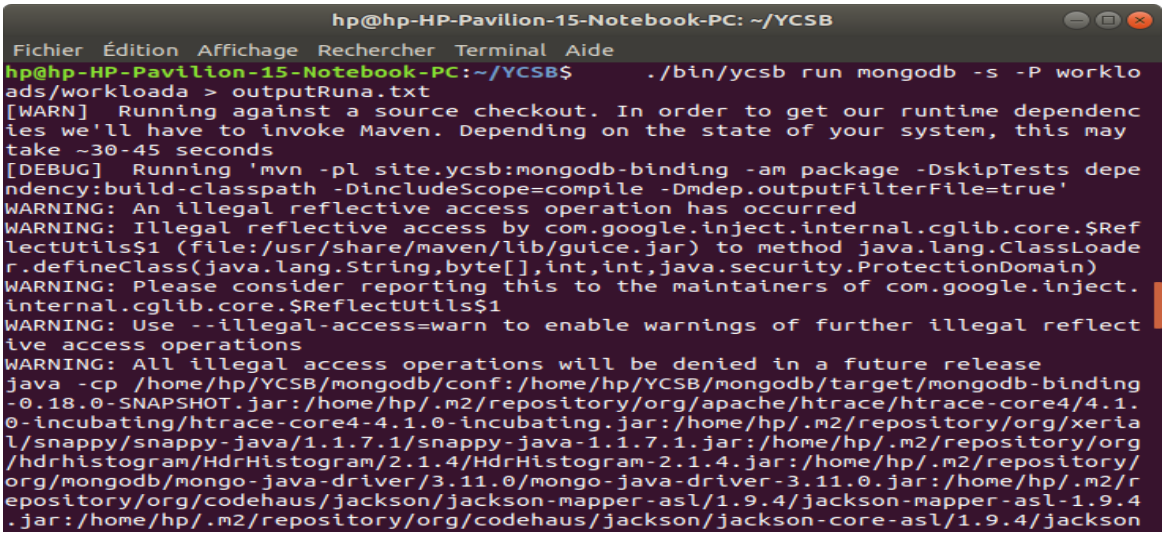

Le journal d'exécution du Workload A par Redis :



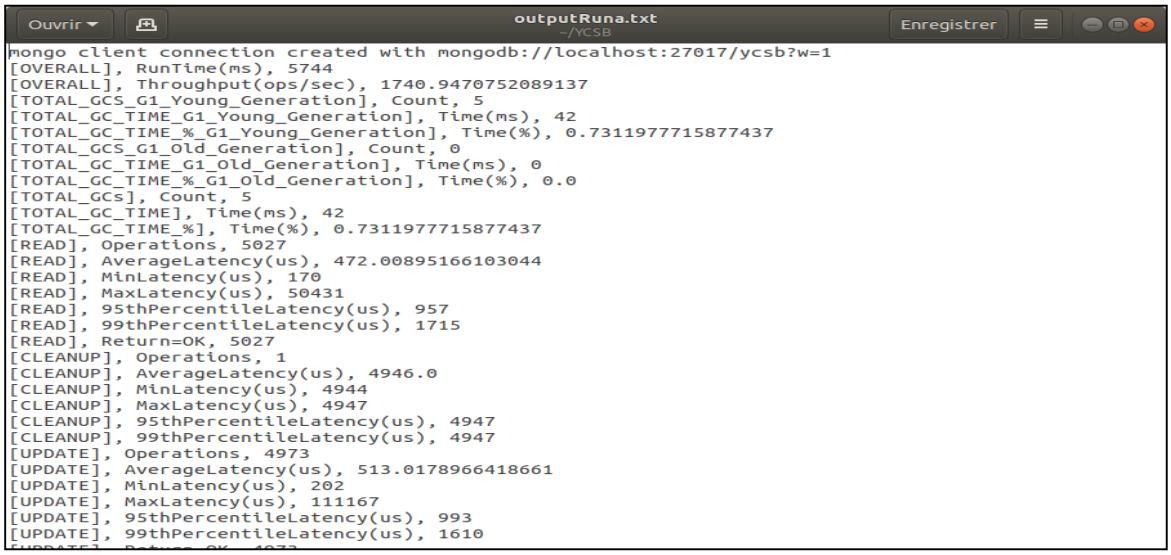
- **Test de Mongodb : Taper la commande suivante :**

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

Le résultat du test est le suivant :



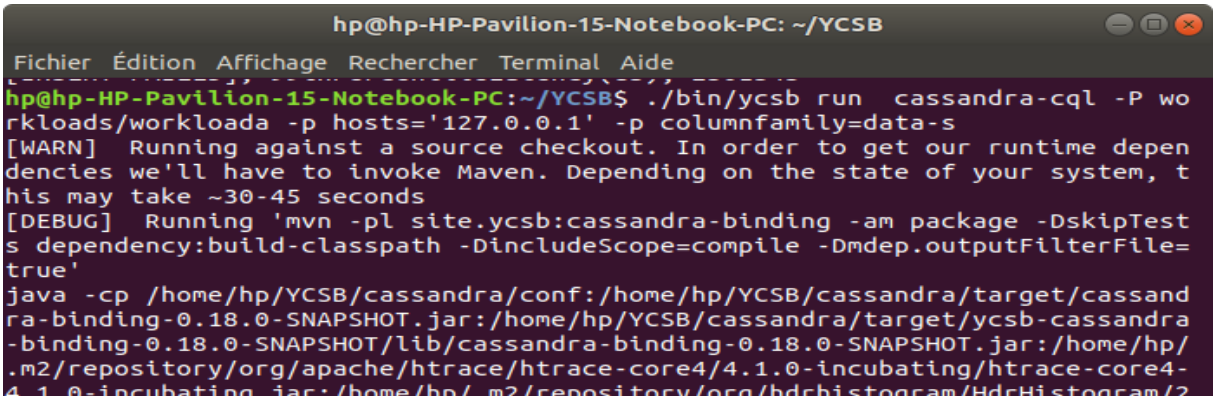

Le journal d'exécution du Workload A par Mongodb :



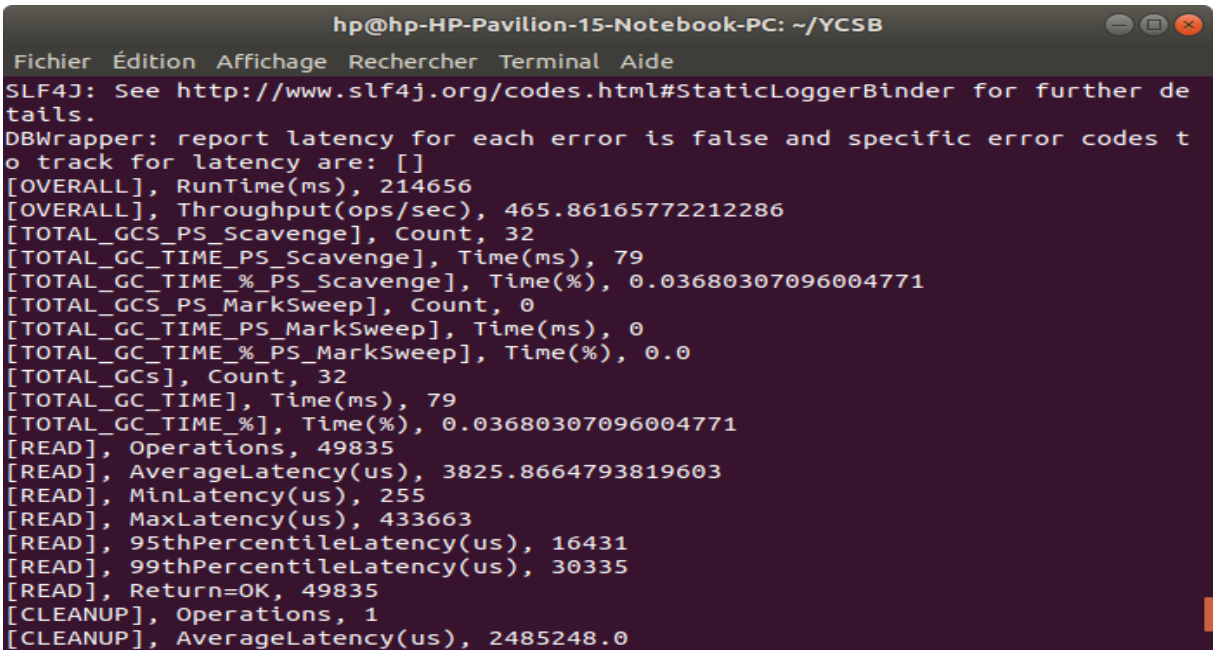

- **Test de Cassandra : Taper la commande suivante :**

```
.bin/ycsb run cassandra2-cql -P workloads/workloada –p hosts='127.0.0.1' –p columnfamily=data
```

Le résultat du test est le suivant :




Le journal d'exécution du Workload A par Cassandra :




Le Tableau suivant représente le temps d'exécution de workload A :

| SGBDs Nosql | Temps d'exécution du workload A (sec) | | | |
|---|---|---|---|---|
| | 1er jour | 2eme jour | 3eme jour | Temps moyenne |
| Redis | 8.253 sec | 11.940 sec | 10.302 sec | 10.165 sec |
| Mongodb | 79.630 sec | 72.471 sec | 43.196 sec | 65.099 sec |
| Cassandra | 214.656 sec | 106.706 sec | 231.692 sec | 184.153 sec |

**Tableau IV.3 : le temps d'exécution de workload A**

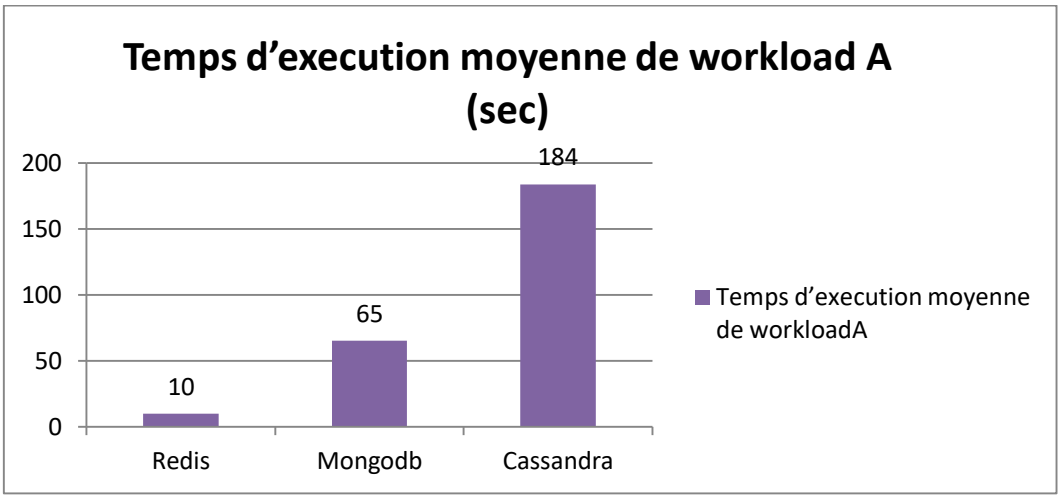



**Figure IV.2. Histogramme de Temps d'exécution moyenne du Workload A**

La Figure IV.2 montre les résultats obtenus durant l'exécution du Workload A composé de 50% d'opérations Read et 50% Update de 100 000 opérations, effectuées sur 800 000 enregistrements. Après la lecture des résultats obtenus, nous remarquons que les bonnes performances sont présentées en premier lieu par la catégorie orientée Clé-valeur où Redis avec 10 secondes était la plus rapide. En deuxième lieu on retrouve la catégorie orientée document MongoDB avec 65 secondes et finalement la catégorie orientée colonne Cassandra avec 184 secondes. Les bases de données clé-valeur sont les plus performantes relativement aux précédentes. En effet, il faut voir les résultats des charges pour favoriser une solution NoSQL par rapport aux autres.

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

- **Test de Redis :** Taper la commande suivante :

```
.bin/ycsb run redis –s -P workloads/workloadb –p "redis.port=6379" > redisworkb.txt
```

- **Test de MongoDB :** Taper la commande suivante :

```
.bin/ycsb run mongodb –s -P workloads/workloadb >outputRunb.txt
```

- **Test de Cassandra :** Taper la commande suivante :

```
.bin/ycsb run cassandra2-cql -P workloads/workloadb –p hosts='127.0.0.1' –p culumnfamily=data
```

Le Tableau suivant représente le temps d'exécution de workload B :

| SGBDs Nosql | Temps d'exécution du workloadB (sec) | | | |
|---|---|---|---|---|
| | 1er jour | 2eme jour | 3eme jour | Temps moyenne |
| Redis | 5.838 sec | 9.308 sec | 6.929 sec | 7.358 sec |
| Mongodb | 21.796 sec | 39.088 sec | 21.847 sec | 27.577 sec |
| Cassandra | 51.929 sec | 78.996 sec | 160.216 sec | 97.047 sec |

**Tableau IV.4 : le temps d'exécution de workload B**

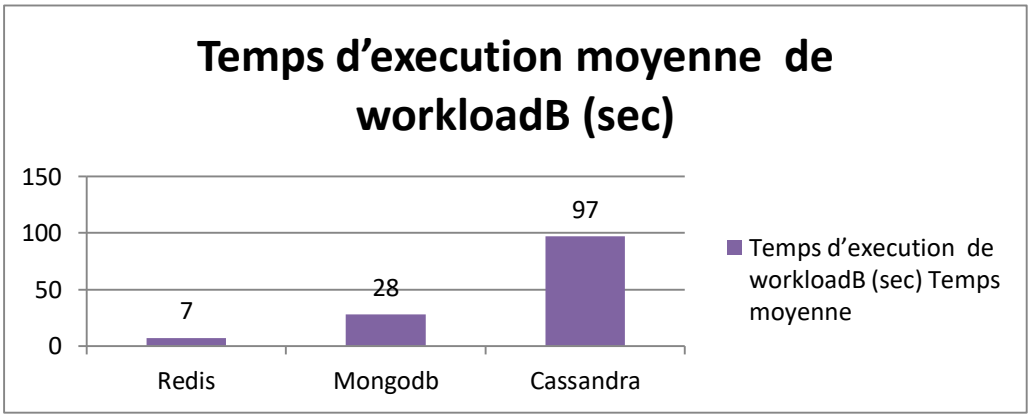



**Figure IV.3. Histogramme de Temps d'exécution moyenne du Workload B**

Les résultats de la figure IV.3 prouve l'ascendance une autre fois des modèles orientés clé-valeur Redis 7 secondes pour les charges composées principalement d'opérations de lecture (**95%Read)**. En revanche, ceux des ont prouvé leur performance par rapport aux modèles orientés document MongoDB et colonnes Cassandra.

## 3. Workload C (100% Read)

- **Test de Redis :** Taper la commande suivante :

```
.bin/ycsb run  redis –s  -P  workloads/workloadc –p  "redis.port=6379" > redisworkc.txt
```

- **Test de Mongodb :** Taper la commande suivante :

```
.bin/ycsb run mongodb –s -P workloads/workloadc >outputRunc.txt
```

- **Test de Cassandra :** Taper la commande suivante :

```
.bin/ycsb run cassandra2-cql -P workloads/workloadc –p hosts=’127.0.0.1’ –p culumnfamily=data
```

Le Tableau suivant représente le temps d’exécution de workload C :

| SGBDs Nosql | Temps d’exécution du workload C (sec) | | | |
|---|---|---|---|---|
| | 1er jour | 2eme jour | 3eme jour | Temps moyenne |
| Redis | 5.230 sec | 8.296 sec | 6.811 sec | 6.779 sec |
| Mongodb | 20.472 sec | 28.817 sec | 20.175 sec | 23.155 sec |
| Cassandra | 47.057 sec | 73.250 sec | 105.114 sec | 75.140 sec |

**Tableau IV.5 : le temps d’exécution de workload C**

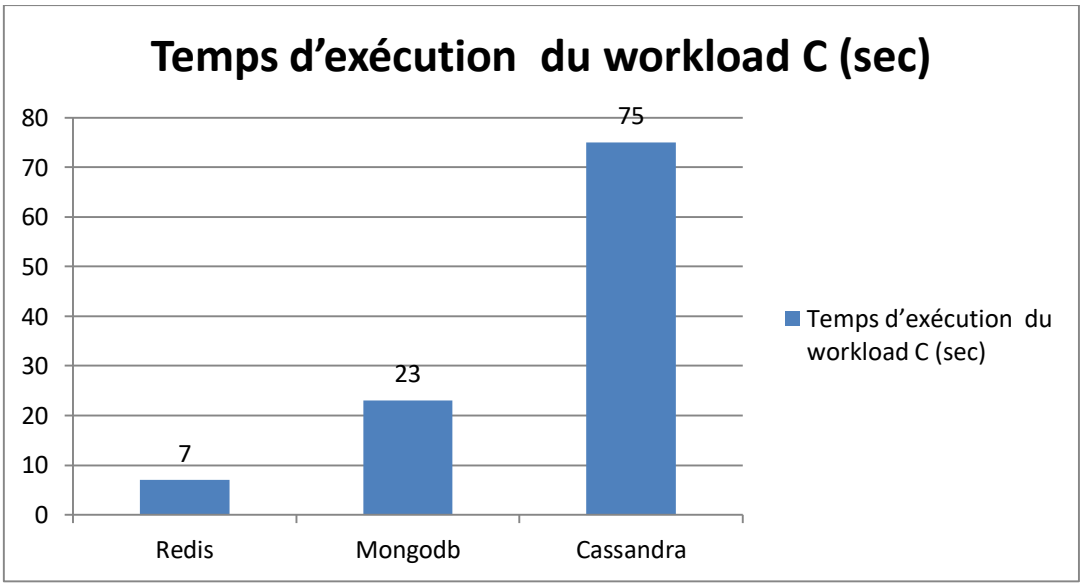



**Figure IV.4**. Histogramme Temps d’exécution moyenne du Workload C

La Figure IV.4. Montre les résultats obtenus compose de 100% d’opérations Read, Nous remarquons que Redis 7 secondes est plus rapide par rapport à MongoDB 23 sec et Cassandra 75 sec.

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

- **Test de Redis :** Taper la commande suivante :

```
.bin/ycsb run  redis –s  -P  workloads/workloadd –p  “redis.port=6379” > redisworkd.txt
```

- **Test de Mongodb :** Taper la commande suivante :

```
.bin/ycsb run mongodb –s -P workloads/workloadd >outputRund.txt
```

- **Test de Cassandra :** Taper la commande suivante :

```
.bin/ycsb run cassandra2-cql -P workloads/workloadd –p hosts=’127.0.0.1’ –p culumnfamily=data
```

Le Tableau suivant représente le temps d’exécution de workload D :

| SGBDs Nosql | Temps d’exécution du workload C (sec) | | | |
|---|---|---|---|---|
| | 1er jour | 2eme jour | 3eme jour | Temps moyenne |
| Redis | 6.273 sec | 9.025 sec | 7.460 sec | 7.586 sec |
| Mongodb | 22.277 sec | 35.679 sec | 26.437 sec | 28.131 sec |
| Cassandra | 50.377 sec | 60.379 sec | 66.032 sec | 58.929 sec |

**Tableau IV.6 : le temps d’exécution de workload D**

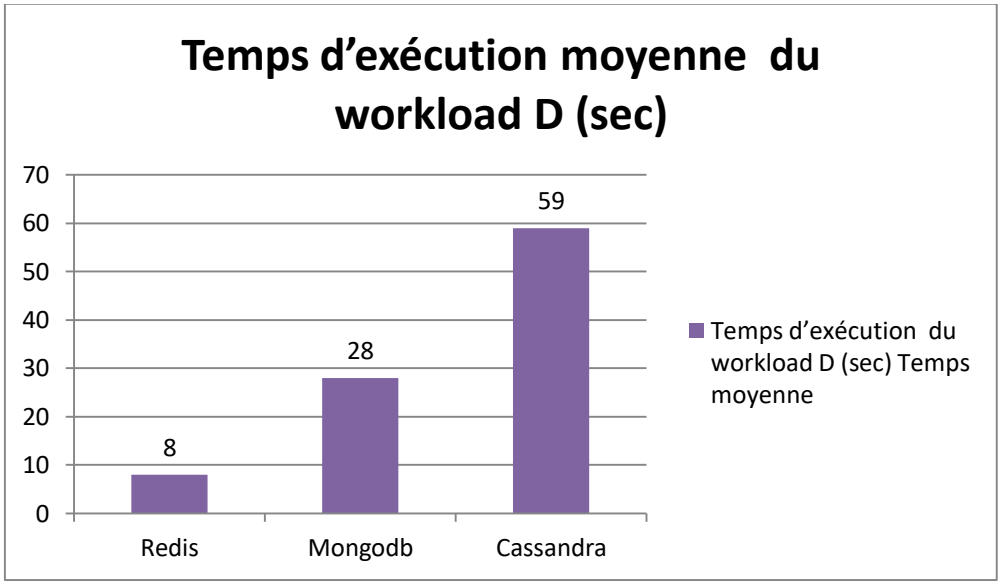



**Figure IV.5. Histogramme Temps d'exécution moyenne du Workload D**

La Figure IV.5. Montre les résultats obtenus compose de 95% d'opérations Read et 5% Insert, Nous remarquons que Redis et plus rapide par rapport à MongoDB et Cassandra.

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

- **Test de Redis :** Taper la commande suivante :

```
.bin/ycsb run  redis –s  -P  workloads/workloade –p  "redis.port=6379" > redisworke.txt
```

- **Test de Mongodb :** Taper la commande suivante :

```
.bin/ycsb run mongodb –s -P workloads/workloade >outputRune.txt
```

- **Test de Cassandra :** Taper la commande suivante :

```
.bin/ycsb run cassandra2-cql -P workloads/workloade –p hosts='127.0.0.1' –p culumnfamily=data
```

Le Tableau suivant représente le temps d'exécution de workload E :

| SGBDs Nosql | Temps d'exécution du workload C (sec) | | | |
|---|---|---|---|---|
| | 1er jour | 2eme jour | 3eme jour | Temps moyenne |
| Redis | 245.836 sec | 356.804 sec | 295.488 sec | 29.937 sec |
| Mongodb | 112.937 sec | 118.542 sec | 113.306 sec | 114.9283sec |
| Cassandra | 138.619 sec | 344.651 sec | 369.989 sec | 284.419 sec |

**Tableau IV.7 : le temps d'exécution de workload E**

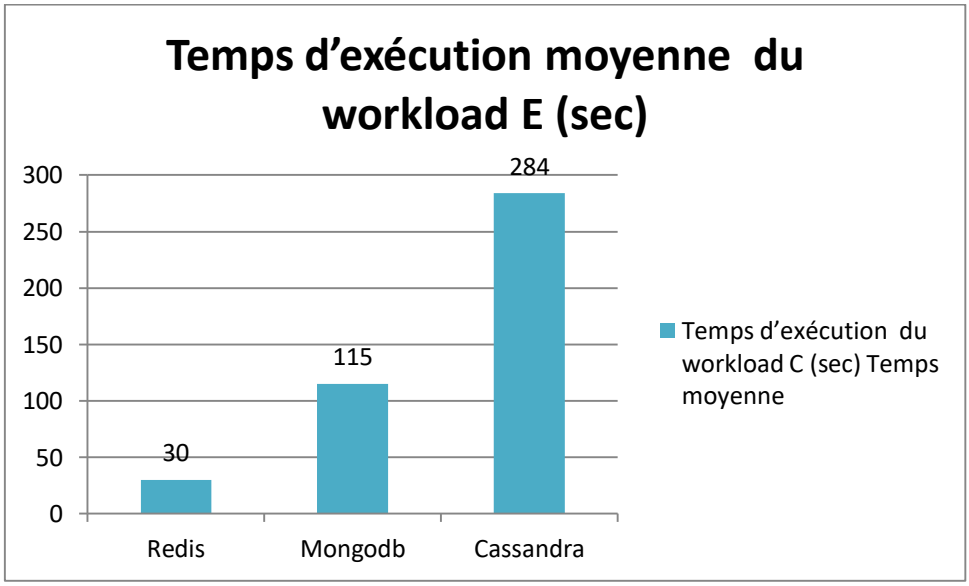



**Figure IV.6. Histogramme Temps d'exécution moyenne du Workload E**

La Figure IV.6. Montre les résultats obtenus compose de 95% Scan /5% Insert, Nous remarquons que toujours Redis et plus rapide par rapport à MongoDB et Cassandra.

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

- **Test de Redis :** Taper la commande suivante :

```
.bin/ycsb run  redis –s  -P  workloads/workloade –p  "redis.port=6379" > redisworke.txt
```

- **Test de Mongodb :** Taper la commande suivante :

```
.bin/ycsb run mongodb –s -P workloads/workloade >outputRune.txt
```

- **Test de Cassandra :** Taper la commande suivante :

```
.bin/ycsb run cassandra2-cql -P workloads/workloadf –p hosts='127.0.0.1' –p culumnfamily=data
```

Le Tableau suivant représente le temps d'exécution de workload F :

| SGBDs Nosql | Temps d'exécution du workload C (sec) | | | |
|---|---|---|---|---|
| | 1er jour | 2eme jour | 3eme jour | Temps moyenne |
| Redis | 8.058 sec | 15.308 sec | 10.985 sec | 114.503 sec |
| Mongodb | 41.362 sec | 48.361 sec | 35.443 sec | 41.722 sec |
| Cassandra | 194.116 sec | 110.267 sec | 107.593 sec | 125.666 sec |

**Tableau IV.8 : le temps d'exécution de workload F**

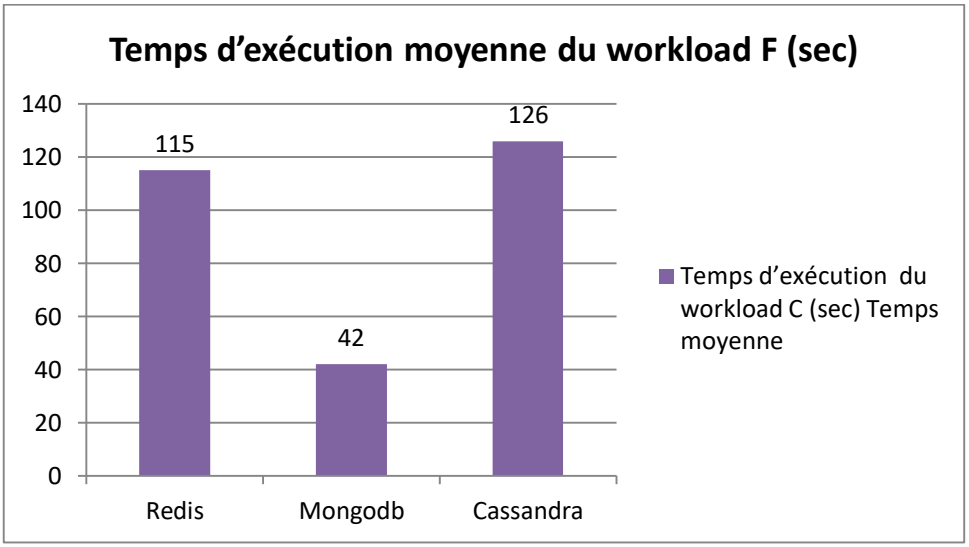



**Figure IV.7 : Histogramme Temps d'exécution moyenne du Workload F**

La Figure IV.7 Montre les résultats obtenus compose de 50% de Read et 50%Update, Nous remarquons que MongoDB et plus rapide par rapport à Redis et Cassandra.

La Figure IV.8 regroupe les temps d'exécutions de chargement des 800 000 enregistrements et ceux des Workloads composées de 10 000 opérations ainsi que le temps moyen global de tous les tests effectués.

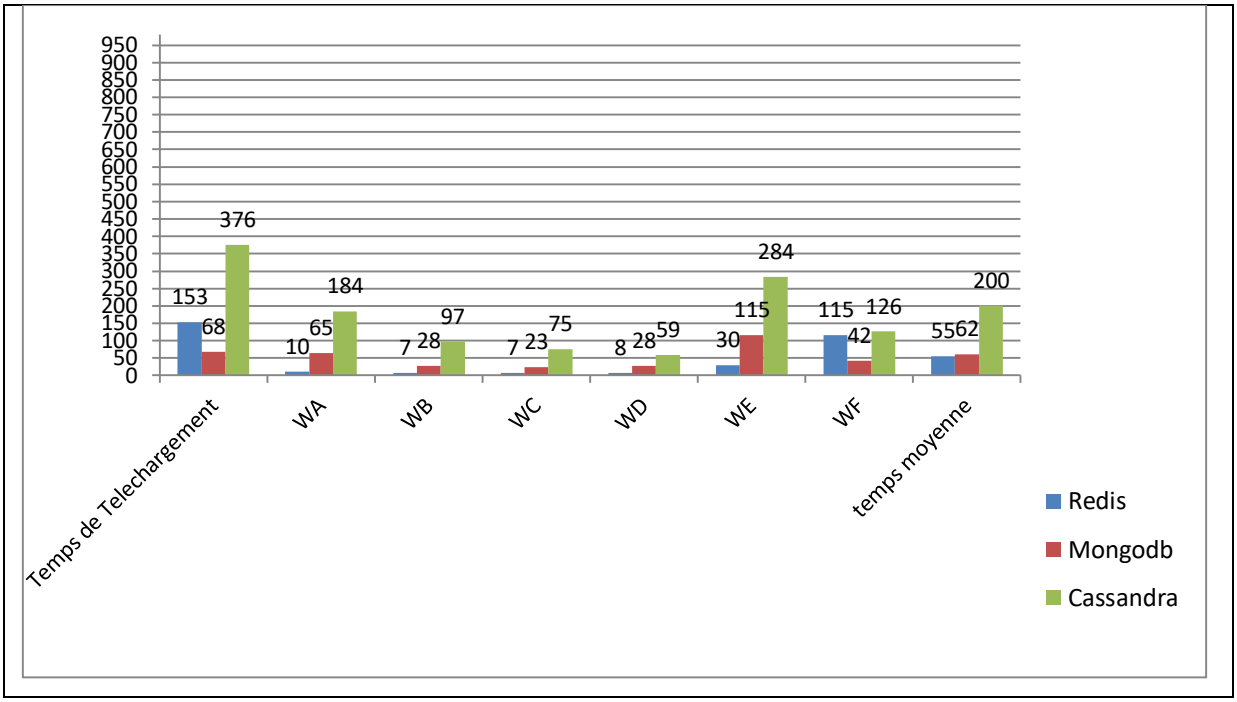



**Figure IV.8 : Histogramme globale de Temps de chargement et le temps d'exécution moyenne de toutes les Workloads**

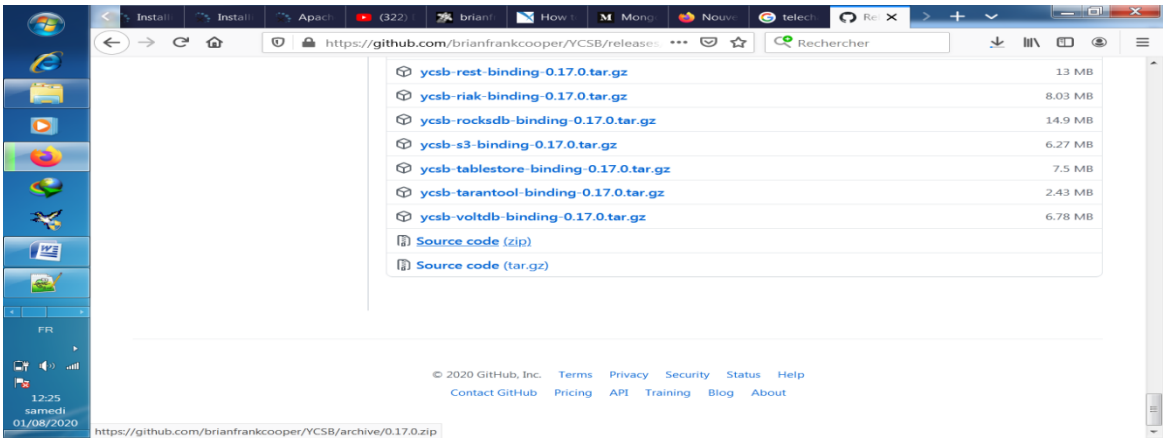
## 4.3. Evaluation globale

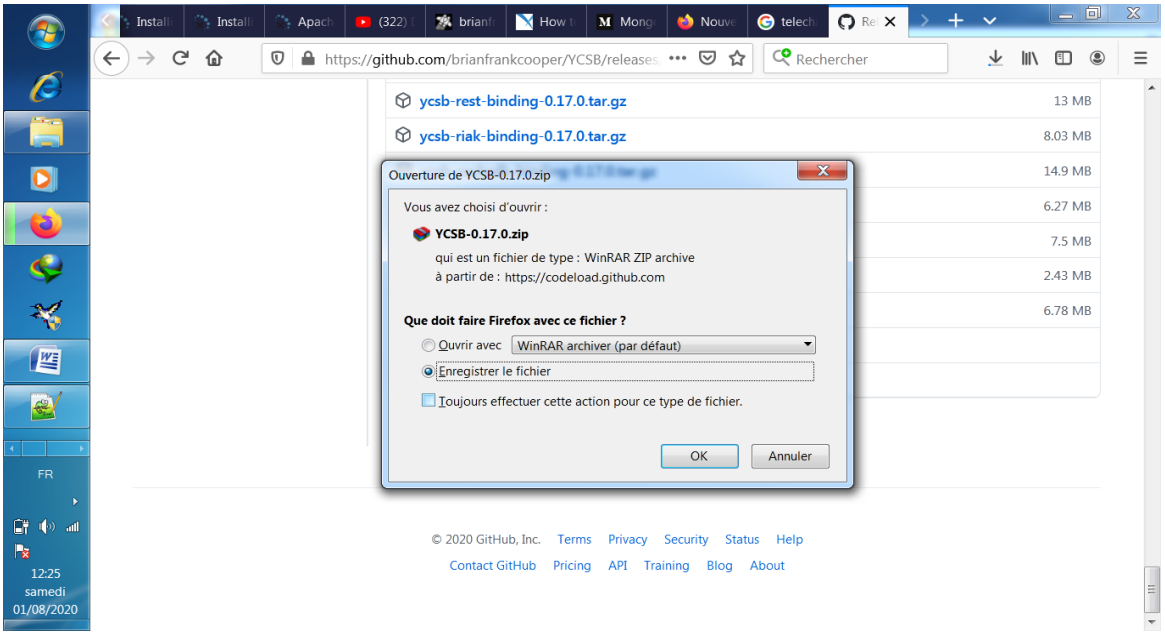
Après lecture des résultats obtenus par les trois systèmes, plusieurs enseignements peuvent être tirés :

- Redis a montré sa performance dans les opérations de lecture et mise à jour.
- La différence entre les temps d'exécutions durant le chargement des 800 000 enregistrements est justifiée par le fait que Redis ne nécessite pas un grand espace mémoire pour exécuter des opérations de chargement initial.
- Redis utilise la mémoire vivante RAM en plus utilise la mémoire virtuelle si le volume des données est intéressante ; un journal pour conserver toutes les modifications effectuées, ce qui accélère le processus de mise à jour.
- L'exécution d'une opération de mise à jour dans MongoDB, est ralentit relativement par rapport aux nombres de mises à jour appliquées. Elle utilise des mécanismes de verrouillage qui alourdissent le temps d'exécution de mise à jour.
- MongoDB a été plus performant dans les opérations de lecture et de scan par rapport Cassandra. Ceci est du, principalement, à la cartographie des registres de MongoDB chargée en mémoire qui améliore ces performances en lecture.

# 5. Configuration et Installation des BDs NoSql sous Windows10

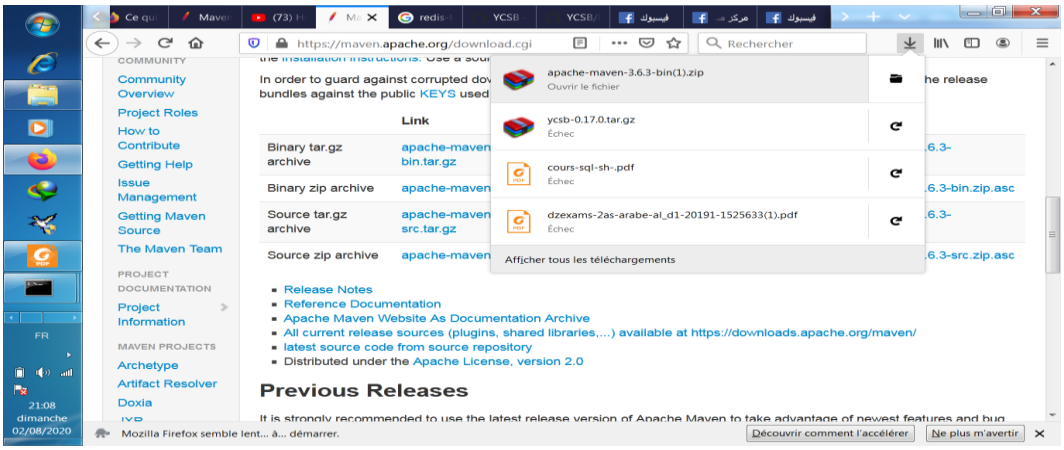
## 5.1. Configuration de CYSB v0.17.0.zip



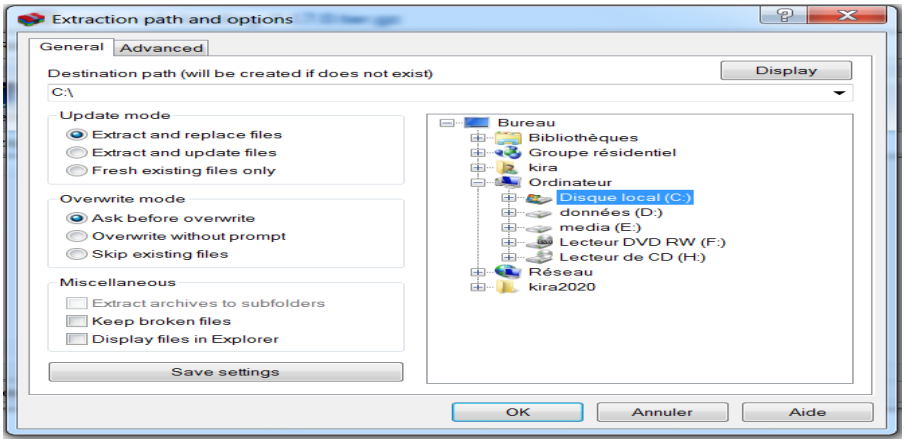




## 5.2. Configuration de maven3.6.3.bin.zip

- Etape 1 : Télécharger maven3.6.3.bin.zip **[Site 10]**



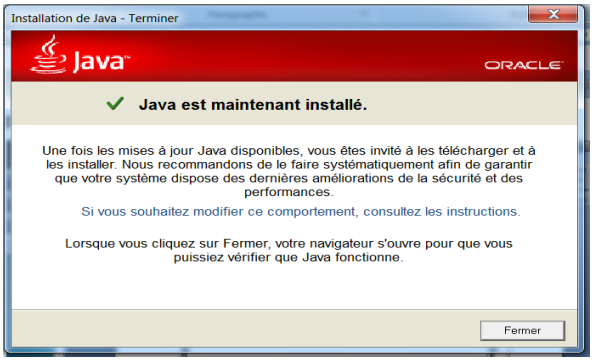
- Etape 2 : Extraire maven3.6.3.bin.zip et copier au c:\



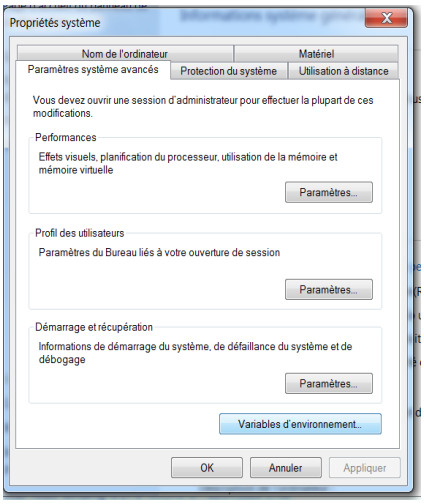

## 5.3. Configuration et Création des variables d'environnement

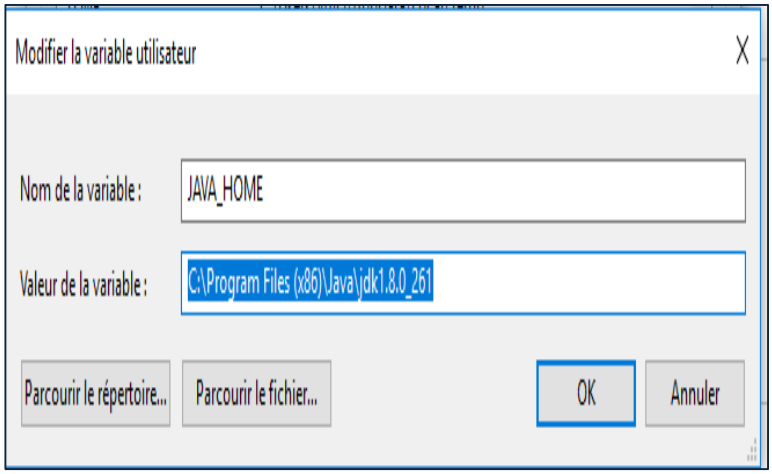
### 5.3.1. Les variables utilisateur

#### a- Créer le variable : JAVA_HOME

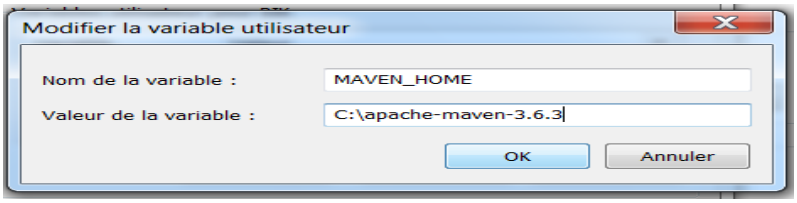



télécharger java 8 ou jdk 1.8.0_261 et installer

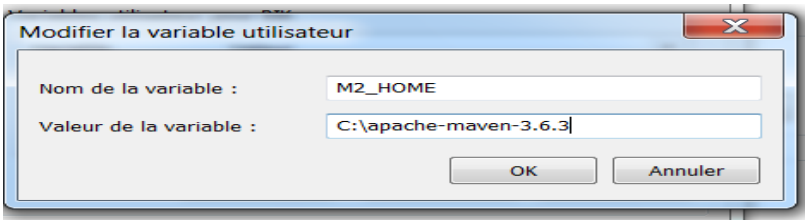


### b- Créer le variable : MAVEN_HOME

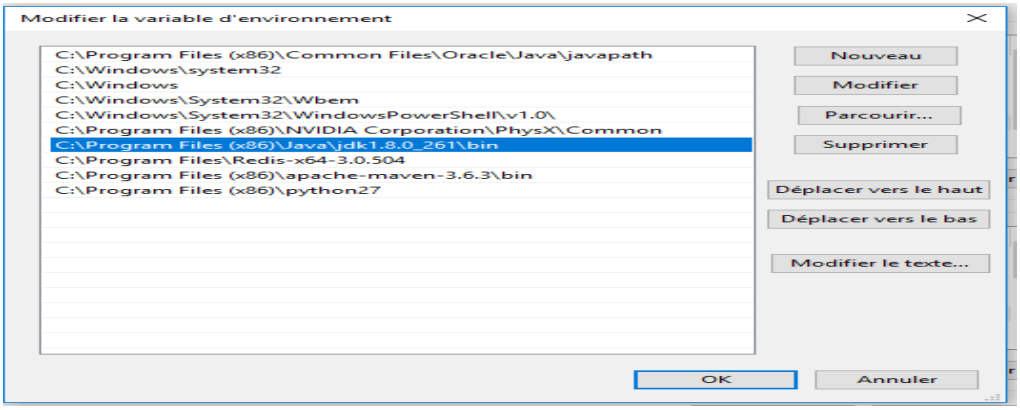



### c- Créer le variable : M2_HOME



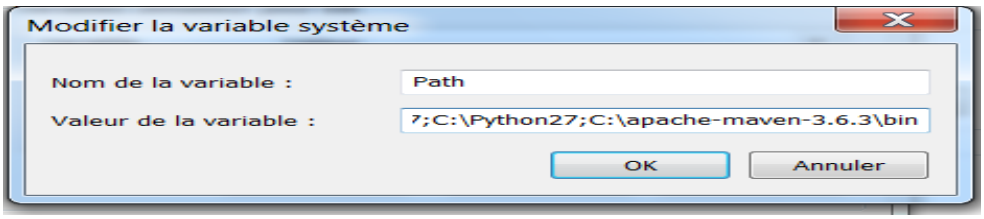

## 5.3.2. Les variables système

### a- Path Java

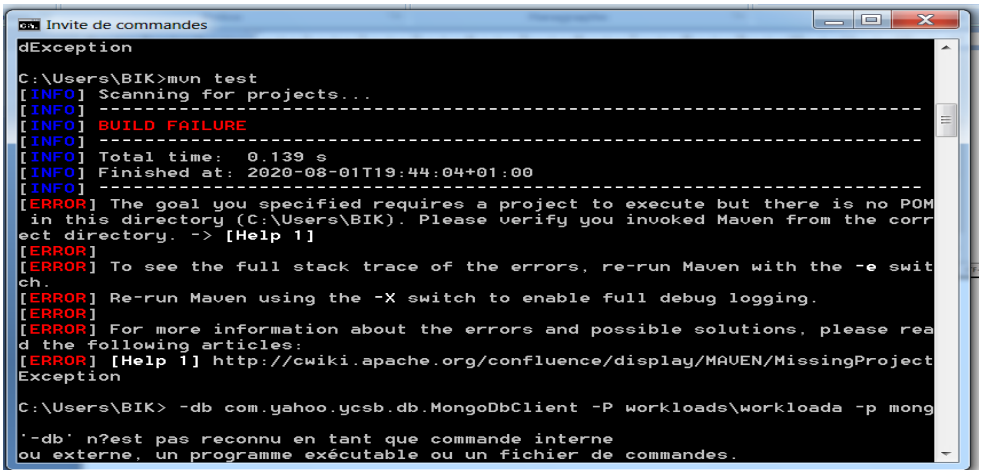



### b- Path Maven

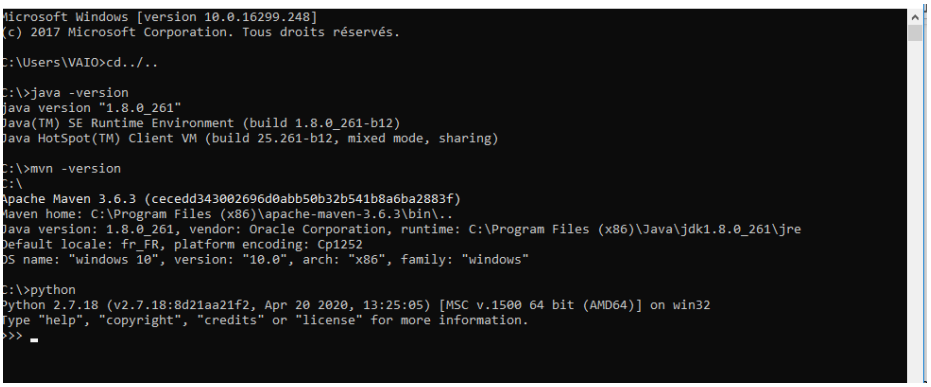
1. Tester **maven** installé et faire l'exécution




2. Taper sur cmd : **mvn test** ou **mvn** ou **mvn install**




3. Tester la version de java et maven et python
4. Taper sur cmd : **java –version** et **mvn –version** et **python-version**

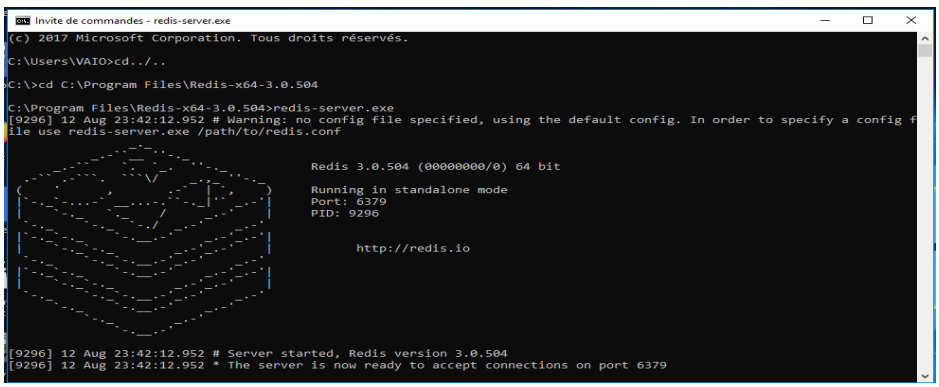



## 5.4. Installation de Redis v3.0.504

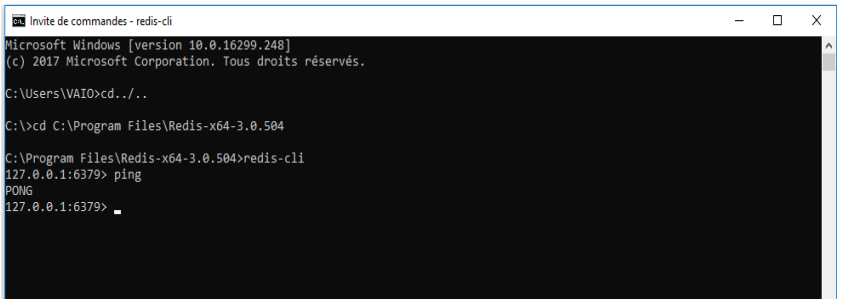
1- Télécharger le fichier zip **Redis 3.0.504.zip**. **[Site 11]**

2- Extraire le fichier zip dans le répertoire préparé (disque c ).

3- Exécuter redis-server.exe, vous pouvez soit exécuter directement en cliquant sur **redis-server.exe** ou exécuter via l'invite de commande.




4- Exécuter **redis-cli.exe** dans une autre invite de commandes

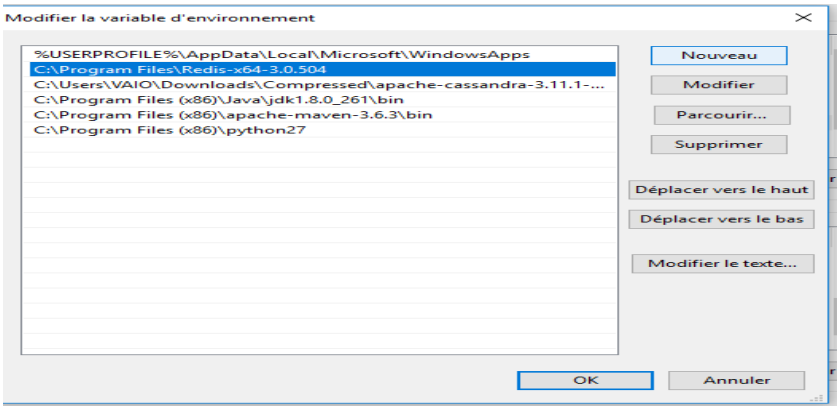



5- **PING** est utilisée pour tester si une connexion est toujours active.

```
127.0.0.1:6379> ping
PONG
127.0.0.1:6379> ping "hello world"
"hello world"
127.0.0.1:6379>
```

6- Vous pouvez maintenant commencer à utiliser Redis.

7- Ajouter Redis au path utilisateur et système

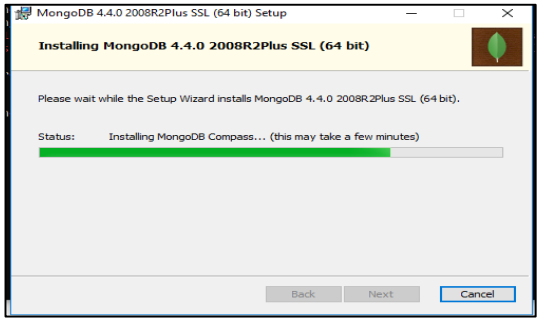



## 5.5. Installation Mongodb version 4.4.0 msi

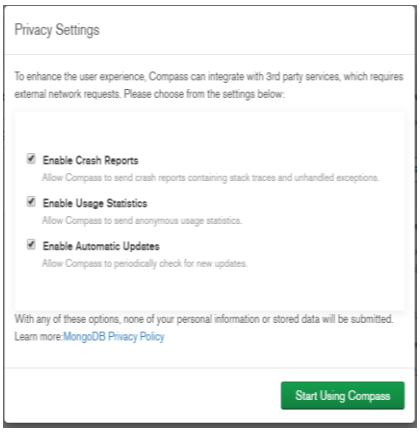
1- Télécharger le fichier zip Mongodb **4.4.0msi**. **[Site 12]**

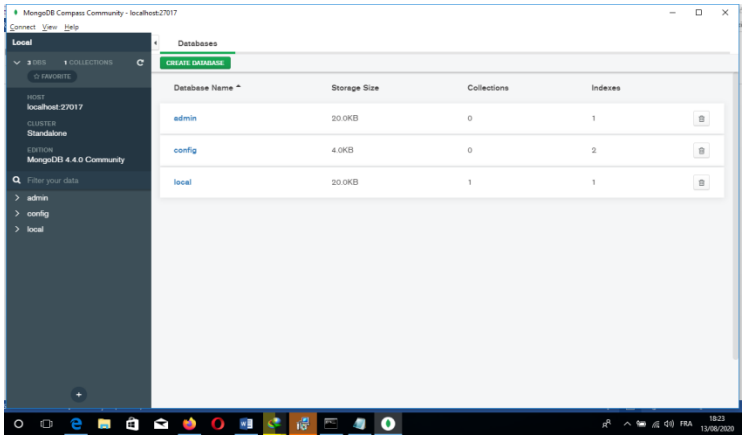
2- Extraire le fichier zip dans le répertoire préparé (disque c ).

**3-** Exécuter mongodb.msi, et à chaque fois cliquer sur suivant jusqu'a terminer l'installation



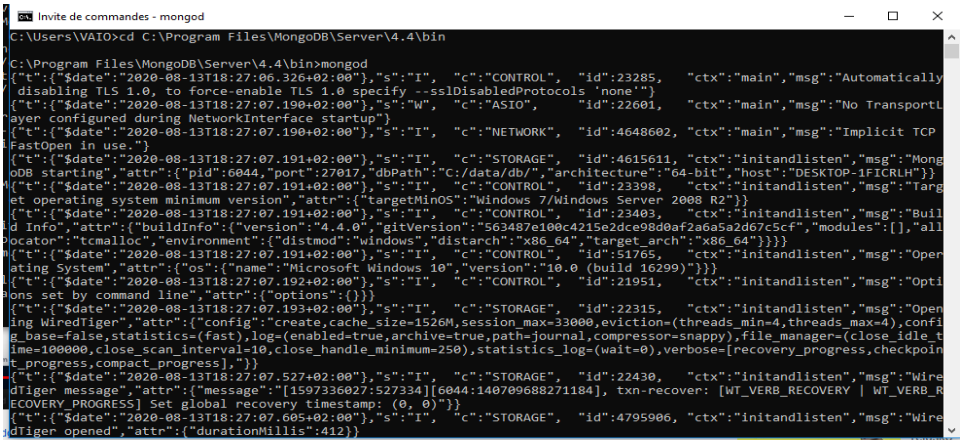
4- Cliquer sur start



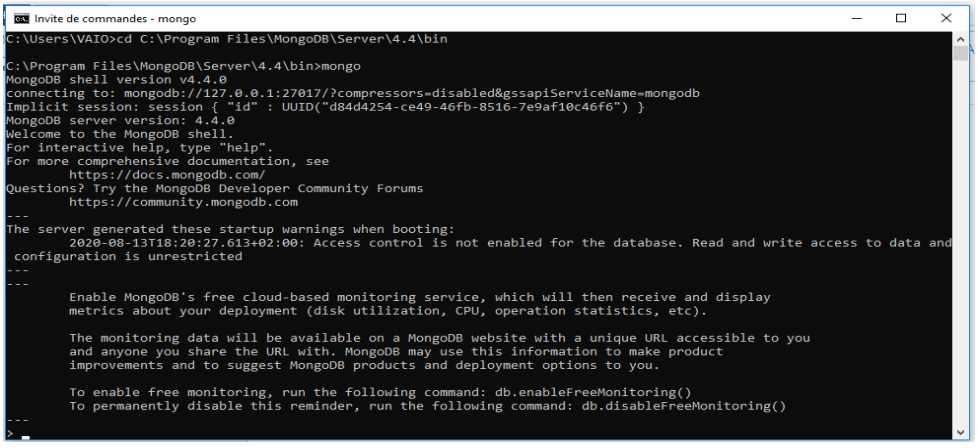




5- Créer un rep data et puis rep db : C:\data\db

6- Connexion de serveur



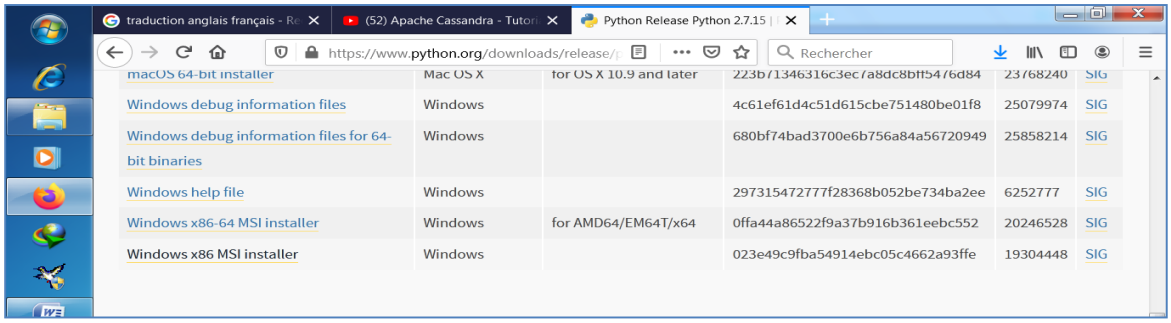

7- Connexion au Shell

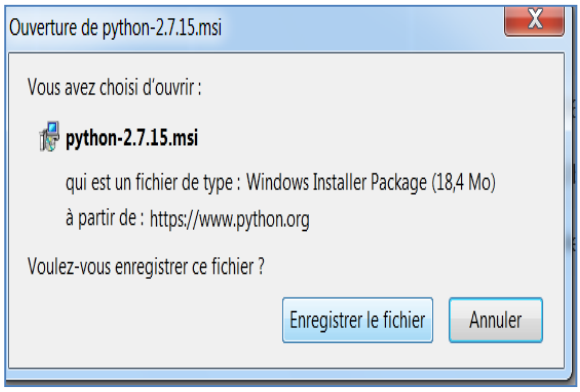


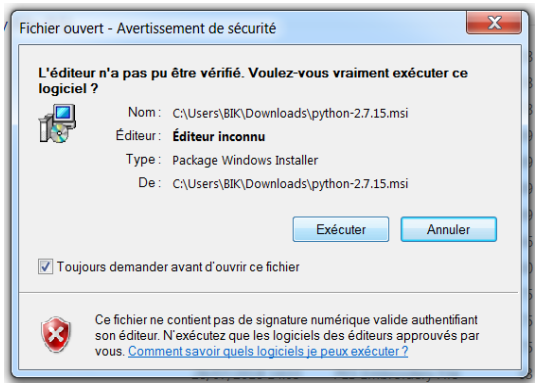

8- Vous été le choix d'ajouter **MongoDB** au path utilisateur et système :

## 5.6. Installation Cassandra Version 3.11.1

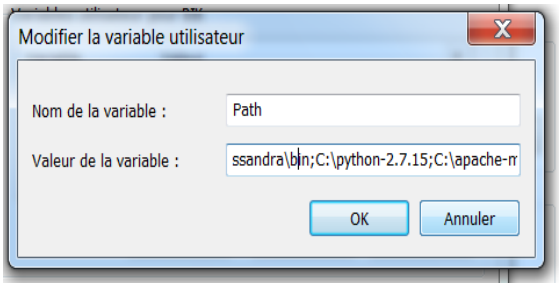
### a- Installer python v 2.7.15 [Site 13]

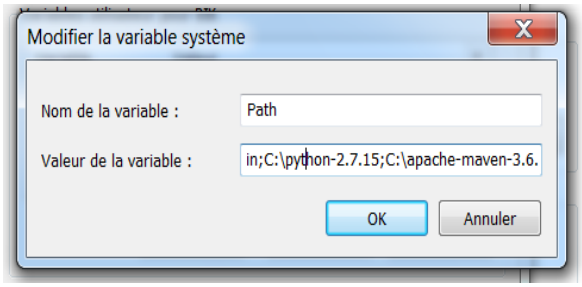


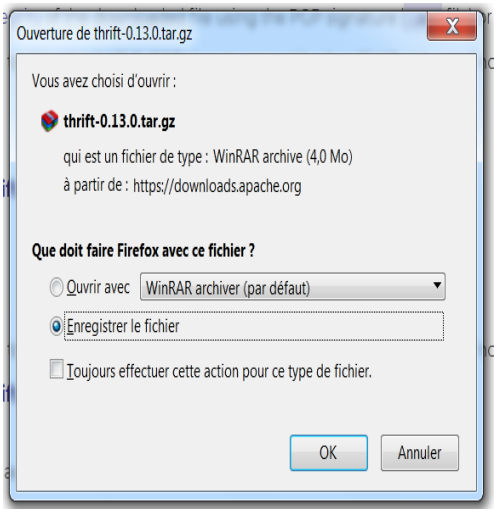



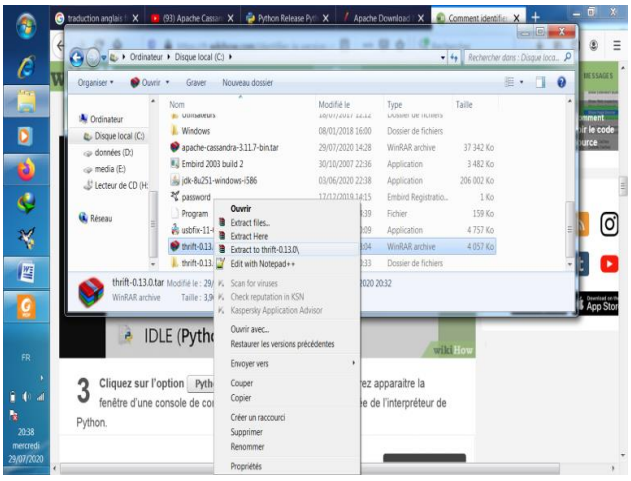

Ajouter **python** au path utilisateur et système :




### b- Extract et installer thrift v 0.13.0.tar.gz [Site 14]

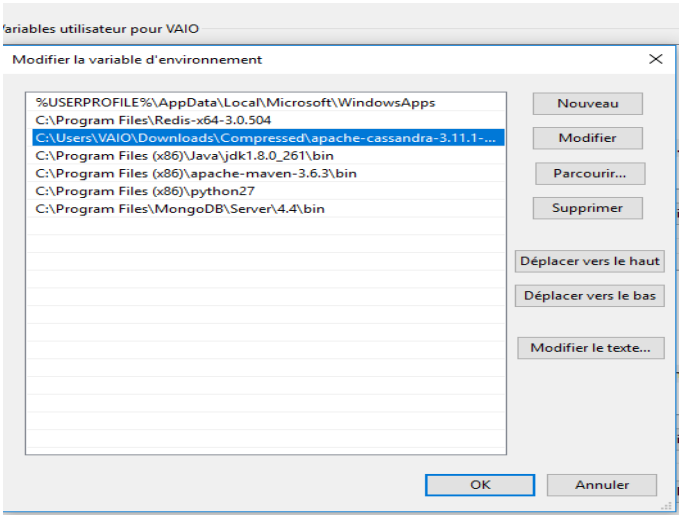


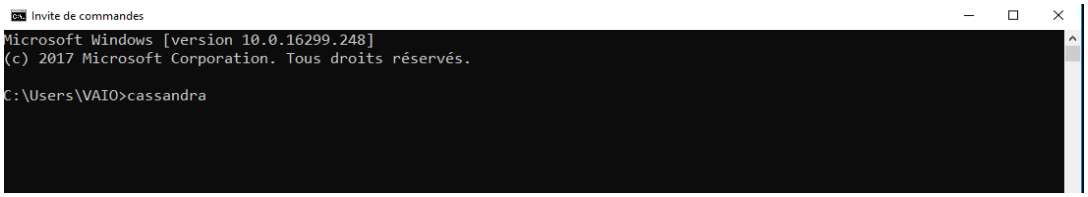
### c- Installer Cassandra v 3.11.1.tar.gz [Site 15]

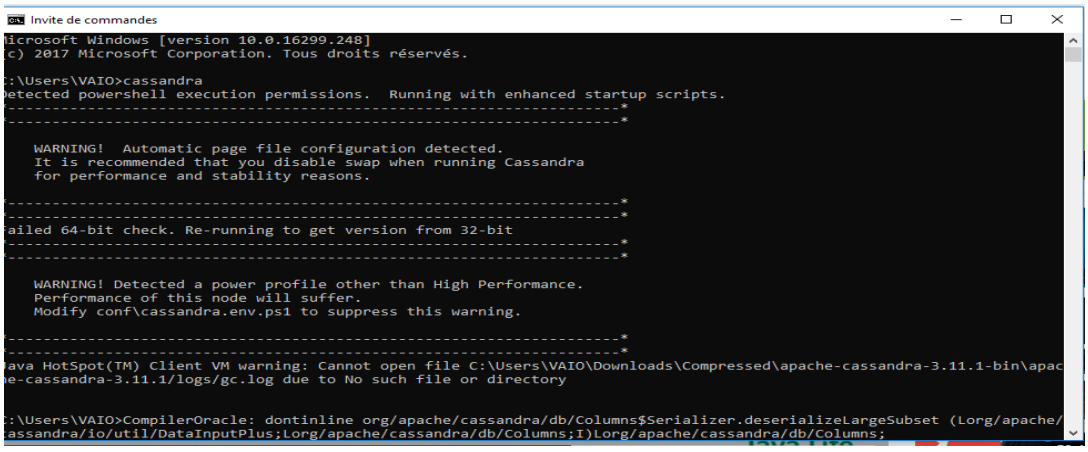
1- Téléchargement et installation de cassandra 3.11.1 à partir de **[Site 9]**

2- Ajouter cassandra au path de variable utilisateur

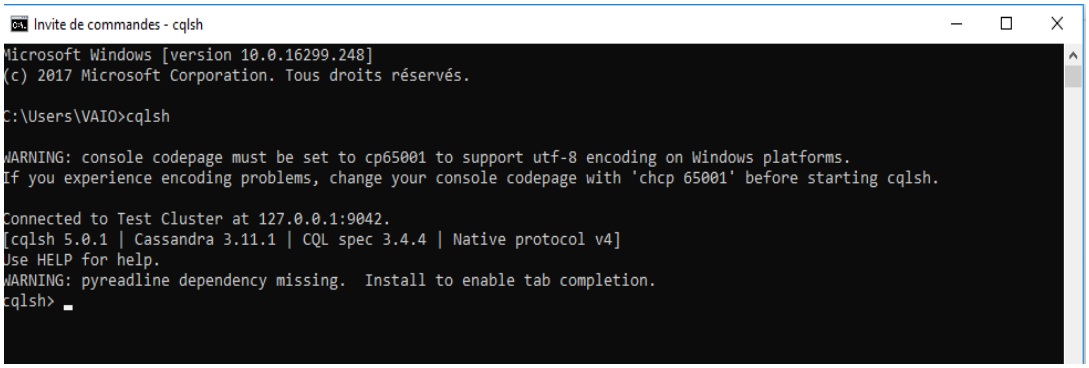



3- Taper Cassandra au cmd pour lancer le server




4- Lancer cqlsh Cassandra (cql_Schell)



### 5.7. Chargement de données (LoadProcess)

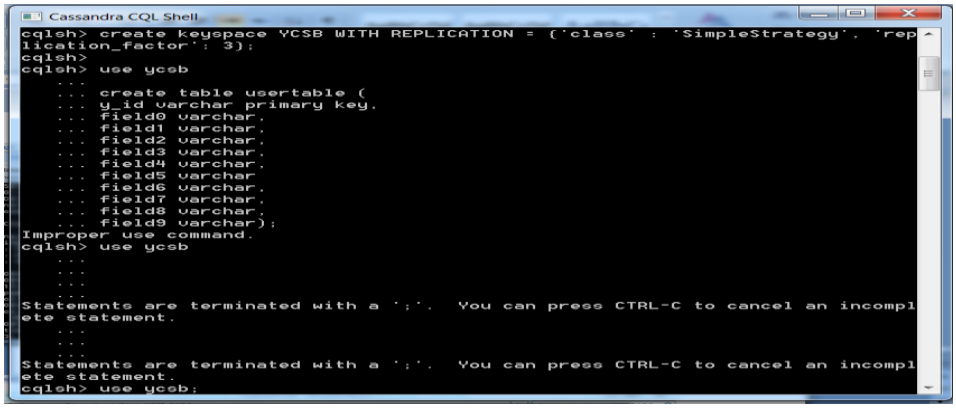
Nous commençons par l'étape de chargement de 800 000 enregistrements générés par YCSB (Load Process), on modifier le nombre d'enregistrements et le nombre d'opérations dans le fichier **workload A** dans le répertoire **YCSB.**

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

- **Test de Cassandra :**

**1-** Creation de keyspace ycsb et puis la table usertable avec la commande suivante dans cassandra cql_Schell.

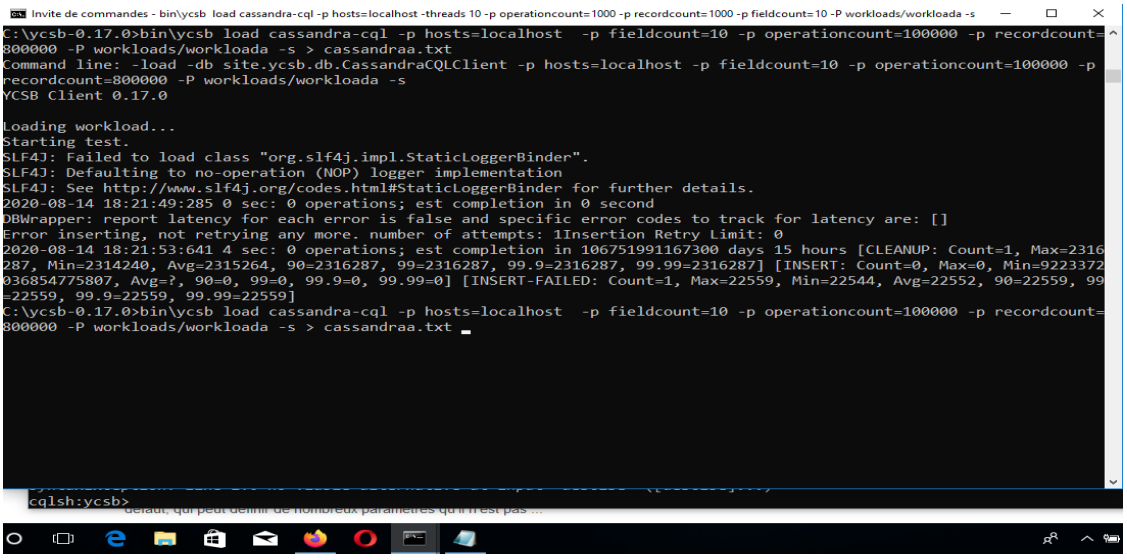
**2-** après ouvrir le répertoire YCSB dans le terminal.



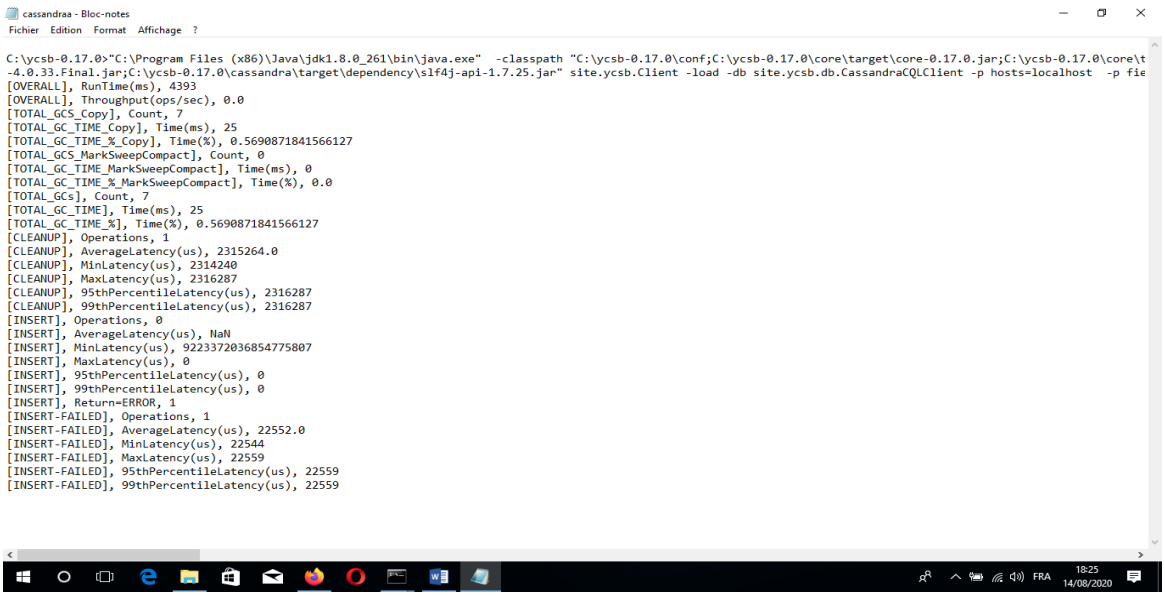

**3-** Taper la commande suivante :

```
c:\ycsb-017.0>bin\ycsb load Cassandra-cql -p  hosts=localhost -p operationcount=100000 -p recordcount=800000 -p fieldcount=10 -p workloads/workloada -s > Cassandra  LOAD2 .txt
```

Le résultat du test est le suivant :



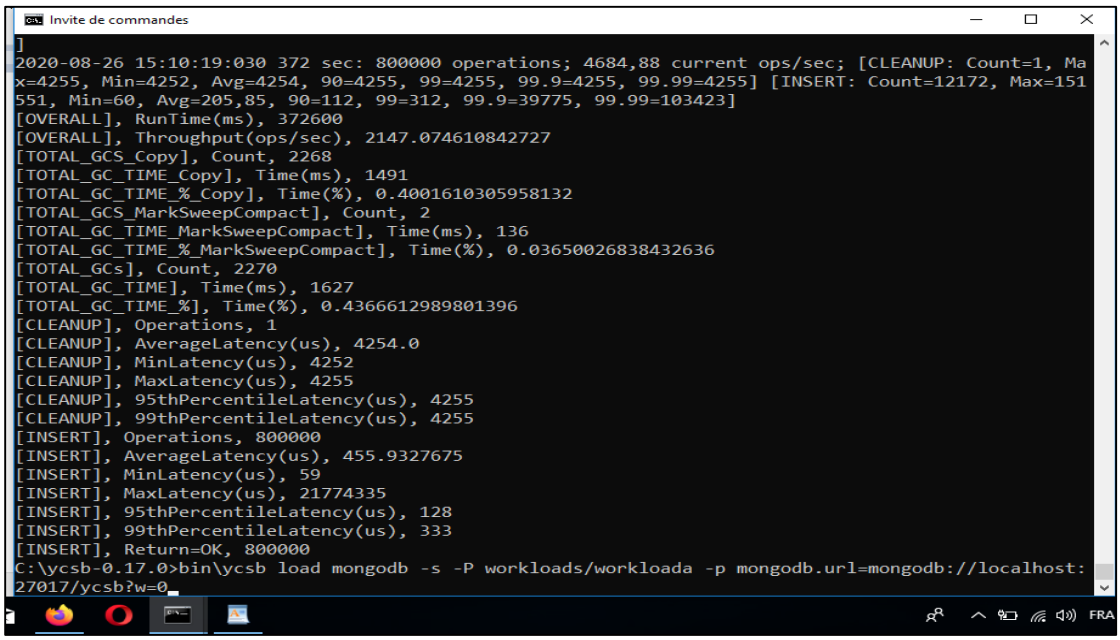
Le journal d'exécution du Workload A par cassandra :




- **Test de mongoDB : Taper la commande suivante :**

```
c:\ycsb-017.0>bin\ycsb load mongodb -s -P workloads\workloada -p mongodb.url = mongodb://localhost:27017/ycsb?w=0
```

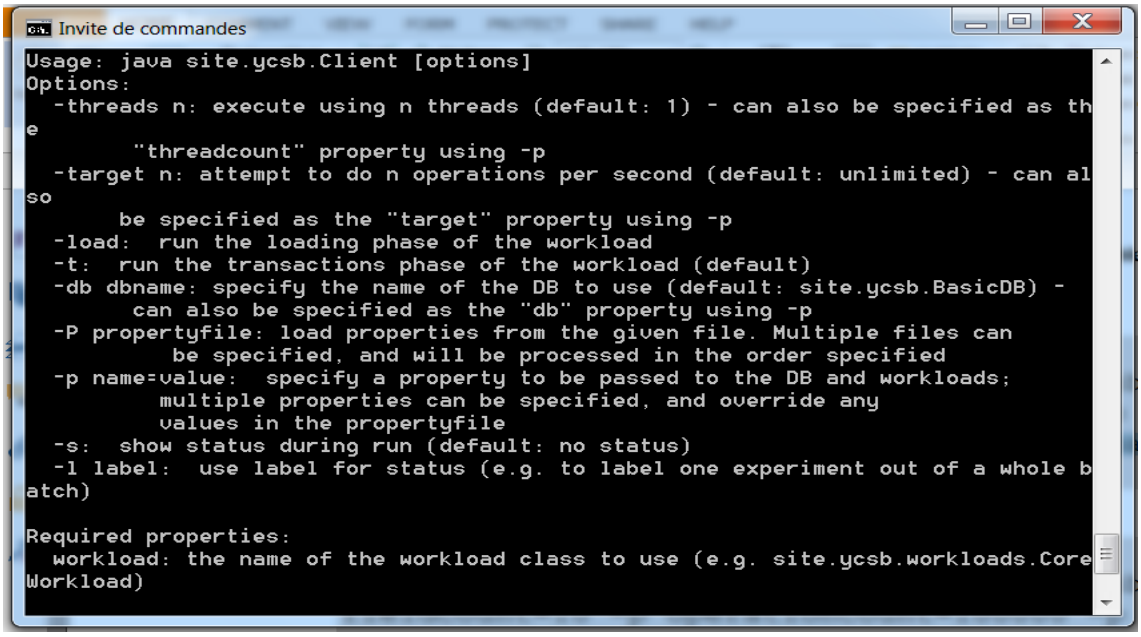
Le journal d'exécution du Workload A par mongodb :



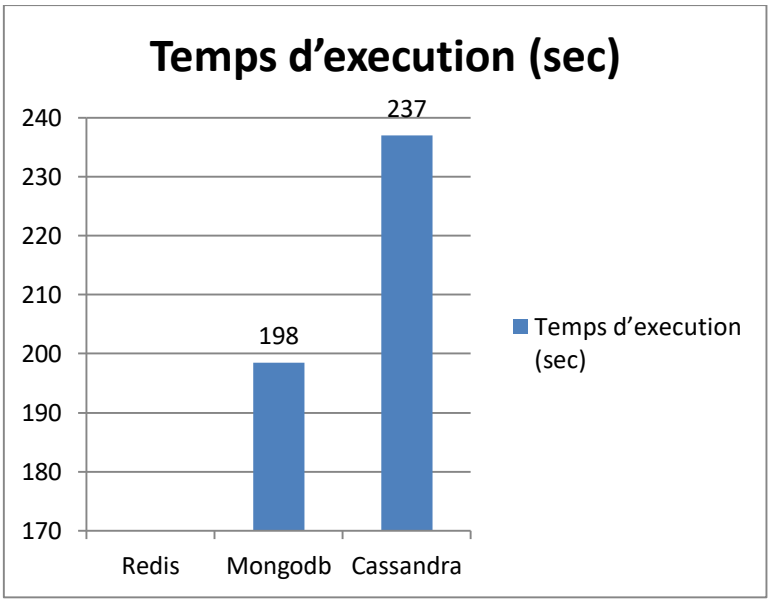

- **Test de Redis**

- c:\ycsb-017.0> bin\ycsb

Quelques options et propriétés utilise dans les commandes de load ou run:




Les résultats sont résumés dans l'histogramme suivant :




| SGBDs Nosql | Temps d'execution (sec) |
|---|---|
| Redis | - |
| Mongodb | 198.481 |
| Cassandra | 236.949 |

**Tableau IV.9 : Temps de chargement**

**Figure IV.9 : Histogramme de Temps de chargement**

La Figure IV.9 montre le temps d'exécution pendant le chargement de 800 000 enregistrements dans deux bases de données (MongoDB et Cassandra ). Nous avons constaté que durant le chargement de 800 000 enregistrements, le meilleurs temps d'insertion a été fournit par MongoDB avec un temps de chargement de **198 sec**. Cela signifie que MongoDB était plus rapide que Cassandra pendant la phase de chargement. La cause est le fait que

MongoDB ne nécessite pas de grande quantité de mémoire pendant l'exécution des opérations de chargement initial.

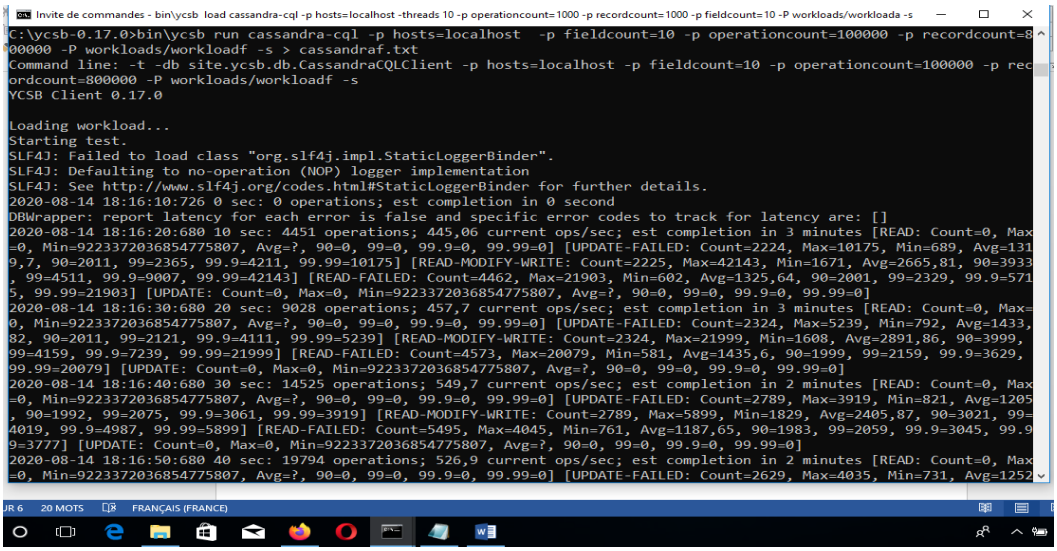
### 5.8. Exécution des Workloads

Dans la section suivante, on va présenter et analyser les résultats des tests. Un rapproche des temps d'exécutions des Workloads effectuées sur 800 000 enregistrements, sera établi. Le nombre d'opérations est limité à 100000 opérations, on modifier le nombre d'enregistrements et le nombre d'opérations dans le fichier **workload A, workload B , workload C ; workload D , workload E, workload F** dans le répertoire **YCSB.**

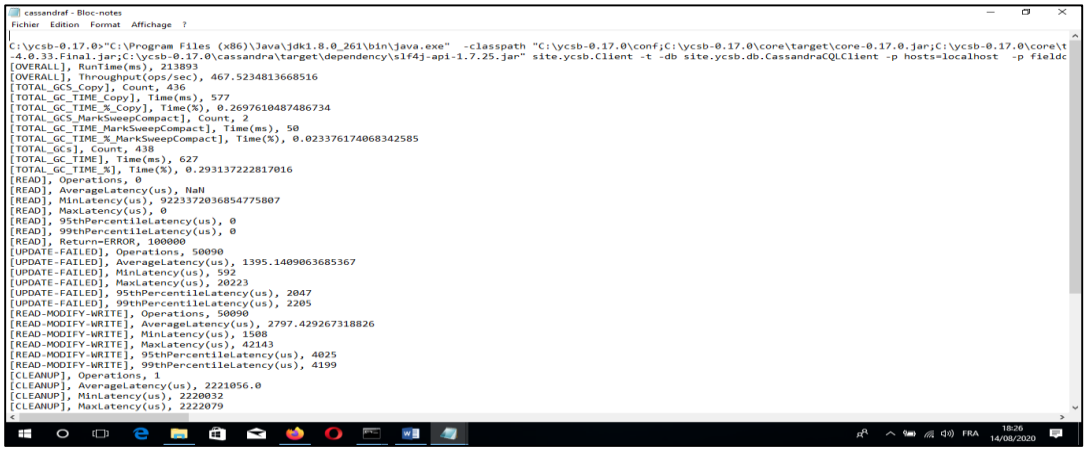
## 1. Workload A (de 50% Read / 50% Update)

- **Test de Cassandra : Taper la commande suivante :**

```
c:\ycsb-017.0>bin\ycsb run Cassandra-cql -p  hosts=localhost -p operationcount=100000 -p recordcount=800000 -p fieldcount=10 -p workloads/workloada -s > Cassandra  LOAD2 .txt
```




Le journal d'exécution du Workload A par cassandra :



- **Test de mongoDB : Taper la commande suivante :**

```
c:\ycsb-017.0>bin\ycsb run mongodb -s -P workloads\workloada -p mongodb.url = mongodb://localhost:27017/ycsb?w=0
```

- **Test de Redis : Taper la commande suivante :**

```
c:\ycsb-017.0> bin\ycsb
```

Le résultat du test des trois systèmes est récapitulé sur le tableau suivant :

| SGBDs Nosql | Temps d'exécution du workload A (sec) | | | |
|---|---|---|---|---|
| | 1er jour | 2eme jour | 3eme jour | Temps moyenne |
| Redis | - | - | - | - |
| Mongodb | 372.600 | 130.719 | 108.194 | 203,925 |
| Cassandra | 141,39 | 143,55 | 130.111 | 138,350 |

**Tableau IV.10 : le temps d'exécution de workload A**

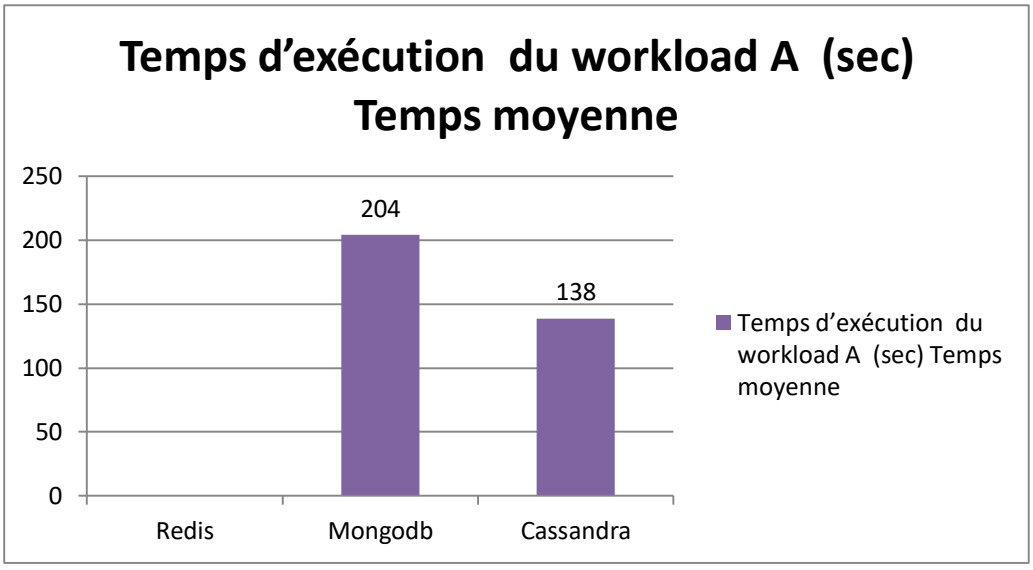



**Figure IV.10 : Histogramme Temps d'exécution moyenne du Workload A**

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

- **Test de Cassandra : Taper la commande suivante :**

```
c:\ycsb-017.0>bin\ycsb run Cassandra-cql -p  hosts=localhost -p operationcount=100000 -p recordcount=800000 -p fieldcount=10 -p workloads/workloadb -s > Cassandrab.txt
```

- **Test de mongoDB : Taper la commande suivante :**

```
c:\ycsb-017.0>bin\ycsb run mongodb -s -P workloads\workloadb -p mongodb.url = mongodb://localhost:27017/ycsb?w=0
```

- **Test de Redis : Taper la commande suivante :**

```
c:\ycsb-017.0> bin\ycsb
```

Le résultat du test des trois systèmes est récapitulé sur le tableau suivant :

| SGBDs Nosql | Temps d'exécution du workload B (sec) | | | |
|---|---|---|---|---|
| | 1er jour | 2eme jour | 3eme jour | Temps moyenne |
| Redis | - | - | - | - |
| Mongodb | 131,242 | 240,814 | 156,411 | 176,156 |
| Cassandra | 143,792 | 142,715 | 139.976 | 142,161 |

**Tableau IV.11 : le temps d'exécution de workload B**

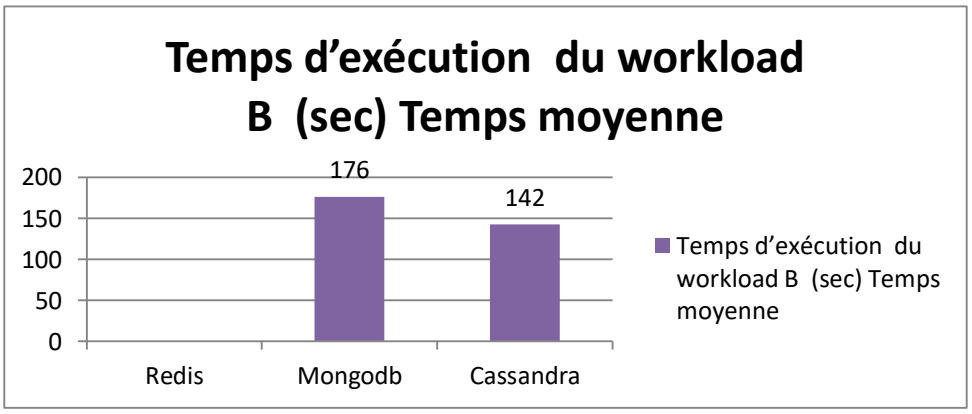



**Figure IV.11 : Histogramme Temps d'exécution moyenne du Workload B**

## 3. Workload C (100% Read)

- **Test de Cassandra : Taper la commande suivante :**

```
c:\ycsb-017.0>bin\ycsb run Cassandra-cql -p  hosts=localhost -p operationcount=100000 -p recordcount=800000 -p fieldcount=10 -p workloads/workloadb -s > Cassandrab.txt
```

- **Test de mongoDB : Taper la commande suivante :**

```
c:\ycsb-017.0>bin\ycsb run mongodb -s -P workloads\workloadc -p mongodb.url = mongodb://localhost:27017/ycsb?w=0
```

- **Test de Redis : Taper la commande suivante :**

```
c:\ycsb-017.0> bin\ycsb
```

Le résultat du test des trois systèmes est récapitulé sur le tableau suivant :

| SGBDs Nosql | Temps d'exécution du workload C (sec) | | | |
|---|---|---|---|---|
| | 1er jour | 2eme jour | 3eme jour | Temps moyenne |
| Redis | - | - | - | - |
| Mongodb | 107.145 | 77,871 | 44,662 | 76,559 |
| Cassandra | 143,957 | 174.994 | 187,152 | 168,701 |

**Tableau IV.12 : le temps d'exécution de workload C**

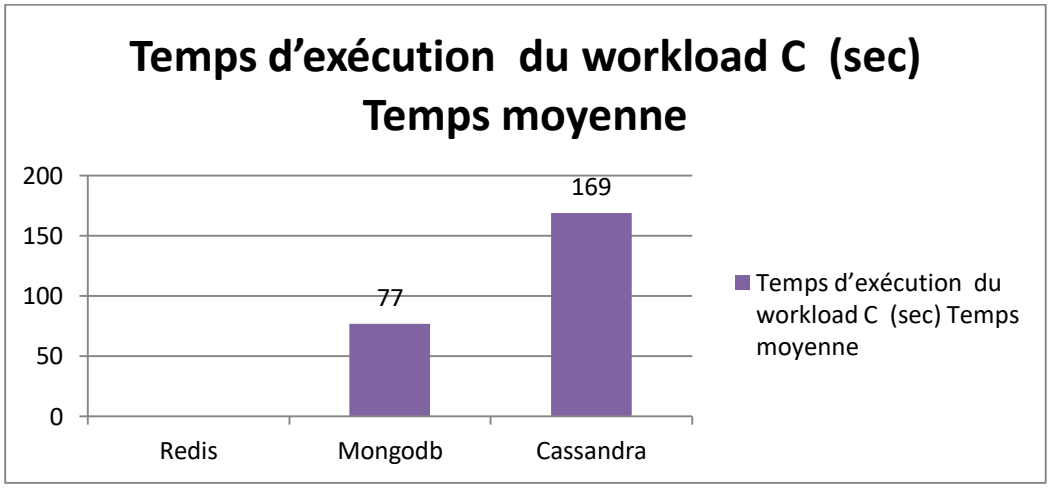



**Figure IV.12 : Histogramme Temps d'exécution moyenne du Workload C**

## 4. Workload D (de 50% Read / 50% Update)

- **Test de Cassandra : Taper la commande suivante :**

```
c:\ycsb-017.0>bin\ycsb run Cassandra-cql -p  hosts=localhost -p operationcount=100000 -p recordcount=800000 -p fieldcount=10 -p workloads/workloadd -s > Cassandrab.txt
```

- **Test de mongoDB: Taper la commande suivante:**

```
c:\ycsb-017.0>bin\ycsb run mongodb -s -P workloads\workloadd -p mongodb.url = mongodb://localhost:27017/ycsb?w=0
```

- **Test de Redis : Taper la commande suivante :**

```
. bin\ycsb run  redis –s  -P  workloads/workloada  –p  “redis.port=6379” > redisworka.txt
```

Le résultat du test des trois systèmes est récapitulé sur le tableau suivant :

| SGBDs Nosql | Temps d’exécution du workload D (sec) | | | |
|---|---|---|---|---|
| | 1er jour | 2eme jour | 3eme jour | Temps moyenne |
| Redis | - | - | - | - |
| Mongodb | 58.690 | 90,176 | 65,922 | 71,596 |
| Cassandra | 148,824 | 141,242 | 136,09 | 141,988 |

**Tableau IV.13 : le temps d’exécution de workload D**

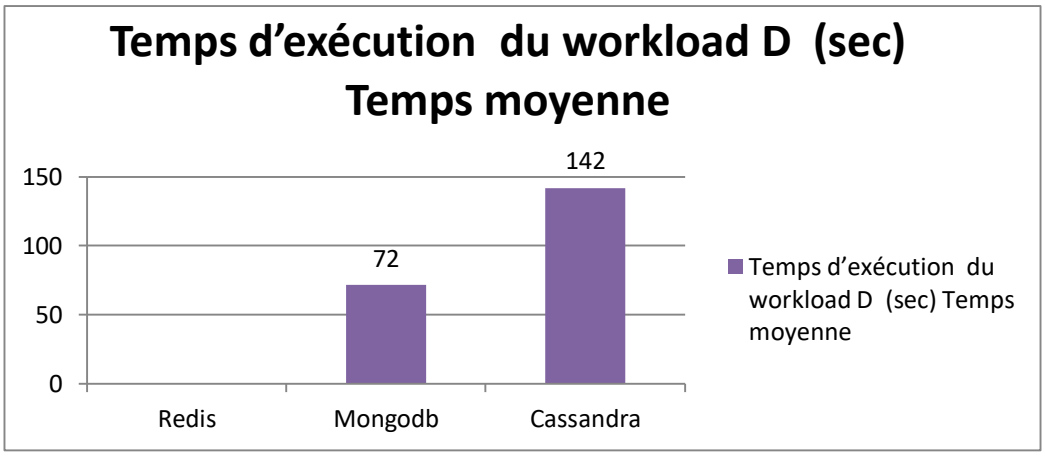



**Figure IV.13 : Histogramme Temps d’exécution moyenne du Workload D**

## 5. Workload E (de 50% Read / 50% Update)

- **Test de Cassandra : Taper la commande suivante :**

```
c:\ycsb-017.0>bin\ycsb run Cassandra-cql -p  hosts=localhost -p operationcount=100000 -p recordcount=800000 -p fieldcount=10 -p workloads/workloade -s > Cassandrab.txt
```

- **Test de mongoDB : Taper la commande suivante :**

```
c:\ycsb-017.0>bin\ycsb run mongodb -s -P workloads\workloade -p mongodb.url = mongodb://localhost:27017/ycsb?w=0
```

- **Test de Redis : Taper la commande suivante :**

```
. bin/ycsb run  redis –s  -P  workloads/workloada  –p  “redis.port=6379” > redisworka.txt
```

Le résultat du test des trois systèmes est récapitulé sur le tableau suivant :

| SGBDs Nosql | Temps d’exécution du workload E (sec) | | | |
|---|---|---|---|---|
| | 1er jour | 2eme jour | 3eme jour | Temps moyenne |
| Redis | - | - | - | - |
| Mongodb | 195.086 | 268,15 | 267,725 | 243,654 |
| Cassandra | 164.753 | 143,196 | 140,581 | 149,510 |

**Tableau IV.14 : le temps d’exécution de workload E**

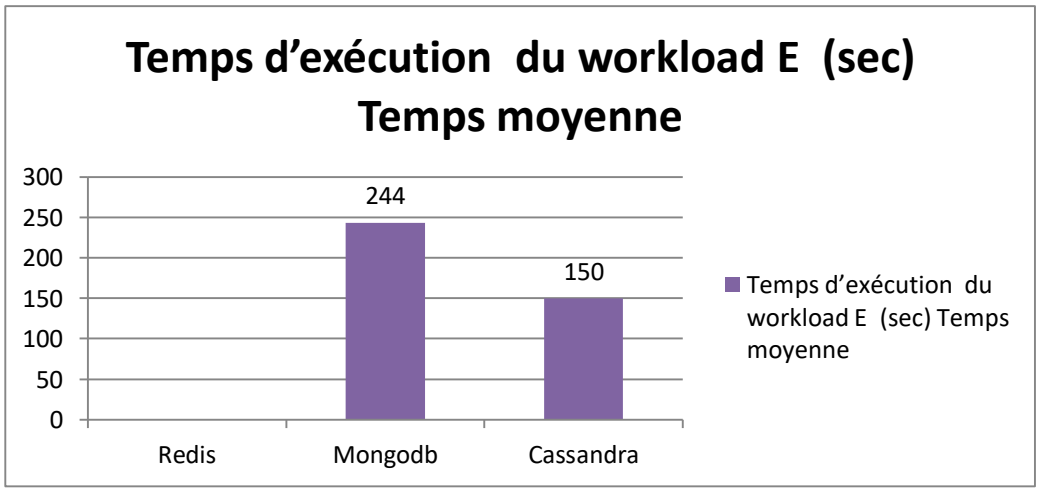



**Figure IV.14 : Histogramme Temps d’exécution moyenne du Workload E**

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

- **Test de Cassandra : Taper la commande suivante :**

```
c:\ycsb-017.0>bin\ycsb run Cassandra-cql -p  hosts=localhost -p operationcount=100000 -p recordcount=800000 -p fieldcount=10 -p workloads/workloadf -s > Cassandrab.txt
```

- **Test de mongoDB : Taper la commande suivante :**

```
c:\ycsb-017.0>bin\ycsb run mongodb -s -P workloads\workloadf -p mongodb.url = mongodb://localhost:27017/ycsb?w=0
```

- **Test de Redis : Taper la commande suivante :**

```
. bin/ycsb run  redis –s  -P  workloads/workloada  –p  “redis.port=6379” > redisworka.txt
```

Le résultat du test des trois systèmes est récapitulé sur le tableau suivant :

| SGBDs Nosql | Temps d’exécution du workload F (sec) | | | |
|---|---|---|---|---|
| | 1er jour | 2eme jour | 3eme jour | Temps moyenne |
| Redis | - | - | - | - |
| Mongodb | 119.363 | 136,729 | 167,114 | 141,069 |
| Cassandra | 230.301 | 213,893 | 201,035 | 215,100 |

**Tableau IV.15 : le temps d’exécution de workload F**

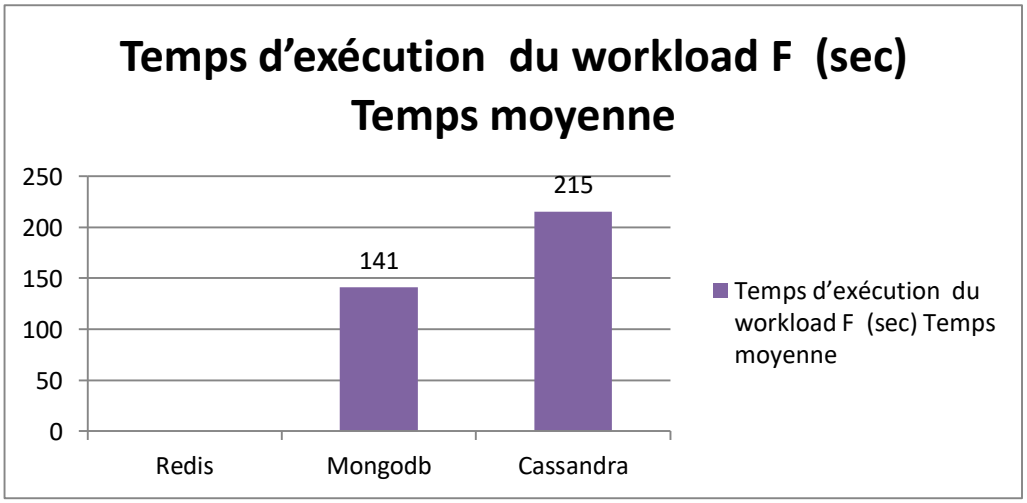



**Figure IV.15 : Histogramme Temps d’exécution moyenne du Workload F**

La Figure IV.16 regroupe les temps d'exécutions de chargement des 800 000 enregistrements et ceux des Workloads composées de 10 000 opérations ainsi que le temps moyen global de tous les tests effectués.

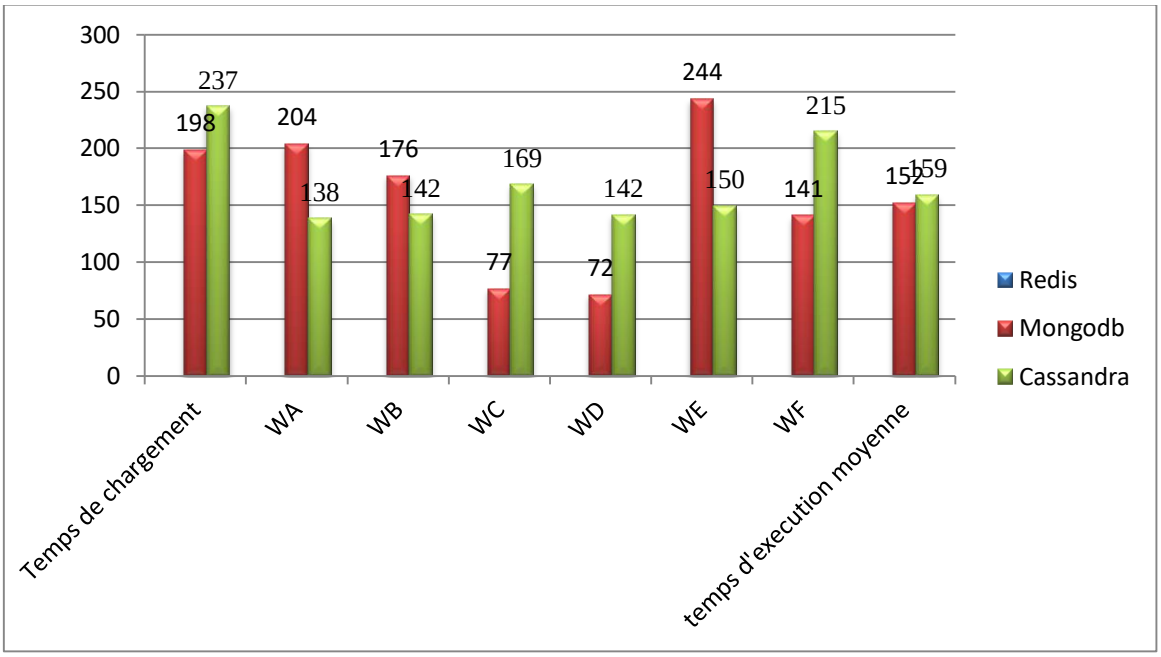



**Figure IV.16 : Histogramme globale de Temps de chargement et le temps d'exécution moyenne de toutes les Workloads**

## 5.9. Evaluation globale

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

- Cassandra a montré sa performance dans les workloads : A, B, E C.à.d. les opérations de lecture et scan (suppression).
- Par contre MongoDB a été plus performant dans les workloads : C, D, F c.à.d. les opérations de lecture. Ceci est du, principalement, à la cartographie des registres de MongoDB chargée en mémoire qui améliore ces performances en lecture.
- L'exécution d'une opération de mise à jour dans MongoDB, est ralentit relativement par rapport aux nombres de mises à jour appliquées. Elle utilise des mécanismes de verrouillage qui alourdissent le temps d'exécution de mise à jour.
- Par contre Cassandra est optimisée pour permettre une écriture rapide et hautement disponible des données avec un débit élevé. Pour ce faire, les données sont écrites dans un premier temps dans un journal de commit (pour s'assurer de la durabilité), puis dans une structure en mémoire appelée la Memtable. Une écriture est considérée comme ayant réussi si les deux actions précédentes ont réussi. Cela permet d'avoir peu d'entrées/sorties

disque au moment de l'écriture. Ainsi, les données sont mises en mémoire et périodiquement écrites sur le disque dans une structure appelée une SSTable (pour Sorted String Table).

## 6. Conclusion

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 MongoDB.
- Pour les opérations de mis à jour lourde, il est très intéressant d'adopter des architectures orientées colonne à savoir Redis.
- Pour des opérations de scan, Cassandra a prouvé sa performance.
- Pour les architectures clé-valeur, beaucoup d'efforts restent à fournir par les concepteurs pour améliorer leurs performances.

# Conclusion générale

## ❖ Bilan

Notre travail était une étude théorique générale des bases de données NoSQL, leurs définitions, leurs caractéristiques et leurs types ; suivie d'une étude spécifique de trois des plus fameuses sur le marché ; c.à.d. Redis, Cassandra et MongoDB ; chacune appartient à une famille différente des BD NoSQL. Nous avons finis par tester leurs performances sur un banc d'essais YCSB et analyser les résultats des tests.

Le modèle clé-valeur est le modèle NoSQL fondamental. Il bénéficie d'une structure simple qui lui permet un gain de performances importantes. Cependant il ne permet pas une manipulation fine de la valeur. Cette limite a motivé le développent de nouveaux modèles orientés colonnes et orienté documents, qui peuvent être considérés comme une forme évoluée du modèle orienté clé-valeur. Ces deux modèles introduisent une structuration de la valeur, mais selon des principes orthogonaux. La valeur peut ainsi être soit atomique, soit composée. Ces modèles se distinguent par le stockage des données qui est effectué soit par document (horizontal), soit par ligne décomposée en familles de colonnes (vertical). Un quatrième modèle NoSQL repose sur le modèle orienté graphes. Il se caractérise par une structure basée sur la théorie des graphes étiquetés.

Dans le Tableau IV.16, nous synthétisons les caractéristiques de ces quatre modèles NoSQL. Nous considérons 9 caractéristiques principales :

1. La **technologie** sur laquelle repose le système, il s'agit du langage avec lequel la solution a été développée.
2. **L'accès** ou la politique utilisée pour la modification des données. Lors d'une opération d'un enregistrement les données sont bloquées (Loks). Un enregistrement peut ne pas être bloqué grâce à la politique MVCC (Multiversion Concurrency control).
3. Le type de mémoire utilisé pour le **stockage** de données. Les données peuvent être stockées par exemple en RAM ou sur disque.
4. La technique de **réplication** des données utilisées (synchrone, asynchrone…).
5. Le type de **licence**. Il s'agit des conditions de mise à disposition de la solution.
6. Les outils **d'interrogation** possibles pour la manipulation des données.

7. La **structure** de la valeur peut être soit atomique (plus petite valeur manipulable) soit composée (valeur composée de plusieurs éléments manipulables).
8. Le **schéma** peut être statique (schéma unique pour un ensemble d'enregistrements) ou dynamique (schéma différent pour chaque enregistrement).
9. Le **requêtage** fournit des fonctions d'accès aux valeurs. L'acronyme CRUD représente les fonctions élémentaires de tous systèmes de gestion de données (écriture, lecture, modification, suppression). Il peut être complété par des fonctions avancées développées dans un langage propriétaire du système comme par exemple SQL, HQJ et CQL.

| Approches / Caractéristiques | Exemple de systèmes | Technologies | Accès | Stockage | Réplication | Licence | Interrogation | Structure | Schéma | Requêtage |
|---|---|---|---|---|---|---|---|---|---|---|
| Clé-Valeur | Redis | c | Locks | RAM | Async | Apache | Ligne de commandes Librairies | atomique | dynamique | CRUD |
| | Voldmort | Java | MVCC | RAM ou BDB | Async | None[2] | ThriftAvroProtobuf | | | |
| | Riak | Erlang | MVCC | Plug-in | Async | Apache | RESET (HTTP) | | | |
| | Memcachedb | Erlang | Locks | Disk | Sync | BSD3 | API | | | |
| orientées colonnes | HBase | Java | Locks | HDFS | Async | Apache | Hive Phoenix | atomique & composée | dynamique | CRUD + langage CRUD |
| | Cassandra | C | MVCC | GFS | Async | Apache 4 | CQL | | | |
| | Bigtable | C | Locks | GFS | Sync&Async | Prop5 | Java | | | |
| | Hypertable | C++ | Locks | Files | Sync | Sync | GPL6 HQL | | | |
| orientées documents | Terrastore | Locks | RAM+ | Sync | Apache7 | HTTP | API Python & java | atomique & composée | dynamique (colonnes) | CRUD + langage CRUD |
| | SimpleDB | Erlang | - | S3 | Async | Prop8 | API | | | |
| | MongoDB | C++ | Locks | Disk | Async | GPL9 | Javascript (Schell) API | | | |
| | CouchDB | Erlang | MVCC | Disk | Async | Apache | REST | | | |
| orientées graphes | Neo4J | Cypher | - | Disk | sync | GPLv3 | REST | atomique & composée | dynamique | CRUD + langage CRUD |
| | OrientDB | Java | - | - | - | Apache 2 | ORIENTDB HOME + dossier bin | | | |

**Tableau IV.16 : Tableau Comparatif des modèles 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.

Nous pouvons dire que les BD NoSQL sont la nouvelle génération des SGBD. Ils sont capables de gérer des volumes de données colossales sur un seul serveur par exemple dans notre étude, mais il est inutile de migrer vers NoSQL ; dans les opérations financières ou les opérations administratives qui nécessitent la réunion de tous les critères ACID, et surtout une cohérence immédiate est intolérable, ce sont des applications à secrets. Aussi, les applications qui n'ont pas l'intention d'évoluer ou de s'intégrer avec d'autres ; les SGBDR leurs suffit largement. Le NoSQL n'est pas toujours le bon choix. Il est possible de mélanger les deux produits, à savoir le SGBDR et le NoSQL pour avoir le résultat escompté.

## ❖ 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, Hbase, Accumulo et autres.
- Multiplier le nombre d'enregistrements pour atteindre ou dépasser le million.
- Tester des bases de données NoSQL dans un environnement distribué réel, par exemple le Cloud Computing.

# Références

[AMW15] AMARA Manel Warda , « Etude comparative des bases de données NoSQL », Année universitaire : 2014-2015

[Abr et all 13] Abramova, V and Bernardino, J (2013). NoSQLdatabases:MongoDB vs Cassandra. In Proceedings of the International C* Conference on Computer Science and Software Engineering (C3S2E '13). July 10 - 12, 2013, Porto, Portugal

[BABM16] BACHIR Ahmed - BOUDEFLA Mohammed ,« Migration de la base de données des passeports biométriques sous Oracle vers une base de données NoSQL MongoDB » , Année universitaire: 2015-2016

[BEN15] Melle. BENAMARA Houa Melle. BENYAHIA Housna, « Conception et Réalisation d'une application pour la gestion des stocks », Année universitaire 2014/2015.

[BZTH16] BENALLAL Zeyneb - TAHRAOUI Hayet, «Etude comparative des bases de données NoSQL : MongoDb, CouchBase, Cassandra, HBase, Redis, OrientDB » , Année universitaire :2015-2016

[Cat et all 10] Cattell, R (2010). Scalable SQL and NoSQL Data Stores. SIGMOD Record, Volume 39, Issue 4, pp. 12-27, Indiana, USA

[CBF et all 10] Cooper, BF, Silberstein, A, Tam, E, Ramakrishnan, R and Sears, R (2010). Benchmarking cloud servingsystemswith YCSB. In Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC'10). ACM, Indianapolis, Indiana, USA, June 10 - 11, 2010, pp.143-154.

[DEV et all 16] Deepika V. Shetty, Sana J.Chidimar, « Comparative Study of SQL and NoSQL Databases to evaluate their suitability for Big Data Application» 2, Mumbai : International Journal of Computer Science and Information Technology Research, 2016, Vol.

[DIM 12] Di Maglie, « Adoption d'une solution NoSQL dans l'entreprise» , Bachelor HES (Haute École de Gestion de Genève),M (2012)

[Fur15] Furrer, H SQL, NoSQL, NewSQL stratégie de choix. Bachelor HES (Haute École de Gestion de Genève), (2015).

[GDE 16] GC, Deepak, A Critical Comparison of NOSQL Databases in the Context of Acid and Base. St. Cloud : Culminating Projects in Information Assurance, 2016. Paper 8.

[GKH et all13] Grolinger, K, Higashino, WA, Tiwari, A and Capretz, MAM, «Data management in cloud environments: NoSQL and NewSQL data stores» Journal of Cloud Computing: Advances, Systems and Applications, (2013).

[Han et all 11] Han, J, Haihong, E, Le, G and Dual, J, «Survey on NoSQL database», (2011).

[Hei12] Heinrich, L, Architecture NoSQL et réponse au Théorème CAP. BachelorHES (Haute École de Gestion de Genève), (2012).

[KEM15] Kouedi Emmanuel Maître en Informatique, « Approche de migration d'une base de données relationnelle vers une base de données NoSQL orientée colonne ».

[Kum et all 15] Lokesh Kumar, Dr. Shalini Rajawat, Krati Joshi, Comparative analysis of NoSQL (MongoDB) with MySQL Database". 711, Rajasthan, India : International Journal of Modern Trends in Engineering and Research, Vol. 1. 2393- 8161, 2015 .

[Léo14] Léonard, M ,L'avenir du NoSQL, (2014). http://www.leonardmeyer.com/wpcontent/uploads/ 2014/06/avenirDuNoSQL.pdf

[Mal12] Maletras, X, Le NoSQL - Cassandra, Thèse Professionnelle (Université Paris 13), (2012).

[MH18] Mr MATALLAH Houcine, «Vers un nouveau modèle de stockage et d'accès aux données dans les Big Data et les Cloud Computing»,Soutenue publiquement en Septembre 2018.

[OH et all 18] Omar Hajoui, Rachid Dehbi, Mohammed Talea, Zouhair Ibn Batouta, 2348-1196. An advanced comparative study of the most promising NoSQL and NewSQL databases with a multi-criteria analysis method . 03, MOROCCO : International Journal of Computer Science and Information Security, Vol. 81. 1947-5500, 2018.

[PAG 13] Adriano Girolamo PIAZZA, « NoSQL Etat de l'art et benchmark », Bachelor HES (Haute École de Gestion de Genève), (2013).

[PS] Pierre Senellart, « Limites des systèmes classiques de gestion de bases de données »

[Site 1] https://www.oracle.com/fr/database/base-de-donnees-relationnelle-definition.html

[Site 2] https://www.lebigdata.fr/acid-base-de-donnees-definition

[Site 3] http://cours.singaomara.com/baseDeDonnees.html

[Site 4] https://laurent-audibert.developpez.com/Cours-BD/?page=introduction-bases-de- donnees#L1-1-2-c

[Site 5] Solid IT, (2020). DB-EnginesRankinghttp://db-engines.com/en/ranking. (Consulté en mars2020).

[Site 6] MongoDB (2016). Release Notes. http://docs.mongodb.org/manual/release-notes. http://hbase.apache.org/book.html#arch.overview (Consulté en mars 2020)

[Site7] Goebel, N (2013). A Quick Introduction to Apache Cassandra. https://www.sitepoint.com/a-quick-introduction-to-apache-cassandra/

[Site8] Doan, DH (2014). Modèle de stockage physique dans Cassandra. http://www.infoq.com/fr/articles/modele-stockage-physique-cassandra.

[Site 9] http://cassandra.apache.org

[Site 10] https://maven.apache.org/download.cgi

[Site 11] https://riptutorial.com/redis/example/29962/installing-and-running-redis-server-on-windows

[Site12] https://www.codeflow.site/fr/article/mongodb__how-to-install-mongodb-on-windows

[Site 13] https://www.python.org/downloads/release/python-2715/

[Site 14] http://www.apache.org/dyn/closer.cgi?path=/thrift/0.13.0/thrift-0.13.0.tar.gz

[Site 15] https://phoenixnap.com/kb/install-cassandra-on-windows