# Sommaire

# Introduction Générale

# Introduction Générale

L'information joue un rôle important et vital pour les organisations publiques ou privées. Avec la démocratisation de l'internet et la multiplication des sources produisant des données informatiques, on assiste ces dernières années à un changement dans les habitudes des utilisateurs qui devient très exigeants, ces masses d'informations stockées et manipulées par le nombre important d'utilisateurs est quant à lui estimé aujourd'hui à plusieurs centaines de millions objets.

La variété et la nature des données prises en charges, issues des différentes sources informatiques, est un effet marquant de cette nouvelle ère informatique. La gestion des données, durant les trois dernières décennies, a utilisé principalement les systèmes de gestion de bases de données relationnelles.

Le fort trafic des données caractérisant des sites Web tels que Facebook, Google, Amazon, LinkedIn et autres grands acteurs ont été trouvé rapidement limités par ces systèmes transactionnels pour deux raisons majeures à savoir, la volumétrie et la montée en charge, ce qui engendre naturellement une dégradation conséquente dans les performances. N'ayant pas trouvé de solution sur le marché répondant à ces problèmes, ces acteurs du Web décidèrent de développer chacun à l'interne leurs propres SGBD et à chercher des remèdes en multipliant leurs infrastructures et en basculant vers des environnements distribués.

De ce fait, et face à ces contraintes, des nouveaux SGBD nommés «NoSQL» viennent occuper leurs places en proposant des alternatives au modèle relationnel. Les bases de données NoSQL offrent de nouvelles solutions de stockage dans les environnements à grande échelle.

Actuellement, beaucoup d'utilisateurs des SGBD relationnels dits « SQL » veulent basculer vers ces nouvelles solutions non relationnelles « NoSQL » pour anticiper l'explosion de leurs données dans le futur et la prise en charge des données non structurées.

Pour justifier et motiver un tel basculement du SQL vers du NoSQL, nous allons développer, dans le cadre de notre PFE, une étude comparative des performances entre deux types de solutions très répandues actuellement, à savoir MySQL, faisant partie des bases de données relationnelles connu par SQL et MongoDB, faisant partie des bases de données non relationnelles, représentant la nouvelle mouvance NoSQL.

Ce manuscrit est organisé comme suit :

Après l'introduction générale, nous allons rappeler, dans le premier chapitre, les concepts de base de l'architecture relationnelle ainsi que ses limites.
La nouvelle génération des bases de données dites « NoSQL » est présentée dans le second chapitre.
L'étude comparative développée ainsi que tous les résultats expérimentaux feront l'objet du dernier chapitre.

# Chapitre 1 :

## Bases de données relationnelles



### Introduction

Dans ce premier chapitre, nous allons présenter les bases de données relationnelles en rappelant les principes et les concepts de base. Par la suite, nous dévoilerons leurs limites dans les environnements à grande échelle et face aux nouveaux besoins des systèmes d'informations modernes.

## I.1 Généralités sur le modèle relationnel

Le modèle relationnel a été introduit par **Edgar Frank Codd** du centre de recherche d'IBM à travers son article « A Relational Model of Data for Large Shared Data Banks » publié en juin 1970, sa simplicité et ses fondations mathématiques sont faites l'objet de très nombreuses recherches qui ont débouché sur la réalisation de nombreux SGBD relationnels.

Un seul type de structure utilisé pour représenter les données stockées dans la machine : la relation ou la table. Ces données sont gérées par un système de gestion de bases de données relationnel [1].

### I.1.1 Définitions

#### I.1.1.1Base de données

Une base de données est une entité dans laquelle est possible de stocker des données de façon structurée et persistante (peut être rechargée en mémoire principale à tout instant), avec le moins de redondance possible, servant pour les besoins d'une ou plusieurs applications, interrogeables et modifiables par un groupe d'utilisateurs [2].

#### I.1.1.2Table (ou Relation) et Attribut

Une table (ou relation) est un tableau à deux dimensions composé de colonnes appelée Attribut et de lignes, l'intersection des ces derniers contenant les données. Une table peut en contenir un nombre quelconque de lignes. On trouve aussi ce qu'on appel enregistrement, n-uplet ou tuple pour désigner une ligne et un champ pour désigner une colonne ; un enregistrement est donc un ensemble de valeurs, chacun renseignant un champ.

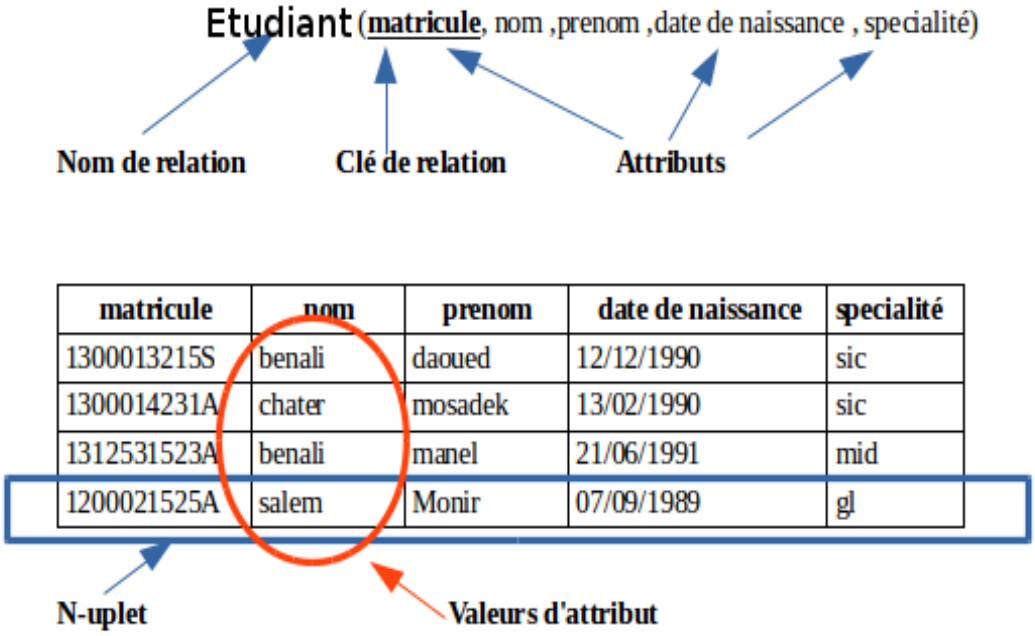



**Figure1.1 : Schéma d'une relation**

### I.1.1.3Clés

Parmi les concepts fondamentaux du modèle relationnel est la notion de clé. Deux types de clé peuvent être distingués :

- **Clés primaires**

Dans la table de l'exemple ci-dessus, le matricule d'étudiant est un attribut constitué d'un code unique. Cet attribut particulier, appelé clé, est indispensable qui permet d'identifier chaque étudiant d'une manière unique. En effet, l'attribut nom peut prendre la même valeur plusieurs fois et donc ne peut être la clé de la table, de même si l'on avait disposé du prénom des étudiants, la combinaison d'attributs **<nom et prénom>** n'aurait pas été d'avantage satisfaisante comme clé, car il est possible que deux étudiants portent le même nom et le même prénom à la fois [3].

- **Clés étrangères**

Une clé étrangère est la duplication de la clé primaire d'une table dans une autre table dans le but de traduire une association conceptuelle entre les deux entités représentées par les deux tables. Ce concept est un des concepts majeurs du modèle relationnel [2].

## I.1.2 Système de gestion de base de données (SGBD)

### I.1.2.1 Définition

Un Système de Gestion de Bases de Données (SGBD) peut être défini comme un ensemble de logiciels prenant en charge la structuration, le stockage, l'interrogation et la mise à jour des données. Il permet l'accès aux données de façon simple, manipuler les données présentes dans la base de données ainsi qu'autoriser l'accès aux informations à de multiples utilisateurs.

### I.1.2.2 Objectifs

Un SGBDR a comme objectifs [3] :

- **Administration facilitée des données**

Toutes les données doivent être centralisées dans un réservoir unique commun à toutes les applications. En effet, des visions différentes des données se résolvent plus facilement si les données sont administrées de façon centralisée.

- **Efficacité des accès aux données**

Les performances en termes de débit (nombre de transactions types exécutées par seconde) et le temps de réponse (temps d'attente moyen pour une requête type) sont un problème clé des SGBD. L'objectif de débit élevé nécessite un temps de latence minimal dans la gestion des tâches accomplies par le système. L'objectif d'avoir un bon temps de réponse implique qu'une requête courte d'un utilisateur n'attend pas une requête longue d'un autre utilisateur. Il faut donc partager les ressources (unités centrales, unités d'entrées-sorties, etc.) entre les utilisateurs en optimisant l'utilisation globale.

- **Redondance contrôlée 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 être vérifiées automatiquement à chaque insertion, modification ou suppression des données. Les contraintes d'intégrité sont décrites dans le Langage de Description de Données (LDD).

- **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 multiutilisateurs 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**

Cet objectif a deux aspects : Tout d'abord, les données doivent être protégées contre les accès non autorisés ou mal intentionnés, pour cela, il doit exister des mécanismes adéquats pour autoriser, contrôler ou retirer les droits d'accès de n'importe quel usager à un ensemble de données. D'un autre côté, la sécurité des données doit aussi être assurée en cas de panne d'un programme, du système, ou même de la machine. Un bon SGBD doit être capable de restaurer des données cohérentes après une panne disque, bien sûr à partir de sauvegardes. Aussi, si une transaction commence une mise à jour (par exemple un transfert d'un compte bancaire vers un autre) qui est interrompue par une panne en cours de son exécution, le SGBD doit assurer l'intégrité de la base (c'est-à- dire que la somme d'argent gérée doit rester constante) et par suite défaire la transaction qui a échoué [4].

### I.1.2.3Quelques SGBD connus et utilisés

Il existe de nombreux systèmes de gestion de bases de données, en voici une liste non exhaustive :

- **ACCESS** : Plateforme Windows, monoposte, licence commerciale
- **SQL SERVER** : Plateforme Windows, mode client/serveur, licence commerciale
- **ORACLE** : Plateformes Windows et Linux, mode client/serveur, licence commerciale
- **SYBASE** : Plateforme Windows et Linux, mode client/serveur, licence commerciale
- **POSTGRESQL** : Plateforme Windows et Linux, mode client/serveur, licence libre
- **MySQL** : Plateforme Windows et Linux, mode client/serveur, licence libre

## I.2 Langages de manipulation des données relationnelles

Les langages utilisés dans les bases de données relationnelles se fondent sur l'algèbre relationnel et/ou le calcul relationnel. Ce dernier est basé sur la logique des prédicats (expressions d'une condition de sélection définie par l'utilisateur). On peut distinguer 3 grandes classes de langages :

### I.2.1 Calcul relationnel

Construit à partir de la logique des prédicats (logique 1$^{iere}$ ordre), un langage déclaratif qui permet de spécifier des prédicats qui doivent être vérifiés par les données pour former le résultat.

### I.2.2 Langages algébriques

Basés sur l'algèbre relationnelle d'Edgar Frank Codd, est le langage interne d'un SGBD relationnel. Il se compose d'un certain nombre d'opérateurs de manipulation des relations. Ces opérateurs sont regroupés en deux familles, les opérateurs ensemblistes et les opérateurs relationnels.

### I.2.3Langage SQL (Structured Query Language)

**St**ructured **Q**uery **L**anguage est un langage standard pour exploiter les bases de données. Le langage est reconnu par la grande majorité des systèmes de gestion de bases de données relationnelles (SGBDR), il permet :

- La définition des données (CREATE, ALTER, DROP, RENAME)
- La manipulation de données (INSERT, UPDATE, DELETE)
- L'interrogation de la base (SELECT)

SQL a été créé au début des années 1970 par la société américaine IBM (International Business Machines) dont sa première version commercialisable a vu le jour en 1979. Depuis, il a été repris par le groupe ORACLE qui a lancé son premier SGBD Relationnel et a été normalisé par l'Institut de normalisation américaine ANSI en 1986 et ratifié par l'organisation internationale de normalisation ISO en 1987.

Le langage SQL peut être considéré comme le langage d'accès normalisé aux bases de données. Il est aujourd'hui supporté par la plupart des produits commerciaux que ce soit par les systèmes de gestion de bases de données telles qu'Access ou par les produits plus

professionnels tels qu'Oracle. Il a fait l'objet de plusieurs normes ANSI/ISO, dont la dernière est la norme SQL3 qui a été définie en 1999, et qui a subi plusieurs révisions dont la plus récente date de 2011.

Le succès du langage SQL est dû essentiellement à sa simplicité et au fait qu'il s'appuie sur le schéma conceptuel pour énoncer des requêtes en laissant la responsabilité de la stratégie d'exécution au SGBD. SQL est un langage déclaratif qui permet d'interroger une base de données sans se soucier de la représentation interne (physique) des données, de leur localisation, des chemins d'accès ou des algorithmes nécessaires. A ce titre il s'adresse à une large communauté d'utilisateurs potentiels (pas seulement des informaticiens) et constitue un des atouts les plus spectaculaires des SGBDR. [6].

## I.3 Principe d'ACID

La plupart des SGBD relationnels sont transactionnels, ce qui impose le respect des contraintes : Atomicité, Consistance, Isolation, Durabilité ; l'acronyme ACID permet de garantir la fiabilité d'une transaction informatique :

### I.3.1 Atomicity (Atomicité)

Une transaction doit être atomique, qui veut dire qu'elle doit être complètement effectué, ou pas du tout. Par exemple, sur 9000 lignes devant être modifiés au sein d'une même transaction : si la modification d'une seule ligne échoue donc la transaction entière doit être annulée. Ceci est nécessaire, car chaque ligne modifiée peut dépendre du contexte de modification d'une autre et toute rupture de ce contexte pourrait engendrer une incohérence des données de la base.

### I.3.2 Consistency (Cohérence)

Chaque transaction doit préserver la cohérence de données; Cela signifie que les modifications apportées à la base doivent être valides, en accord avec l'ensemble de la base et de ses contraintes d'intégrité. Si le contenu final d'une base de données contient des incohérences, cela entraînera l'échec et l'annulation de toutes les opérations de la dernière transaction. Le système revient au dernier état cohérent. La notion de cohérence est importante dans les SGBDR.

### I.3.3 Isolation

Chaque transaction voit l'état du système comme si elle était la seule à manipuler la base de données, aucune dépendance possible entre les transactions. Par exemple si deux transactions **A** et **B** s'exécutant en même temps, les modifications effectuées par **A** ne sont ni visibles, ni modifiables par **B** tant que la transaction **A** n'est pas terminée et validée par un «commit ».

### I.3.4 Durability (Durabilité)

Une transaction confirmée le demeure définitivement, peu importe les différents imprévus (panne d'électricité, panne d'ordinateur etc.). Pour ce faire, un journal contenant toutes les transactions est créé, afin que celles-ci puissent être correctement terminées lorsque le système est à nouveau disponible.

## I.4 Limites des SGBD relationnels

Le modèle relationnel est fondé sur des concepts simples qui font sa force, en même temps sa faiblesse. Les principales limites des SGBD Relationnels, sont présentées dans ce qui suit :

- **Problème lié à la gestion des objets hétérogènes**

« Le stockage distribué n'est pas la seule contrainte qui pèse à ce jour sur les systèmes relationnels» disait Carl STROZZI. Au fur et à mesure du temps, les structures de données manipulées par les systèmes sont devenues de plus en plus complexes, en contrepartie les moteurs de stockage évoluant peu. Le principal point faible des modèles relationnels est l'absence de gestion d'objets hétérogènes ainsi que le besoin de déclarer au préalable l'ensemble des champs représentant un objet [7].

- **Surcharge sémantique**

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

- **Types de données**

Ces modèles sont limités à des types simples (entiers, réels, chaînes de caractères). Les seuls types étendus se limitant à l'expression de dates ou de données financières, ainsi que des conteneurs binaires de grande dimension (BLOB, pour Binary Large OBjects) qui permettent de stocker des images ainsi que des fichiers audio ou vidéo. Ces BLOBs ne sont toutefois pas suffisants pour représenter des données complexes non structurées. D'un côté les mécanismes

de contrôle de ces types de données sont inexistants, d'un autre côté, le langage de requêtes (SQL) ne possède pas les opérateurs correspondant aux objets stockés dans ces BLOBs [3].

- **Langage de manipulation**

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

- **Scalabilité limitée**

Atteinte par les groupes comme Yahoo, Google, Facebook, etc. Cette limite est la plus gênante pour les entreprises. 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 sites.

- **Problème lié à l'application des propriétés ACID en milieu distribué**

Les bases de données relationnelles s'appuient sur les propriétés ACID. Ces propriétés bien que nécessaires à la logique du relationnel, elles nuisent fortement aux performances et en particulier à la propriété de cohérence. Cette dernière est très difficile à mettre en place dans les environnements distribués à plusieurs serveurs, puisque pour l'assurer, il faut que les serveurs doivent être des miroirs les uns des autres ; alors 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, la transaction n'est validée que si elle s'effectue sur tous les serveurs, ce qui fait patienter l'utilisateur durant ce temps [8].

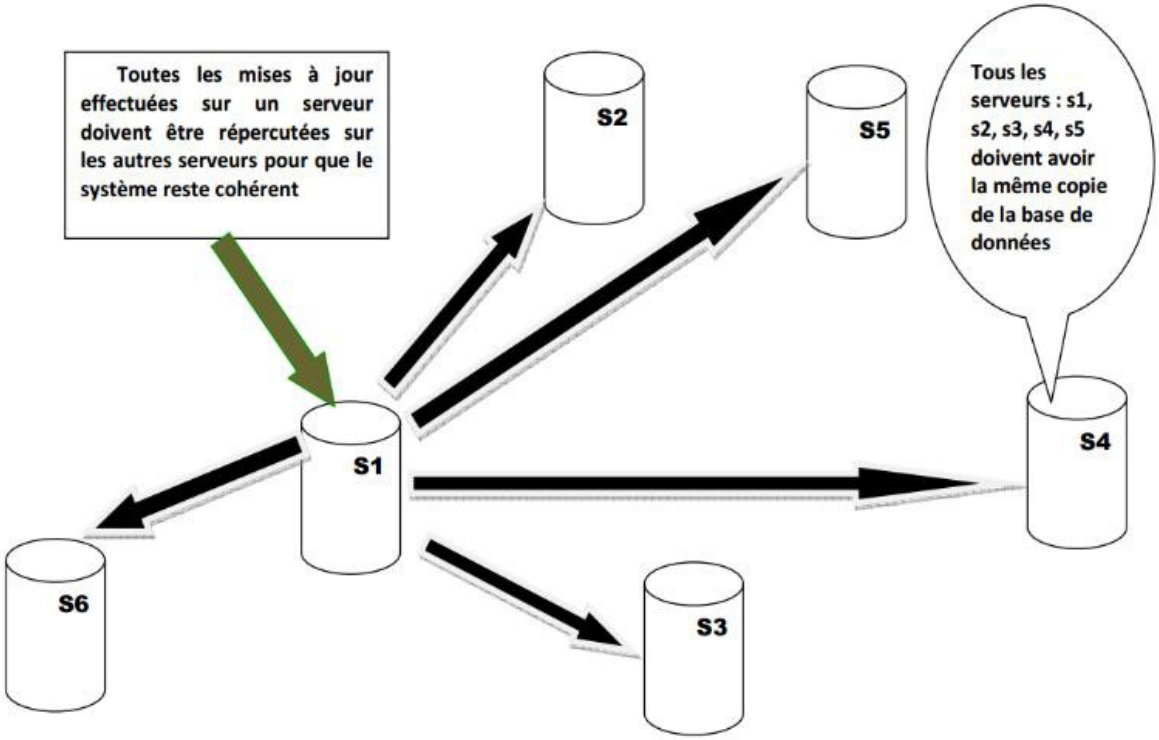



**Figure I.2 : Problème lié aux propriétés ACID en milieu distribué [1]**

## Conclusion

Les bases de données sont considérées comme les éléments les plus importants dans le monde informatique. Le modèle relationnel s'est imposé comme une norme, il s'est très répandu dans le marché grâce à son efficacité et sa robustesse. Cependant l'accroissement exponentiel des données, la prise en compte des données faiblement structurées et les avancées technologiques sont autant d'arguments qui poussent à la migration des SGBD relationnels vers une nouvelle façon de stockage et de manipulation des données. De ce fait, et face à ces contraintes, des nouveaux systèmes de gestion de données nommés «NoSQL» viennent de s'imposer en proposant des alternatives au modèle relationnel.

# Chapitre 2:

## Bases de données NoSQL

## Introduction

Face à l'évolution informatique dans la dernière décennie et notamment les grandes volumes de données échangés, les SGBDR classiques ont montré de très grandes faiblesses en matière de scalabilité et en montée en charge, ce qui a obligé les grands acteurs du web à chercher d'autre solutions. Les systèmes NoSQL viennent, justement, pour accomplir cette mission. Dans ce chapitre nous exposons les solutions NoSQL, leur émergence, les différents types, leurs avantages et leurs limites ainsi qu'un bref aperçu des solutions NoSQL les plus répandus dans le marché.

## II.1 Généralités

Les bases de données relationnelles se prêtent mal aux exigences des applications massivement parallèles exploitant de grandes quantités de données. Les bases de données NoSQL (Not Only SQL) marquent une rupture assez brutale avec la manière de concevoir les schémas de données que l'on pratiquait depuis déjà quelques décennies. Spécifiquement dédiées aux applications orientées Internet, les bases de données NoSQL pallient ainsi aux difficultés de la gestion de bases de données relationnelles un peu trop conséquentes et réparties sur plusieurs machines [9].

NoSQL est une combinaison de deux mots : No et SQL, qui pourrait être mal interprétée car l'on pourrait penser que cela signifie la fin du langage SQL et qu'on ne devrait donc plus l'utiliser. En fait, le « No » est un acronyme qui signifie «Not Only », c'est-à-dire en français, « non seulement », c'est donc une manière de dire qu'il y a autre chose que les bases de données relationnelles [10].

Le terme NoSQL est pour la première fois proposé par Carlo Strozzi en 1998, lors de la présentation de son système de gestion de bases de données relationnelles open source. Il l'a appelé ainsi à cause de l'absence d'interface SQL pour interagir avec les bases de données. Les bases NoSQL peuvent bien entendu être utilisées avec du SQL, c'est pourquoi Strozzi précise qu'il est plus pertinent d'utiliser le terme « NoREL » car ces dernières s'abstraient du côté relationnel des données. Le mot réapparu en 2009 lorsqu'Eric Evans l'utilisa pour caractériser le nombre grandissant de bases de données non-relationnelles lors d'une présentation sur les bases de données distribuées open source [11].

## II.2Emergence du NoSQL

Le langage SQL, en tant que langage d'interrogation standardisé, règne en maître sur les bases de données relationnelles depuis quelques décennies.SQL est un langage d'interrogation dit de

4ème génération standardisé. Les bases de données relationnelles, étroitement associées au langage SQL, comportent intrinsèquement un certain nombre de règles d'organisation : les formes normales par exemple, pour garantir la robustesse du schéma relationnel. Ces règles sont particulièrement efficaces pour un mode de gestion de données classique, mais elles s'avèrent être un véritable obstacle pour le déploiement, le stockage et l'analyse des bases de données redondantes de grande envergure, comme le nécessite le Big Data. Il faut donc adopter un autre mode de gestion des données pour faciliter les analyses massives [9].

En effet, Il existe cependant certains critères déterminants pour basculer vers le NoSQL :

**a. Taille**

Le NoSQL est conçu pour supporter un nombre important d'opérations provenant d'un très grand nombre d'utilisateurs.

**b. Performance en écriture**

Celle-ci est l'intérêt principal du géant Google, Facebook, Twitter, gérant des masses de données qui augmentent considérablement chaque année, et exigeant un temps très important pour stocker ces données. En conséquence, l'écriture se doit être distribuée sur un cluster de machines, ce qui implique du MapReduce, la réplication, la tolérance aux pannes et la cohérence [11].

**c. Type de données flexibles**

Parmi les innovations majeures, c'est le fait que les solutions NoSQL supportent de nouveaux types de données. Les objets complexes peuvent être mappés sans trop de problèmes avec la bonne solution [11].

**d. Scalabilité**

La scalabilité est le terme utilisé pour définir l'aptitude d'un système à maintenir un même niveau de performance face à l'augmentation de charge ou de volumétrie de données, par augmentation des ressources matérielles [10].

L'extensibilité est résolue de deux manières différentes : la scalabilité horizontale (externe) ainsi que la scalabilité verticale (interne).

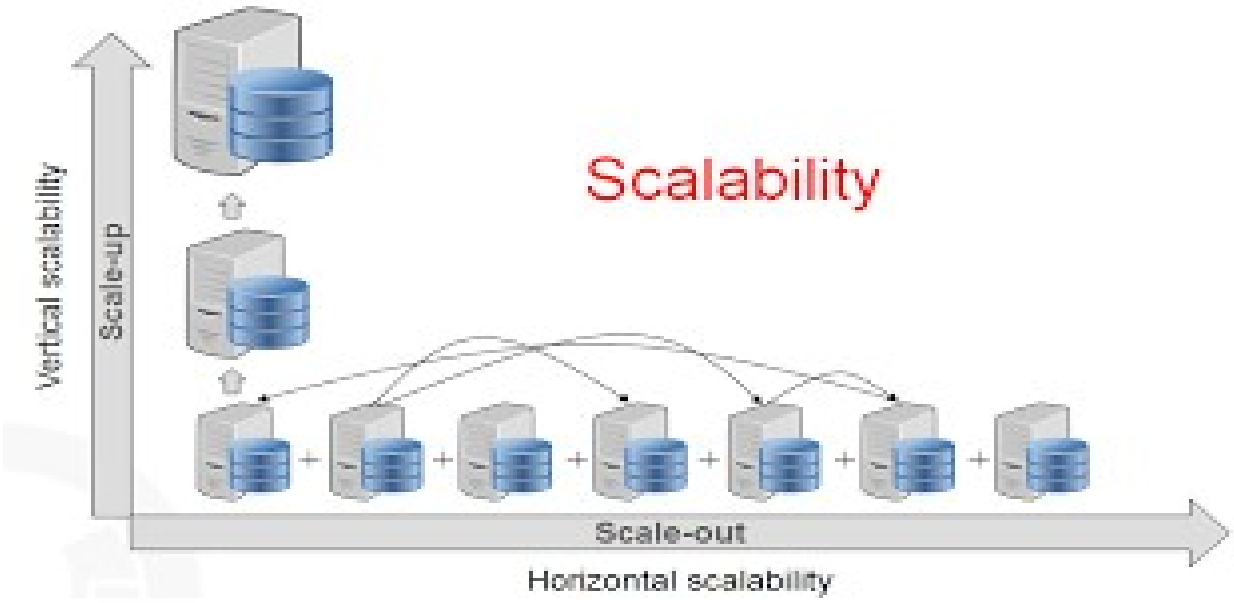



**Figure II.1 : Scalabilité [12]**

- **Scalabilité horizontale**

Cette croissance externe consiste à rajouter en parallèle des machines supplémentaires. On part d'une machine et on rajoute au fur et à mesure l'augmentation de charge de nouvelles machines identiques [13].

- **Scalabilité verticale**

Cette croissance interne consiste à rajouter des ressources supplémentaires à la machine (CPU, RAM, disque, carte réseau…). On part d'une machine conçue pour évoluer et on rajoute au fur et à mesure de l'augmentation de la charge des ressources spécifiques [13].

Le but des systèmes NoSQL est de renforcer la scalabilité horizontale, les SGBD NoSQL sont basés sur des systèmes innovants permettant la scalabilité horizontale et dont la plupart d'entre eux sont Open Source. Ils sont conçus pour répondre à des besoins spécifiques et assurer une extensibilité extrême sur de très grands ensembles de données [10].

## II.3Défis et exigences

Les entreprises adoptent NoSQL pour faire face à une nouvelle série de défis et d'exigences techniques qui sont le résultat de cinq grandes tendances :

| Tendance économie numérique | Exigences |
|---|---|
| **Plus de clients en ligne** | -Mise à l'échelle pour soutenir des milliers, voire des millions d'utilisateurs<br>-Le maintien de la disponibilité : 24 heures par jour, 7 jours par semaine |
| **Tout est connecté** | -Soutenir beaucoup de données avec différentes structures.<br>-Soutenir les mises à jour matérielles / logicielles, générer des données différentes.<br>-Soutien des flux continus de données en temps réel. |
| **Emergence du Big Data** | -Le stockage client génère des données semi-structurées / non structurées.<br>-Le stockage des différents types de données provenant de différentes sources.<br>-Le stockage des données générées par des milliers / millions de clients / objets. |
| **Les applications se déplacent vers le nuage** | -Mise à l'échelle de la demande pour soutenir plus de clients, stocker plus de données.<br>-La réduction des coûts d'infrastructure, la réalisation d'un temps plus rapide sur le marché. |
| **Le monde est devenu mobile** | -La synchronisation des données mobiles avec des bases de données à distance dans le nuage.<br>-Supporter de multiples plates-formes mobiles d'un seul arrière-plan. |

**Tableau 2.1 : Tendances et exigences techniques [14]**

## II.4Théorème CAP

CAP est l'acronyme de « Consistency, Availability and Partition tolerance », ou cohérence, Disponibilité et Résistance au partitionnement. Ce théorème formulé par Eric Brewer en 2000,et démontré par Seth Gilbert et Nancy Lych en 2002,affirme qu'un système d'information à calcul distribué ne peut satisfaire que deux, parmi les trois contraintes suivantes :

### a. Cohérence (Consistency)

A un instant donné, tous les nœuds du système voient les mêmes données.

### b. Disponibilité (Availability)

Les requêtes d'écriture et de lecture sont toujours satisfaites.

### c. Résistance au partitionnement (Partition tolérance)

La seule raison qui pousse un système à l'arrêt est la coupure totale du réseau. Autrement dit, si une partie du réseau n'est pas opérationnelle, cela n'empêche pas le système de répondre.

Afin de créer une architecture distribuée, on est amené à choisir deux de ces propriétés, laissant ainsi trois designs possibles :

**-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.

**-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. Toutes les lectures/écritures des 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) [11].

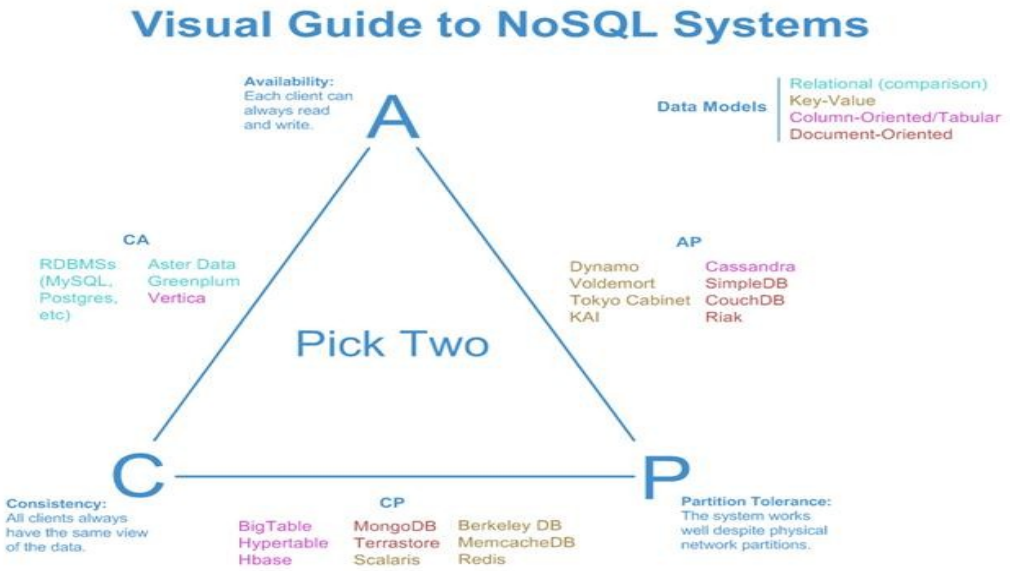



**Figure II.2 : Guide visuel au NoSQL [11]**

## II.5Propriétés B.A.S.E

Les propriétés BASE se considère comme une alternative à ACID. Ces dernières sont faites pour garantir l'intégrité des données, auxquelles doivent répondre les SGBD relationnels. Les SGBD NoSQL qui, selon le théorème CAP, privilégient la disponibilité ainsi que la tolérance à la partition plutôt que la cohérence, répondent parfaitement aux propriétés BASE. Le principe de BASE est le fruit d'une réflexion menée par Eric Brewer.

Les caractéristiques de BASE sont fondées sur les limites des SGBD relationnels.

a. **Disponibilité basique (Basically Available)**

Même en cas de désynchronisation ou de panne d'un nœud du cluster, le système reste disponible selon le théorème CAP.

b. **Cohérence légère (Soft-state)**

Cela indique que l'état du système risque de changer au cours du temps. Cela vient du fait que le modèle est cohérent à terme.

c. **Cohérence à terme (Eventual Consistency)**

Cela indique que le système devient cohérent dans le temps, pour autant que pendant ce laps de temps, le système ne reçoive pas d'autres données [10].

## II.6Types de bases de données NoSQL

Il existe une diversité d'approches NoSQL classées en quatre catégories. Ces systèmes utilisent des technologies distinctes spécifiques aux différentes solutions:

- **Orienté clé-valeur**
- **Orienté colonnes**
- **Orientée documents**
- **Orientée graphes**

Ces différents modèles de structure sont décrits comme suit :

### II.6.1 Bases de données orientée clé-valeur

La base de données de type clé-valeur est considérée comme la plus élémentaire. Son principe est très simple, chaque valeur stockée est associée à une clé unique. C'est uniquement par cette clé qu'il sera possible d'exécuter des requêtes sur la valeur. [10]

La communication avec la base de données se résume aux opérations basiques qui sont :

PUT, GET, UPDATE et DELETE [10].

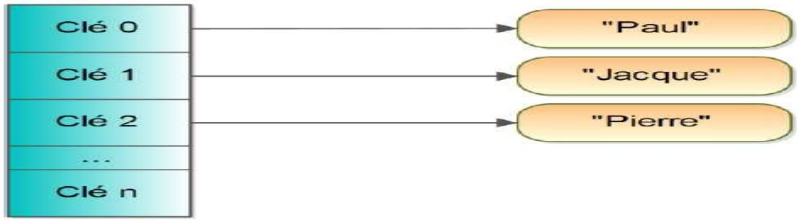



**Figure II.3 : Bases de données clé-valeur [10]**

Ces bases de données sont caractérisées par leur non utilisation du SQL et offrent une meilleure scalabilité horizontale ainsi elles peuvent adopter une architecture distribuée et tolérante aux pannes. En ce qui suit des solutions exemples de ce type :

- **DynamoDB :** Solution d'Amazon à l'origine de ce type de base. Système ultra scalable basé sur des disques électroniques SSD (Solid State Drive) rapides. Design de type AP selon le théorème de CAP mais peut aussi fournir une consistance éventuelle[11].
- **Voldemort :** Proposé par Dynamo, c'est une implémentation open-source qui offre une Très grande scalabilité.

## II.6.2 Bases de données orientée colonnes

Les bases de données orientées colonnes 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.
Ce modèle ressemble à première vue à une table dans un SGBDR à la différence que le nombre de colonnes est dynamique.
En effet, dans une table relationnelle, le nombre de colonnes est fixé dès la création du schéma de la table et ce nombre reste le même pour tous les enregistrements dans cette table. Par contre, avec ce modèle, le nombre de colonnes peut varier d'un enregistrement à un autre, ce qui évite de retrouver des colonnes ayant des valeurs NULL.

Les concepts de base de cette architecture sont :

- **Column** : L'entité de base qui représente un champ de données. Toutes les colonnes sont définies par un couple clé/valeur**.**
- **Supercolumn** : Une colonne qui contient d'autres colonnes**.**
- **Columnfamily** : Considérée comme un conteneur de plusieurs colonnes ou super-colonnes.
- **Row** : L'identifiant unique de chaque ligne de colonne.

- **Value** : Contenu de la colonne qui peut, elle-même, être une colonne.

Ce type de structure permet d'être plus évolutif et flexible ; cela vient du fait qu'on peut ajouter à tout moment une colonne ou une super-colonne à n'importe quelle famille de colonnes, nous citons comme exemple :

- **HBase :** Utilise un API Java. Adopte un design CA.

- **Cassandra :** Beaucoup d'API disponibles. Adopte un design AP avec consistance éventuelle.

### II.6.3 Bases de données orientée documents

Les systèmes de type documentaire sont composés de collections de documents. La représentation en document est une sorte d'extension du concept clé/valeur. La valeur est représentée sous forme de document contenant des données. Des champs et des valeurs associées composent le document. Ces valeurs associées peuvent soit être de type simple (integer, string, date, ...), soit de type composé, c'est-à-dire de plusieurs couples clé/valeur. Bien que les documents soient organisés de manière hiérarchique à l'image de ce que permettent les fichiers de type XML ou JSON, ces bases sont dites «Schémaless», ce qui signifie sans schéma défini. Cela veut tout simplement dire qu'il n'est pas nécessaire de définir au préalable les champs dans le document : on peut très bien en rajouter en cours de développement. Les documents peuvent être très différents les uns des autres au sein de la base [10].

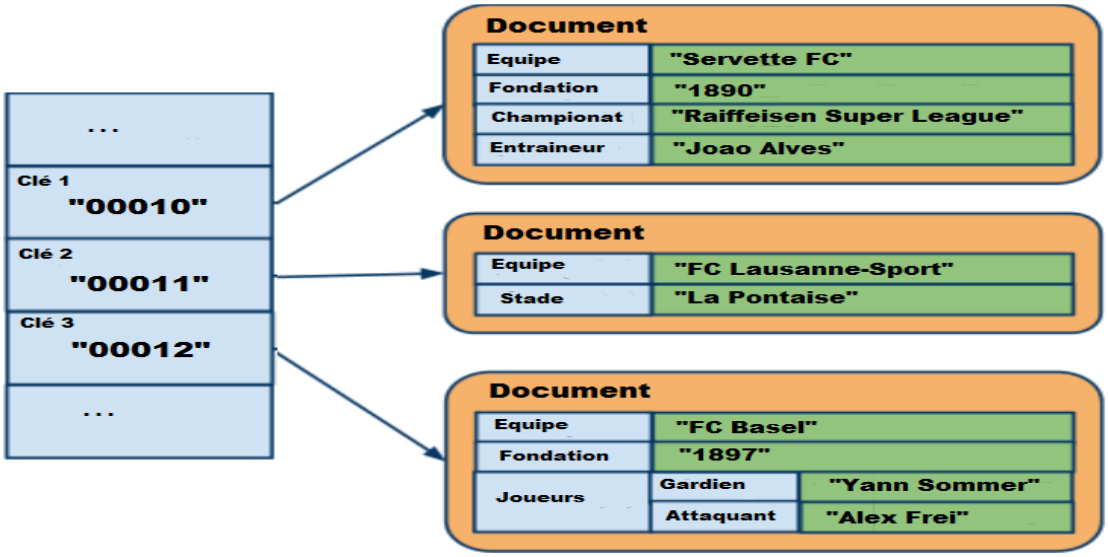



**Figure II.4 : Schéma d'une base de données orientée documents [10]**

Parmi ces solutions:

- **MongoDB :** Développé en C++, API officielles pour beaucoup de langages, Protocole personnalisé BSON, Réplication master/slave et auto-sharding, Licence commerciale et gratuite.

- **CouchDB :** Développé en Erlang, Protocol http, Réplication master/master, Licence Apache.

### II.6.4 Bases de données orientée graphes

Bien que les bases de données de type clé-valeur, colonne, ou document tirent leur principal avantage de la performance du traitement de données, les bases de données orientées graphe permettent de résoudre des problèmes très complexes qu'une base de données relationnelle serait incapable de faire [10].

Ce modèle se base sur la théorie des graphes et s'appuie sur la notion de nœuds, de relations et de propriétés qui leur sont rattachées. Ce modèle facilite la représentation du monde réel, ce qui le rend adapté au traitement des données des réseaux sociaux.

Ces bases permettent la modélisation, le stockage ainsi que le traitement de données complexes reliées par des relations. Ce modèle est composé d'un :

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

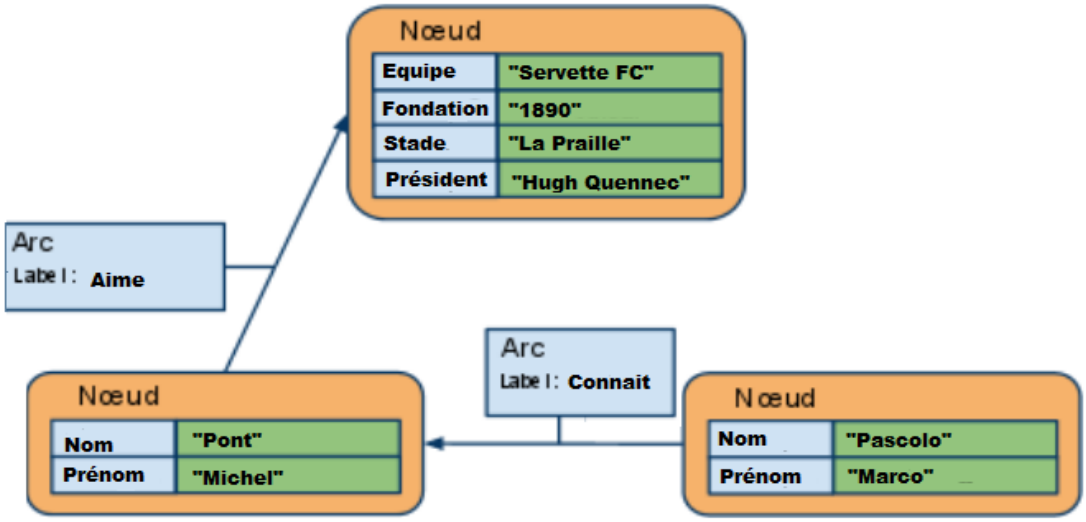



Figure II.5: Schéma d'une base de données Orientée graphe

Exemple d'implémentation :

- **Neo4J :** Développé en Java, supporte beaucoup de langages, propriétés ACID possibles, utilisé dans le monde des réseaux sociaux (Ex : amis sur Facebook) ; Langage de requêtes personnalisé « Cypher ».

## II.7Avantages et inconvénients des bases de données NoSQL

### II.7.1 Avantages

- **La « scalabilité » horizontal**

Pendant longtemps, les administrateurs de bases de données ont opté pour la « scalabilité » verticale, au lieu d'avoir misé sur une « scalabilité » horizontale. Ceci vient du fait qu'il est difficile d'adapter cette stratégie avec une base de données relationnelle.

Aujourd'hui, la rapidité en lecture/écriture ainsi que la haute disponibilité sont devenues des critères indispensables. C'est pourquoi les bases de données NoSQL répondent entièrement à ce besoin. Les performances qu'offre-la « scalabilité » horizontale peuvent même être atteintes avec des serveurs bas de gamme, ce qui rend l'infrastructure plus économique.

- **Gros volume de données « Big data »**

Au cours de la dernière décennie, le volume de données à stocker a augmenté de manière massive. Les bases de données relationnelles ont augmenté leurs capacités afin de suivre la tendance. Mais avec cette constante augmentation de volume de données, il est devenu quasi impossible, pour un unique serveur de base de données relationnelle, de répondre aux exigences des entreprises en terme de performance.

Aujourd'hui, ces gros volumes de données ne sont plus un problème pour les SGBD de type NoSQL et même le plus grand des SGBD relationnel ne peut rivaliser avec une base NoSQL.

- **Administrateur de bases de données (DBA) moins indispensable**

Malgré les nombreuses améliorations vantées, pendant des années par les fournisseurs de SGBD relationnels, concernant la gestion de bases de données, un SGBD relationnel haut de gamme demande une certaine expertise pour sa maintenance. C'est la raison pour laquelle on fait généralement appel à un DBA, ce qui ajoute des coûts supplémentaires. Les DBA sont intimement impliqués dans la conception, l'installation ainsi que dans la configuration des SGBD relationnels haut de gamme. Il est erroné de dire que la présence d'un DBA n'est plus requise pour gérer une base de données NoSQL, mais ses tâches seront facilitées notamment grâce à la distribution des données et surtout grâce à la simplicité du schéma de données. Il

y'aura toujours une personne responsable des performances ainsi que de la disponibilité quand des données critiques sont en jeu.

- **Solution économique**

Les bases de données NoSQL ont tendance à utiliser des serveurs bas de gamme dont le coût est moindre afin d'équiper les « clusters », tandis que les SGBD relationnels, eux, tendent à utiliser des serveurs ultra puissants dont le coût est extrêmement élevé. De ce fait, les systèmes NoSQL permettent de revoir à la baisse les coûts d'une entreprise. Cela permet de stocker ainsi que de manipuler plus d'informations à un coût nettement inférieur.

- **Modèle de données flexible**

Changer le modèle de données d'une base de données relationnelle en production, est un vrai casse-tête, même une petite modification doit être maniée avec précaution et peut nécessiter l'arrêt du serveur pendant la modification ou limiter les niveaux de services.

Les systèmes NoSQL sont plus souples en termes de modèles de données, comme dans les catégories clé/valeur et documentaire. Même les modèles un peu plus stricts comme dans la catégorie orientée colonne permettent d'ajouter une colonne assez facilement.

### II.7.2 Inconvénients

- **Maturité**

Les SGBD relationnels sont là depuis plus de 30 ans, ainsi, cette pérennité est un signe de réconfort pour les responsables. Les SGBD relationnels sont de nature stable et riche en fonctionnalités. En comparaison, la plupart des SGBD NoSQL sont en pré-production et ont encore beaucoup de fonctionnalités futures à implémenter.

Les spécialistes parlent d'une attente d'une bonne dizaine d'années avant de voir ce concept utilisé fréquemment par des applications critiques.

- **Support**

Les entreprises veulent être rassurées de leurs bases de données et cherchent un support rapide et efficace. Les principaux fournisseurs de SGBD relationnels offrent un haut niveau de support pour les entreprises. Bien que certains fournisseurs NoSQL proposent des supports assez fiables, ils ne peuvent, bien évidemment, pas rivalisé avec un support mondial comme peuvent le faire Oracle, IBM ou Microsoft.

- **Expertise**

Il existe des millions de développeurs dans le monde, opérant, dans différents businesses, qui se sont familiarisés avec les concepts ainsi que de la programmation dans un environnement

de bases de données relationnelles. Dans le monde NoSQL, presque tous les développeurs sont en apprentissage avec la technologie.

Avec le temps, cette situation risque de changer, mais pour le moment, il est plus simple de trouver une personne ayant une grande expérience des bases de données relationnelles qu'un expert NoSQL [10].

## Conclusion

Les bases de données NoSQL ont gagné en popularité ces dernières années parce qu'elles ont pu venir à bout de nouveaux problèmes que les bases de données relationnelles étaient incapables de le résoudre. On assiste à une explosion de systèmes NoSQL disponibles sur le marché, avec chacun ses propres propriétés et cas d'utilisations. Un administrateur doit par conséquent examiner soigneusement le type de bases de données le mieux adapté à ses besoins, car un mauvais choix pourrait avoir de très lourdes conséquences.

# Chapitre 3 :

## Etude comparative

## MySQL vs MongoDB

# Introduction

Ce chapitre va faire l'objet d'une étude comparative des performances de deux SGBD d'architectures et de générations différentes. Cette section va être entamée par une présentation de l'outil de comparaison qui va être utilisée dans cette étude, à savoir le benchmark YCSB d'une part, et d'autre part, les deux solutions MongoDB (NoSQL) et MySQL (SQL) étudiées et comparées, sont représentées aussi. Une partie est réservée aux différentes installations et configurations des outils employés. Pour en finir, les résultats expérimentaux des différents tests exécutés, seront présentés, comparés et analysés.

Cette étude a été réalisée dans un environnement monoposte caractérisé par les aspects Hardware et Software suivants :

Software

- Système d'exploitation Linux Mint 17.1 «Rebecca»
- Java version «1.8.0_121»
- Apache Maven 3.2.5

Hardware

- PC portable : LENOVO B590, CPU Intel Core (TM) i5 -3230 M 2.60 GHz, 4 Go RAM, DD 500 Go

# III.1 Présentation du benchmark et les SGBD étudiés

## III.1.1Outil de comparaison YCSB

Pour notre étude comparative développée, l'indice de référence Yahoo Cloud Serving Benchmark(YCSB) version YCSB 0.5.0, va être utilisé. Le noyau d'YCSB contient un générateur de données et permet de créer des charges de travail «Workloads» sous forme de différents scénarios de tests.

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, dont voici celles qui vont être utilisées :

- Workload A : 50% de lectures, 50% de mises à jour.

- Workload B : 95% de lectures, 5% de mises à jour.
- Workload C: 100% de lectures.
- Workload D: 95 % lecture5 % insertion.
- Workload E: 95 % scan 5 % insertion.
- Workload F: 50% lecture50 % Read-Modify-Write.

Deux autres charges additionnelles ont été proposées :

- Workload G: 5 % lecture95 % Read-Modify-Write.
- Workload H: 100% Read-Modify-Write.

### III.1.2 MySQL

MySQL est un système de gestion de bases de données relationnelles, distribué sous une double licence GPL et propriétaire. Il fait partie des logiciels de gestion de base de données les plus utilisés au monde, autant par le grand public (applications web principalement) que par des professionnels, en concurrence avec Oracle, Informix et Microsoft SQL Server [13].

C'est un logiciel libre, open source, développé sous double licence selon qu'il est distribué avec un produit libre ou avec un produit propriétaire. Dans ce dernier cas, la licence est payante, sinon c'est la licence publique générale GNU (GPL) qui s'applique. Un logiciel qui intègre du code MySQL ou intègre MySQL lors de son installation devra donc être libre ou acquérir une licence payante.

MySQL fonctionne sur de nombreux systèmes d'exploitation différents :
AIX, BSDi, FreeBSD, HPUX, Linux,MacOS, NetWare, NetBSD, OpenBSD ,OS/2 Warp, SGI IRIX, Solaris, SCO OpenServer,SCO UnixWare, Tru64 Unix, Windows etc…
Une base de données MySQL est accessible en utilisant les langages de programmation :
C, C++, VB, NET, C#, Delphi/Kylix, Eiffel, Java, Perl, PHP, Python, WinDev, Ruby et Tcl.

### III.1.3 MongoDB

MongoDB (d'origine « humongous » ie « énorme ») est un système NoSQL de type orienté document. C'est une solution open source, écrit en C++, proposé par la société10gen, développé depuis Octobre 2007. Sa popularité est grandissante, vu sa simplicité d'utilisation du point de vue de développement client, ainsi que ces performances remarquables. Il est flexible et peut fonctionner parfaitement avec des masses de données importantes. Il gère des

collections de documents JSON (JavaScript Object Notation), équivalentes à des tables dans MySQL, stockées dans un format binaire (BSON).MongoDB propose le modèle de données suivant [15] :

### III.1.3.1 Modèle de données

- **Documents**

Un document représente l'unité basique de MongoDB, composé d'un ensemble ordonné de paires clé-valeur.

Pour faire une analogie à SQL, un document peut être vu comme un Uplet. A la différence d'une entrée SQL, le nombre de champs d'un document n'est pas défini au préalable pour une même collection, il peut varier d'un document à l'autre.

Un exemple très simple d'un document :

```
{"nom" : "Benali", "prenom" : "Ahmed", "age" : 30}
```

- **Collections**

Une collection est un ensemble de documents équivalent à une table SQL. Les documents d'une même collection peuvent avoir des formes très différentes comme l'illustre l'exemple suivant :

```
{"nom" : "Benali", "prenom" : "Ahmed", "age" : 30}
{"nom" : "Safi", "prenom" : "Mostapha", "age" : 30 ,"spécialité" : "informatique"}
{"marque" : "Lenovo", "nom" : "Laptop"}
```

### III.1.3.2 Architecture

Une base de données MongoDB a deux propriétés importantes en termes d'architecture interne :

- Elle peut être répliquée, pour éviter toute perte de données.
- Elle est extrêmement scalable, par le biais d'une approche qu'on appelle le sharding.

Les deux architectures sont employées dans un contexte « Big Data » ; la première pour éviter toute perte de données critique, et la seconde parce qu'on est évidemment susceptible de stocker de très larges quantités de données au sein des bases de données [16].

## III.2 Installations et configurations

### III.2.1 Pré-requis

1. Installation du JAVA et Maven
2. Modification du contenu fichier « **maven.sh** » en exécutant la commande suivante :

```
sudo vi /etc/profile.d/maven.sh
export M2_HOME=/usr/local/maven
export PATH=${M2_HOME}/bin :${PATH}
```

3. Définition du chemin de JAVA_HOME :

```
JAVA_HOME="/home/abdou/Documents/java/jdk1.8.0_121"
```

### III.2.2 Installation du benchmark YCSB

Voici dans ce qui suit les étapes d'installation et configuration du benchmark YCSB:

1. Téléchargement d'YCSB en utilisant la commande :

```
curl -O --location
https://github.com/brianfrankcooper/YCSB/releases/download/0.5.0/ycsb-
0.5.0.tar.gz
```

2. Décompression du dossier ycsb-0.5.0.tar.gz avec la commande suivante :

```
tar xfvz ycsb-0.5.0.tar.gz
```

### III.2.3 Installation et configuration de MySQL

Nous avons utilisé MySQL Server version 5.5.54 :

1. Télécharger et installer MySQL en exécutant la commande suivante :

```
sudo apt-get install MySQL-server
```

   - Spécifier un mot de passe pour l'utilisateur « root » pendant l'installation.
   - Télécharger MySQL-connector-java-xyz.jar
   - Le décompresser dans le dossier ycsb-0.5.0/jdbc-binding/lib/
2. Créer une base de données nommée**« datpfe »** :

```
MySQL>CREATEDATABASE datapfe ;
```

   - Dans la base de données «datapfe», créer une table nommée

**« usertable »**ayant le schéma suivant :

```
CREATE TABLE usertable (YCSB_KEY VARCHAR (255) PRIMARY KEY,
    FIELD0 TEXT, FIELD1 TEXT,
    FIELD2 TEXT, FIELD3 TEXT,
    FIELD4 TEXT, FIELD5 TEXT,
    FIELD6 TEXT, FIELD7 TEXT,
    FIELD8 TEXT, FIELD9 TEXT
);
```

3. Copier le fichier **db.properties** dans le dossier **ycsb-0.5.0** et le modifier comme suit :

```
jdbc.driver=com.MySQL.jdbc.Driver
jdbc.fetchsize=20db.url=jdbc:MySQL://localhost:3306/datapfe    #datapfe est le
nom de la base de données dans MySQLdb.user=rootdb.passwd=passe
```

### III.2.2 Installation MongoDB

Pour l'étude développée, on a utilisé la version MongoDB 2.6.11.

1. Décompresser MongoDB-linux-i686-2.6.11.tgz à l'aide de la commande suivante :

```
tar xfvz mongodb-linux-i686-2.6.11.tgz
```

2. Installation de MongoDB

```
mkdir /tmp/mongodb
mkdir data
mkdir /data/db
cd mongodb-linux-i686-2.6.11/bin

 mongodb-linux-i686-2.6.11/bin/mongod --dbpath /tmp/mongodb
```

3. Utiliser la commande « **mongod** » pour se connecter au serveur mongo au port **27017** et la commande « **mongo** » pour exécuter le Shell mongo.

## III.3Résultatsexpérimentaux

Les différents tests effectués ont été variés selon les trois axes suivants :

1. le premier concerne la nature des opérations exécutées dans une charge de travail.
2. le deuxième concerne le nombre d'enregistrements chargés (1000, 5000, 10000 et 600000 enregistrements)
3. le troisième concerne le nombre d'opérations exécutés (100 et 1000 opérations).

Chaque enregistrement est de 1000 octets et contient 10 champs. Une clé primaire sous forme de chaine identifie chaque enregistrement, telle que «user234123». Chaque champ est nommé Field0, Field1 et ainsi de suite. Les valeurs dans chaque champ sont des chaînes aléatoires de caractères ASCII, 100 octets chacun.

La performance des bases de données a été définie à base de la vitesse d'exécution des opérations. Chaque charge de travail va être lancée au moins 3 fois dans 3 jours différents pour garder seulement la moyenne des temps d'exécutions obtenus. Pour un SGBD (NoSQL ou relationnel), chaque fil exécute une série séquentielle d'opérations en faisant appel à la couche d'interface de base de données à la fois pour charger la base de données (la phase de chargement) et pour exécuter la charge de travail (phase de transaction).

### III.3.1Chargement des données

Avant d'exécuter les différentes workloads, il faut faire un chargement des données en indiquant le nombre d'enregistrement à charger, en lançant la commande suivante :

- Pour MySQL :

  ```
  abdou@abdou ~/ycsb-0.5.0 $ bin/ycsb load jdbc -P Workloads/Workloada -p
  recordcount=600000 -P db.properties –cp MySQL-connector-java-5.1.41-bin.jar-s >
  loadjdbc600000.txt
  ```

- Pour MongoDB :

  ```
  abdou@abdou ~/ycsb-0.5.0 $ bin/ycsb load mongodb -P Workloads/Workloada -p
  recordcount=600000 -t -s > loadmongo600000.txt
  ```

La commande précédente permet de charger 600 000 enregistrements pour chaque SGBD.

Les temps de chargement pour chaque SGBD sont récupérés à partir des fichiers textes générés (loadjdbc600000.txt, loadmongo600000.txt) lors de l'exécution de load_process indiqué ci-dessus.

Le tableau suivant récapitule les temps de chargements générés par YCSB pour les deux systèmes, une fois un chargement de 1000, une autre fois de 5000, puis 10000 et enfin 600 000 enregistrements chargés :

| Nbre enregistrements chargés | 1 000 | | 5 000 | | 10 000 | | 600 000 | |
|---|---|---|---|---|---|---|---|---|
| SGBD | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql |
| Temps total d'excution | 1,03 sec | 56,34 sec | 1,94 sec | 293,56 sec | 3,63 sec | 611,84 sec | 146,68 sec | 10H 53 |
| | | | | | | | | |

**Tableau III.1: Chargement de données**

Dans ce premier test, nous pouvons constater que les meilleurs temps sont été inscrits par MongoDB qui était largement plus rapide par rapport à MySQL dans cette première phase de chargement. Dans le dernier chargement du plus grand nombre d'enregistrements (600000), les résultats de MySQL ont été très décevants (presque 11Heures pour le chargement).

## III.3.2Exécution des Workloads

Après le chargement des données, nous allons exécuter les charges de travail une après une :

### III.3.2.1WorkloadA (50% Read 50% Update)

Les commandes suivantes permettent d'exécuter le Workload A (50% read 50% update) :

- MySQL :

```
abdou@abdou ~/ycsb-0.5.0 $ bin/ycsb run  jdbc -P Workloads/Workloada -p
recordcount=600000 -P db.properties -cp MySQL-connector-java-5.1.41-bin.jar-s
>runjdbc600000.txt
```

- MongoDB :

```
abdou@abdou ~/ycsb-0.5.0 $ bin/ycsb run mongodb -P Workloads/Workloada -p
recordcount=600000 -t -s >runmongo600000.txt
```

Les résultats obtenus :

- Exécution du Workload A avec 100 opérations

| 100 opérations | A 50%read 50 %update | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Nbre enregistrements chargés | 1 000 | | 5 000 | | 10 000 | | 600 000 | |
| SGBD | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql |
| Temps total d'excution (sec) | 0,44 | 2,78 | 0,46 | 2,70 | 0,52 | 2,51 | 0,51 | 2,75 |

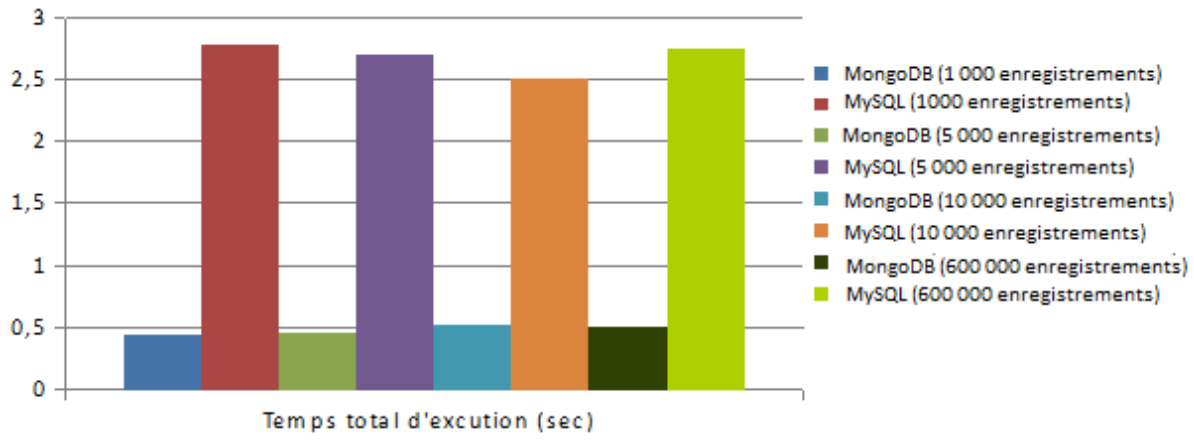



**Figure III.1:Workload A avec 100 opérations**

- Exécution du Workload A avec 1000 opérations :

| 1000 opérations | A 50%read 50 % update | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Nbre enregistrements chargés | 1 000 | | 5 000 | | 10 000 | | 600 000 | |
| SGBD | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql |
| Temps total d'excution (sec) | 1,03 | 27,93 | 1,06 | 26,88 | 0,97 | 26,46 | 1,08 | 26,48 |

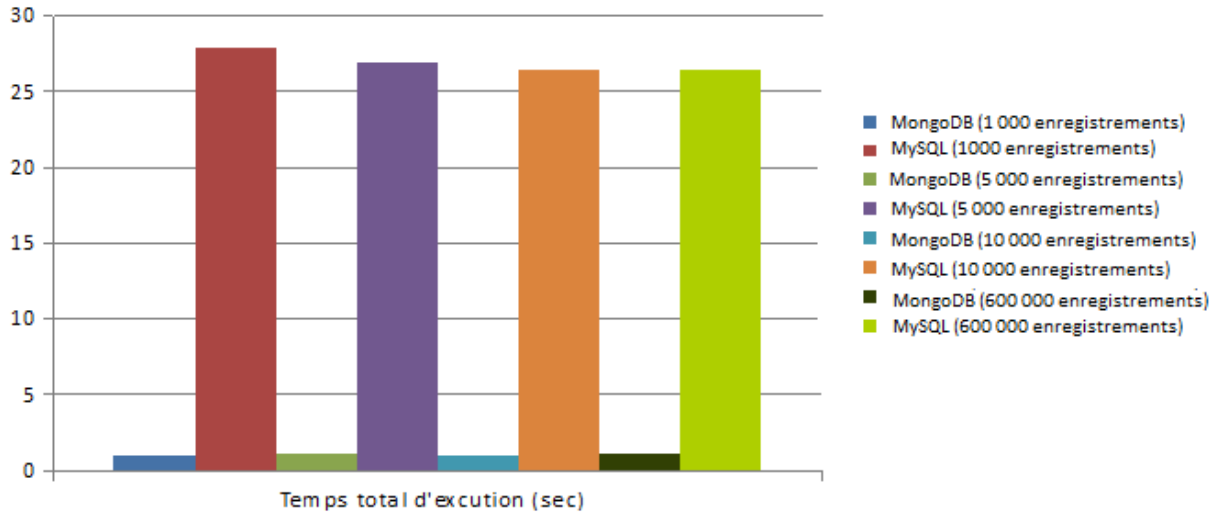



**Figure III.2: Workload A avec 1000 opérations**

Pour cette charge de travail avec des opérations moitié lecture et moitié mise à jour, les tests montrent que les meilleurs temps ont été obtenus par MongoDB, soit en exécutant 100 opérations, soit 1000 opérations. Ce constat reste le même pour les différents chargements faits (de 1000 à 600000 enregistrements chargés). En revanche, les performances de MySQL ont été au-dessous de MongoDB.

Nous affirmons ici, que MongoDB est plus performante que MySQL, mais pour pouvoir juger ça, il faut voir les résultats d'exécutions des charges de travail C (lecture seulement100 % Read) et H (mise à jour seulement 100 % Update), pour confirmer quels sont les types d'opérations qui sont entrain de dégrader les performances de MySQL.

Nous remarquons aussi que le nombre d'opérations à exécuter, influe clairement sur la performance de MySQL :le temps d'exécution augmente au fur et à mesure quand on augmente le nombre d'opérations à effectuer. Contrairement à MongoDB, ou la performance n'a pas été vraiment influencé par ces variations sur le nombre d'opérations.

### III.3.2.2Workload B (95 % Read, 5% Update)

- Exécution du Workload B avec 100 opérations :

| 100 Opérations | B 95% read 5% update | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Nbre enregistrements chargés | 1 000 | | 5 000 | | 10 000 | | 600 000 | |
| SGBD | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql |
| Temps total d'excution (sec) | 0,44 | 0,53 | 0,46 | 0,48 | 0,44 | 0,75 | 0,43 | 0,7 |

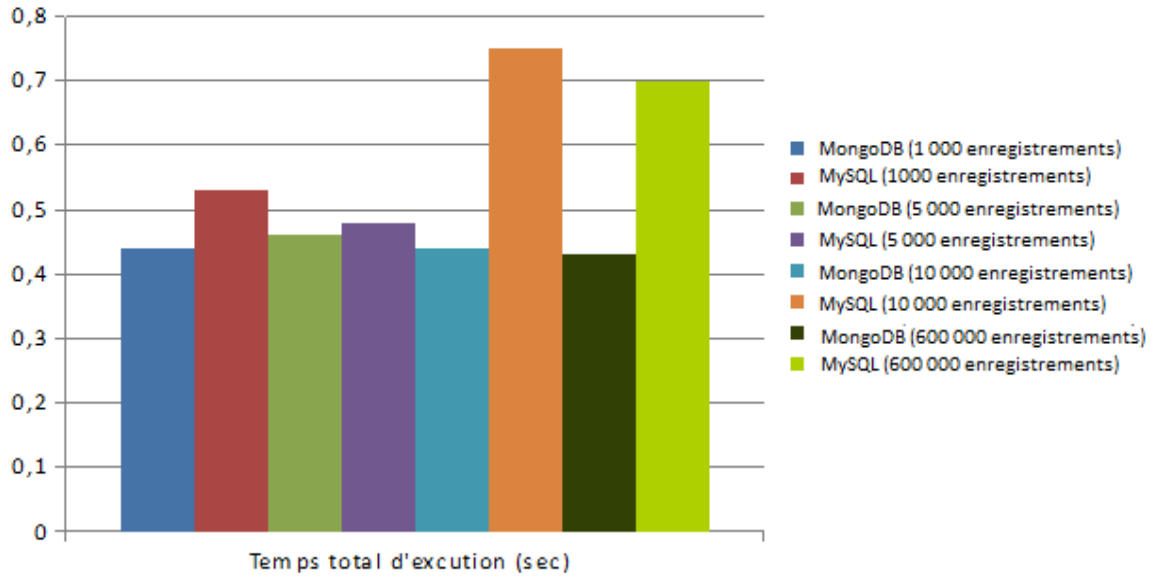



**Figure III.3 : Workload B avec 100 opérations**

- Exécution du Workload B avec 1000 opérations :

| 1000 Opérations | B 95% read 5% update | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Nbre enregistrements chargés | 1 000 | | 5 000 | | 10 000 | | 600 000 | |
| SGBD | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql |
| Temps total d'excution (sec) | 0,86 | 3,06 | 0,85 | 2,86 | 0,92 | 2,84 | 0,89 | 15,35 |

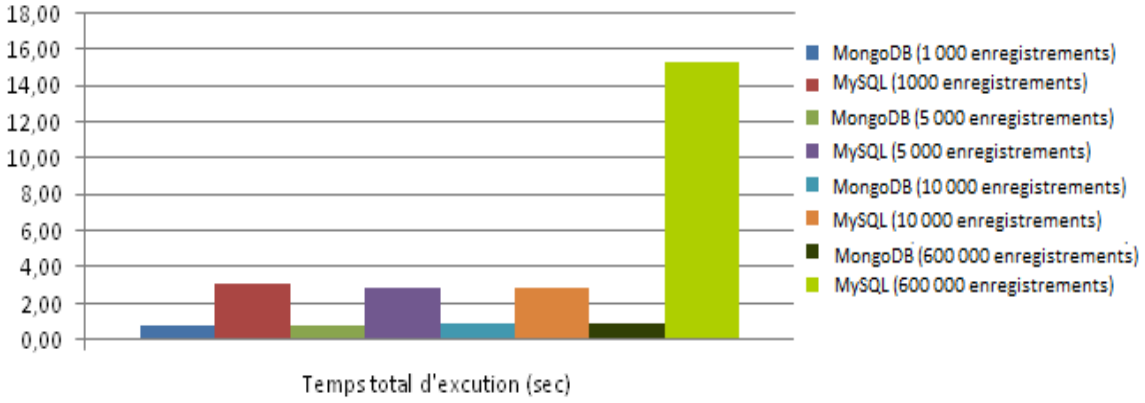



**Figure III.4 : Workload B avec 1000 opérations**

Pour cette charge de travail contenant majoritairement des opérations de lecture : dans le premier cas de 100 opérations, les performances des deux bases de données étaient bonnes en ayant pas dépassé une seconde. Pour le deuxième cas de 1000 opérations exécutées, nous pouvons remarquer que les meilleurs résultats sont été obtenus par MongoDB. Les opérations Update ont ralenti énormément MySQL, surtout avec une base de données volumineuse contenant 600 000 enregistrements.

### III.3.2.3Workload C (100 % Read)

- Exécution du Workload C avec 100 opérations :

| 100 opérations | C 100% Read | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Nbre enregistrements chargés | 1 000 | | 5 000 | | 10 000 | | 600 000 | |
| SGBD | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql |
| Temps total d'excution (sec) | 0,42 | 0,40 | 0,40 | 0,40 | 0,41 | 0,46 | 0,42 | 0,41 |

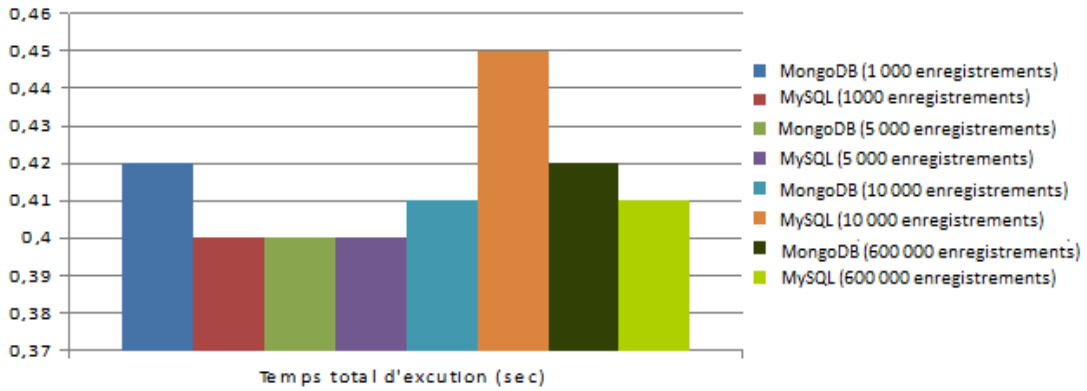



**Figure III.5 : Workloa**

MongoDB (5 000 enregistrements) MongoDB (5 000 enregistrements)
MySQL (5 000 enregistrements) MySQL (5 000 enregistrements)
MongoDB (10 000 enregistrements) MongoDB (10 000 enregistrements)
MySQL (10 000 enregistrements) MySQL (10 000 enregistrements)
MongoDB (600 000 enregistrements) MongoDB (600 000 enregistrements)
MySQL (600 000 enregistrements) MySQL (600 000 enregistrements)

- Exécution du Workload C avec 1000 opérations :

| 1000 opérations | C 100% Read | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Nbre enregistrements chargés | 1 000 | | 5 000 | | 10 000 | | 600 000 | |
| SGBD | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql |
| Temps total d'excution (sec) | 0,85 | 0,57 | 0,84 | 0,64 | 0,86 | 0,69 | 0,86 | 9,24 |

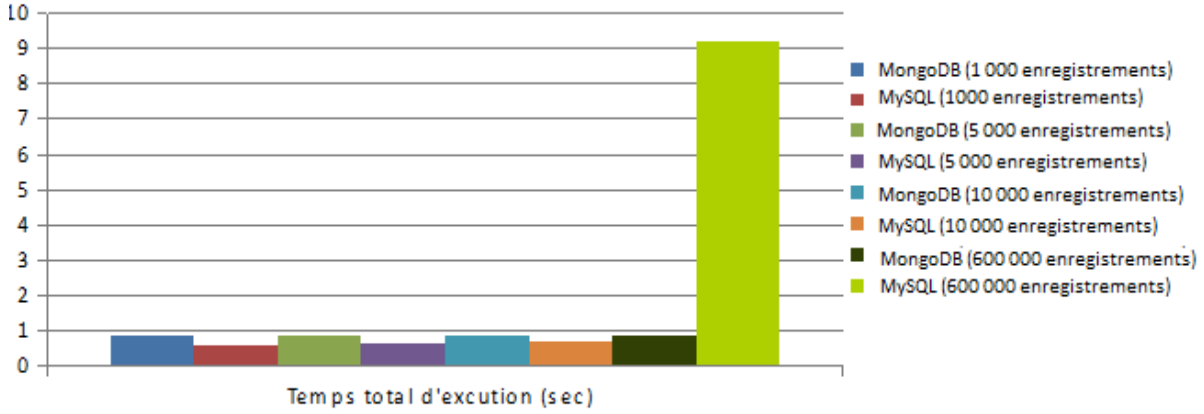



**Figure III.6 : Workload C avec 1000 opérations**

Pour une charge de travail contenant seulement des opérations de lecture : dans le cas de 100 opérations, les performances des deux bases de données étaient presque identiques en ayant pas dépassé une seconde. Pour le deuxième cas de 1000 opérations exécutées, nous pouvons remarquer que MySQL a pris l'ascendant sur MongoDB, sauf dans le cas d'une grande base de données composée de 600000 enregistrements ou MySQL était assez lente (9,24 sec). Notons aussi que les performances de MongoDB ont resté très stables vu les résultats obtenus qui sont presque similaires.

### III.3.2.4Workload D (95 % Read, 5% Insert)

- Exécution du Workload D avec 100 opérations :

D 95% Read 5% insert

| Nbre enregistrements chargés | 1 000 | | 5 000 | | 10 000 | | 600 000 | |
|---|---|---|---|---|---|---|---|---|
| SGBD | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql |
| Temps total d'excution (sec) | 0,43 | 0,59 | 0,43 | 0,73 | 0,45 | 0,58 | 0,47 | 0,66 |

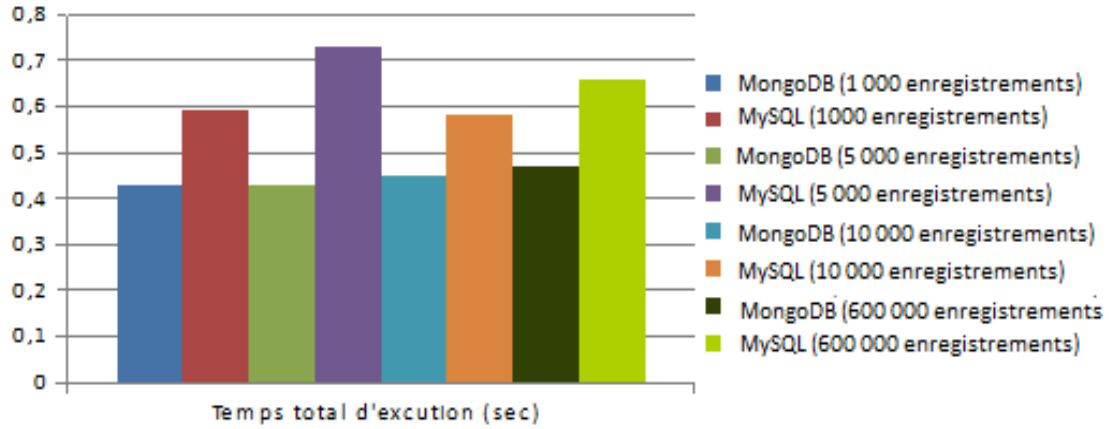



**Figure III.7 : Workload D avec 100opérations**

- Exécution du Workload D avec 1000 opérations :

D 95% Read 5% insert

| Nbre enregistrements chargés | 1 000 | | 5 000 | | 10 000 | | 600 000 | |
|---|---|---|---|---|---|---|---|---|
| SGBD | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql |
| Temps total d'excution (sec) | 0,90 | 3,33 | 0,89 | 3,60 | 0,93 | 2,85 | 0,91 | 10,76 |

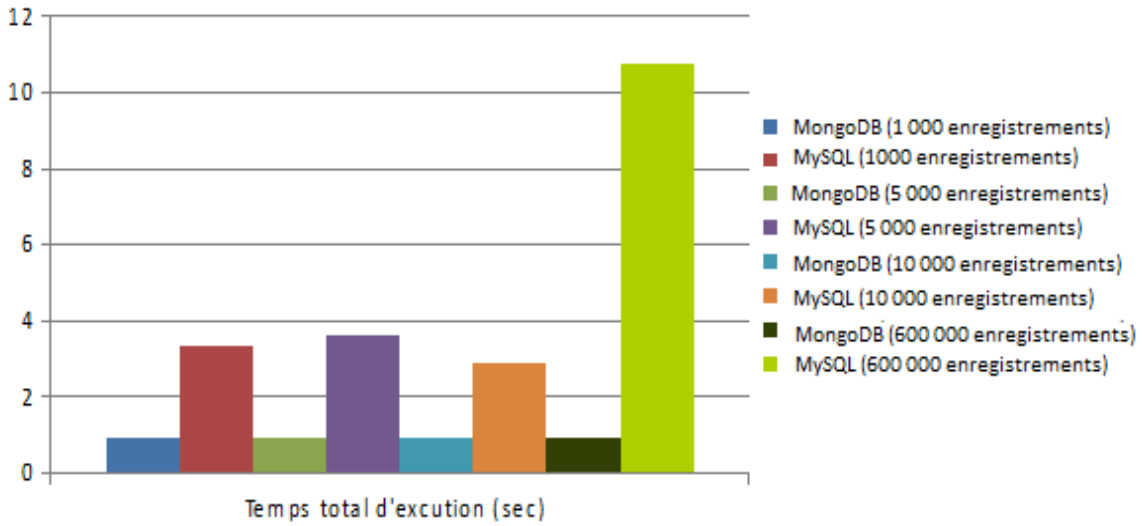



**Figure III.8:Workload D avec 1000 opérations**

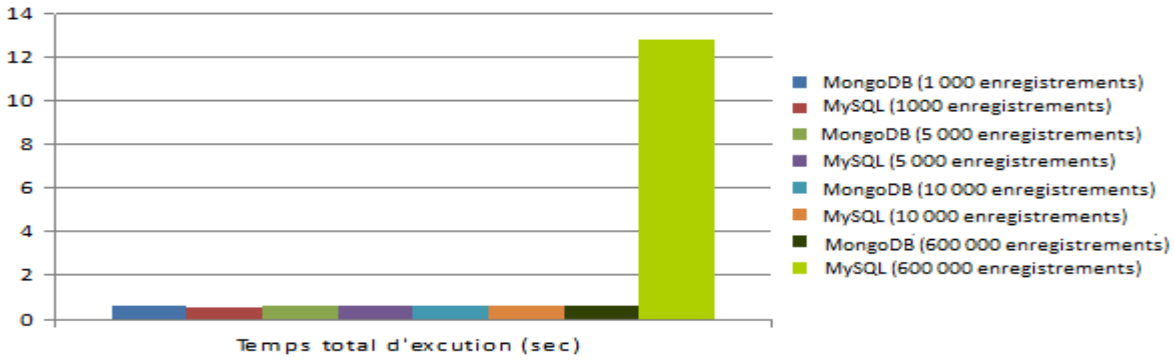
Les mêmes remarques faites pour la charge de travail B (95 % Read, 5% Update), peuvent être faite pour cette charge de travail, ou MySQL a été assez lente à cause des opérations d'écriture. En exécutant 1000 opérations dans une base de données de 600 000 enregistrements, l'influence des opérations d'insertion ont été déterminante puisque le temps d'exécution de MySQL était de 10,76 sec. Dans ce cas précis, MongoDB a confirmé, une nouvelle fois, sa performance et la stabilité de ses résultats.

### III.3.2.5 Workload E (95% Scan, 5% Insert)

- Exécution du Workload E avec 100 opérations :

E 95% scan 5% Insert

| Nbre enregistrements chargés | 1 000 | | 5 000 | | 10 000 | | 600 000 | |
|---|---|---|---|---|---|---|---|---|
| SGBD | Mongo DB | My sql | Mongo DB | My sql | Mongo DB | My sql | Mongo DB | My sql |
| Temps total d'excution (sec) | 0,63 | 0,54 | 0,61 | 0,61 | 0,63 | 0,61 | 0,63 | 12,77 |




E 95% scan 5% Insert

| Nbre enregistrements chargés | 1 000 | | 5 000 | | 10 000 | | 600 000 | |
|---|---|---|---|---|---|---|---|---|
| SGBD | Mongo DB | Mysql | Mongo DB | Mysql | Mongo DB | Mysql | Mongo DB | Mysql |
| Temps total d'excution (sec) | 1,66 | 3,31 | 1,67 | 3,41 | 1,71 | 3,18 | 1,73 | 54,89 |

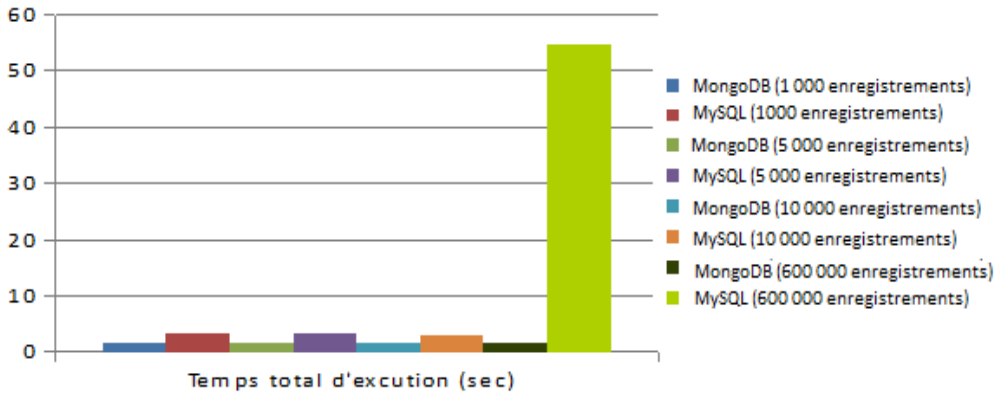



**Figure III.10: Workload E avec 1000 opérations**

Les tests effectués du Workload E prouvent la performance de MySQL, en lançant 100 opérations dans une petite base de données qui ne dépasse pas les 5000 enregistrements (0,54 ou 0,61 sec), mais en augmentant le nombre d'opérations exécutées à 1000, MongoDB

confirme sa stabilité et sa performance par rapport à MySQL. Reste à signaler, une dégradation totale de MySQL, dans un environnement de grande échelle (600 000 enreg). Cette dégradation est accentuée par l'augmentation du nombre d'opérations à effectuer.

### III.3.2.6Workload F (50 % Read, 50% R.M.W)

- Exécution du Workload F avec 100 opérations :

| 100 opérations | F 50% Read 50% RWM | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Nbre enregistrements chargés | 1 000 | | 5 000 | | 10 000 | | 600 000 | |
| SGBD | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql |
| Temps total d'excution (sec) | 0,49 | 3,11 | 0,49 | 2,59 | 0,48 | 2,87 | 0,49 | 2,98 |

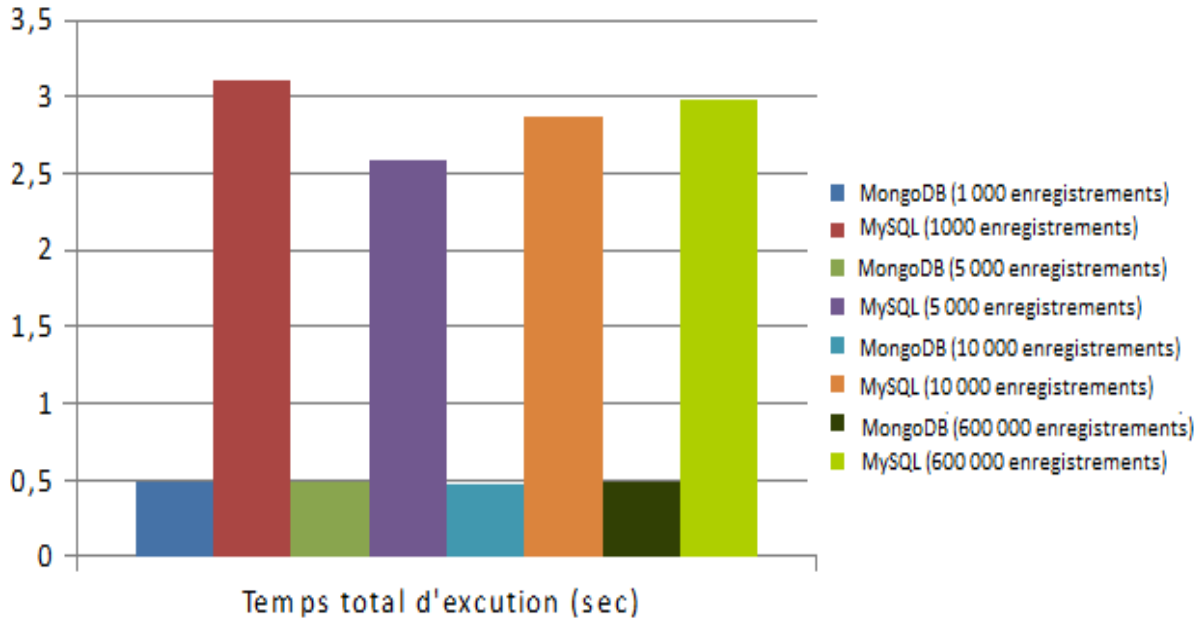



**Figure III.11:Workload F avec 100 opérations**

- Exécution du Workload F avec 1000 opérations :

| 1000 opérations | F 50% Read 50% RWM | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Nbre enregistrements chargés | 1 000 | | 5 000 | | 10 000 | | 600 000 | |
| SGBD | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql |
| Temps total d'excution (sec) | 1,17 | 25,63 | 1,17 | 25,98 | 1,15 | 25,55 | 1,14 | 39,75 |

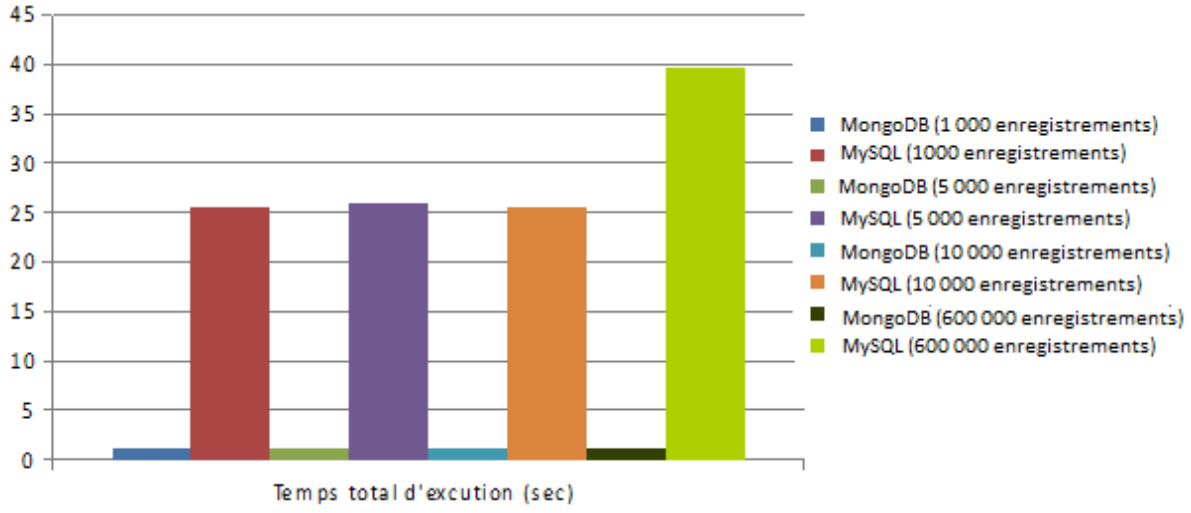



**Figure III.12: Workload F avec 1000 opérations**

Dans cette charge de travail, la moitié des opérations sont de lecture, dans l'autre moitié : les enregistrements sont lus en premier lieu, puis modifiés, ensuite une sauvegarde de données modifiées est effectuée. Les tests montrent que MongoDB est largement performante par rapport à MySQL. Notons aussi la stabilité remarquée des résultats des deux bases de données. Reste à signaler enfin, la grande lenteur approuvée par MySQL en augmentant le nombre d'opérations à 1000 avec 39,75 secondes.

### III.3.2.7Workload G (5 % Read, 95 % R.M.W)

- Exécution du Workload G avec 100 opérations :

| 100 opérations | G 5% Read 95% RWM | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Nbre enregistrements chargés | 1 000 | | 5 000 | | 10 000 | | 600 000 | |
| SGBD | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql |
| Temps total d'excution (sec) | 0,51 | 4,79 | 0,53 | 5,21 | 0,52 | 5,14 | 0,52 | 5,01 |

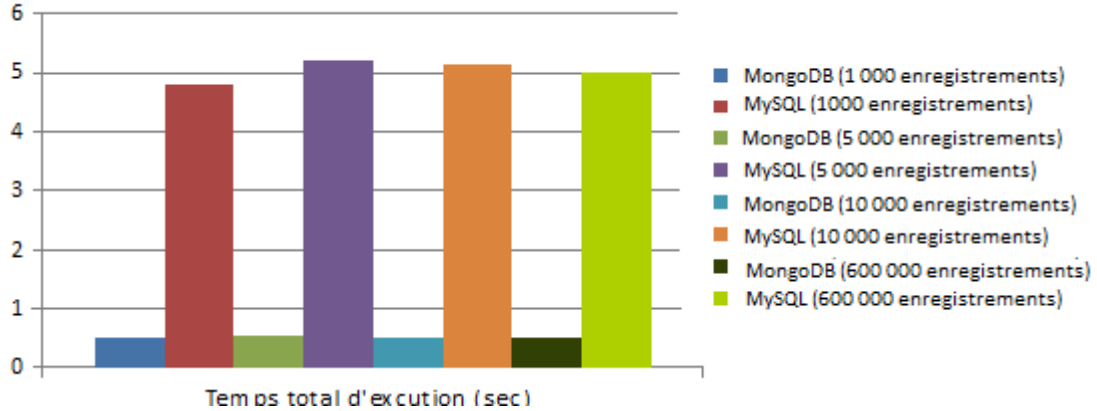



**Figure III.13: Workload G avec 100 opérations**

- Exécution du Workload G avec 1000 opérations :

| 1000 opérations | G 5% Read 95% RWM | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Nbre enregistrements chargés | 1 000 | | 5 000 | | 10 000 | | 600 000 | |
| SGBD | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql |
| Temps total d'excution (sec) | 1,32 | 50,50 | 0,95 | 51,23 | 1,34 | 52,15 | 1,49 | 65,14 |

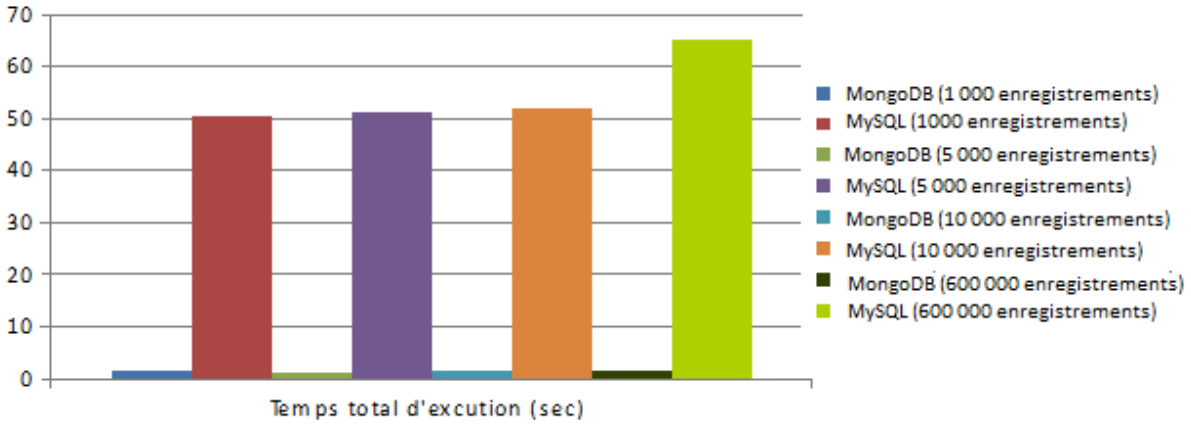



**Figure III.14: Workload G avec 1000 opérations**

Les mêmes remarques énoncées pour la charge de travail E, sont redites pour cette charge de travail G, qui consiste à augmenter le rapport des opérations de lecture-modification-écriture par rapport aux opérations de lecture (50/50).

Notons que les résultats de MySQL sont plus mauvais que ceux obtenus dans le Workload précédent en dépassant la minute pour exécuter 1000 opérations dans 600000 enregistrements.

### III.3.2.8Workload H (100 % R.M.W)

- Exécution du Workload H avec 100 opérations :

| 100 opérations | H 100% RWM | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Nbre enregistrements chargés | 1 000 | | 5 000 | | 10 000 | | 600 000 | |
| SGBD | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql |
| Temps total d'excution (sec) | 0,51 | 5,49 | 0,52 | 5,47 | 0,55 | 5,49 | 0,55 | 5,41 |

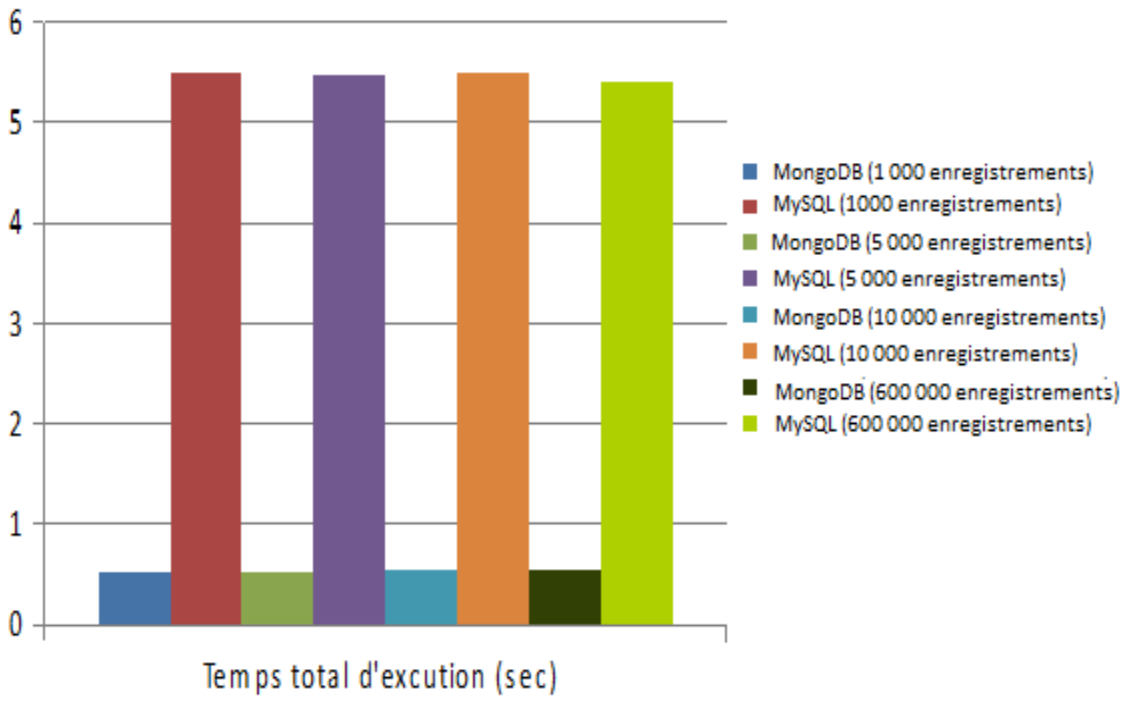



**Figure III.15: Workload H avec 100 opérations**

- Exécution du Workload H avec 1000 opérations:

| 1000 opérations | H 100% RWM | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Nbre enregistrements chargés | 1 000 | | 5 000 | | 10 000 | | 600 000 | |
| SGBD | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql | MongoDB | Mysql |
| Temps total d'excution (sec) | 1,37 | 53,41 | 1,35 | 52,81 | 1,39 | 55,63 | 1,45 | 69,47 |

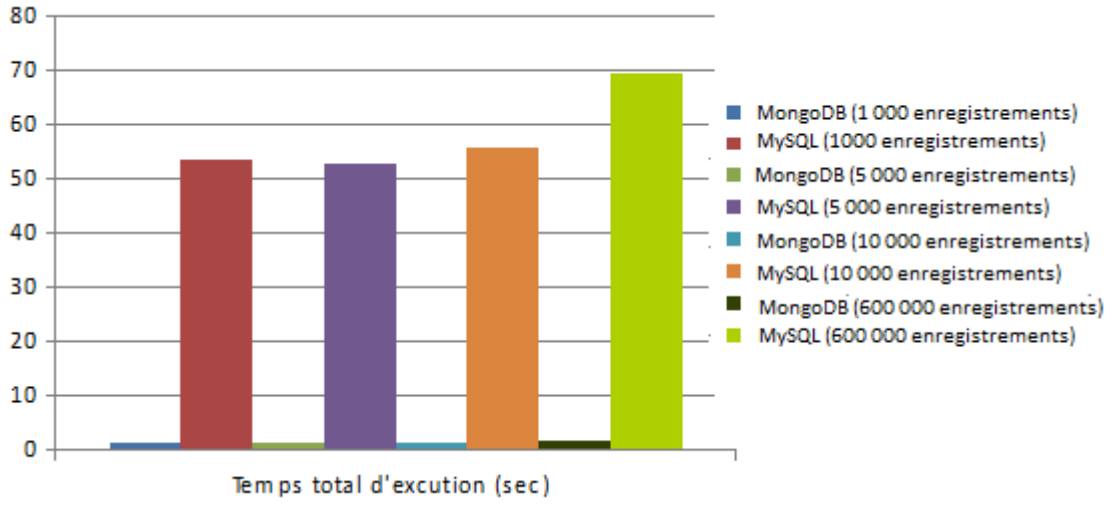



**Figure III.16: Workload H avec 1000 opérations**

Les résultats réalisés dans ce Workload H composé totalement d'opérations de lecture-modification-écriture, ont été reconduits très logiquement : Alors une autre fois, la supériorité de MongoDB sur MySQL était énorme, en termes de performance. La stabilité des résultats obtenus par les deux bases de données est confirmée une nouvelle fois.

Notons enfin, que MySQL a reproduit ses mauvaises performances réalisées dans les deux charges précédentes, en atteignant 69 secondes, pour exécuter 1000 opérations dans une base de 600000 enregistrements.

## III.3.3 Evaluation globale des tests

Les résultats expérimentaux des différents tests effectués sous forme de charges de travail, ont permis d'évaluer et comparer les deux types de SGBD NoSQL (MongoDB) et SQL (MySQL) en s'appuyant sur le temps d'exécution des différents Workloads, en variant à chaque fois le nombre d'enregistrements chargés et le nombre d'opérations à exécuter.

Sur le premier test de chargement de données, MongoDB a été très performante en inscrivant de grands écarts par rapport à MySQL. Le plus grand écart a été inscrit dans le chargement le

plus lourd à savoir 600 milles enregistrements avec 147 secondes pour MongoDB contre 11 heures pour MySQL. Cette contre-performance de MySQL est expliquée par le fait que ce système n'est pas conçu pour faire des importations ou des insertions de grands volumes de données, contrairement à MongoDB.

Les différents résultats obtenus après exécution des Workloads, ont montré que MongoDB est très performante par rapport à MySQL dans toutes les charges de travail qui contiennent des opérations d'écriture dans la base de données à savoir : A (50% mise à jour), F (50% lecture-modification-écriture), G(95% lecture-modification-écriture) et H (100% lecture-modification-écriture). De grands écarts ont été inscrits entre les deux bases de données en faveur de MongoDB, particulièrement, lorsqu'on a exécuté 1000 opérations, ou MySQL a réalisé jusqu'à 48 fois le temps réalisé par MongoDB (69,47 sec contre 1,45 sec). Nous pouvons justifier cette performance de MongoDB ainsi que la lenteur de MySQL par les exigences relatives aux propriétés ACID imposées par les SGBD relationnels (verrou de la table en cas d'écriture en empêchant les opérations de lecture), contrairement aux systèmes NoSQL qui se sont libérés de ces contraintes.

Les mêmes comportements des 2 bases de données lors des charges A, F, G et H sont réitérés pour les charges B et D, puisque les mêmes évolutions de performances sont soulignées.

Notons que les charges B et D combinent entre des opérations de lecture et de mise à jour, ces dernières sont ralenties par le processus de vérification de l'intégrité et la cohérence de la base exigés dans les systèmes transactionnels comme MySQL.

En revanche, MySQL a produit de bons résultats dans l'exécution des Workloads comportant des rapports d'opérations de lecture supérieurs à l'écriture, dans des bases de données qui ne contiennent pas plus de 10000 enregistrements, mais au-delà de ce volume, ses performances se dégradent, contrairement à MongoDB où les résultats ont préservés une certaine stabilité en gardant de bonnes performances, quel que soit le nombre d'enregistrements chargés, ceci peut être confirmé par les résultats obtenus en exécutant les charges de travail C et E.

Après la lecture des résultats obtenus en exécutant les huit charges de travail, nous pouvons conclure que la solution SQL MySQL a été adéquate pour des opérations purement lecture dans des environnements de petites échelles, mais dans des environnements à grandes échelles ou pour des opérations de lecture/écriture, il est très intéressant d'adopter la solution NoSQL MongoDB.

# Conclusion Générale

# Conclusion générale

L'explosion de la volumétrie des données, qui reflète le changement d'échelle des volumes, nombres et types, a imposé aux différents chercheurs depuis quelques années, de nouveaux défis et les a poussé à concevoir de nouvelles technologies et chercher les meilleures solutions pour contenir et traiter ces volumes énormes de données. Cette nouvelle ère informatique, avec ses nouvelles exigences, a conduit aux nouveaux concepts Big Data et Cloud Computing qui ont concouru à l'émergence du mouvement NoSQL.

Beaucoup d'utilisateurs des SGBD relationnels classiques dits « SQL » veulent basculer vers ces nouvelles solutions « NoSQL » pour anticiper l'explosion de leurs données dans le futur. Pour justifier et motiver un tel basculement du SQL vers du NoSQL, nous avons développé, dans le cadre de notre PFE, une étude comparative des performances entre deux types de bases de données d'architectures et de générations différentes : MySQL en tant que modèle relationnel connu par SQL et MongoDB en tant que modèle non relationnel, représentant la nouvelle ère NoSQL.

Les différents tests effectués sous forme de charges de travail étaient basées sur le temps d'exécution pour évaluer les performances. Ces tests ont été soumis à plusieurs configurations et paramétrages :

1. Le premier critère concernait la nature des opérations à exécuter (lecture, mise à jour,...)
2. Le deuxième concernait le nombre de lignes insérés (1000, 5000, 10000 ou 600000)
3. Le troisième concernait le nombre d'opérations effectuées (100 ou 1000).

Bien que les résultats expérimentaux obtenus dans notre étude, soient en faveur de la solution NoSQL MongoDB par rapport à la base de données relationnelle MySQL en termes de performance, la tendance vers une favorisation d'une solution NoSQL et l'abandon du relationnel est loin d'être indiscutable.

On peut retenir aussi que le choix d'utilisation d'un SGBD dépend d'un ensemble de paramètres en relation avec l'environnement dans lequel les données sont exploitées. En effet le type de données et le type des traitements effectués sur ces données sont des indices importants pour définir la solution à adopter, ainsi que les besoins spécifiques qui peuvent varier d'une entreprise à une autre. Citons ces principaux indices :

- L'extensibilité : Dans une organisation qui n'éprouve pas une énorme augmentation des données, on peut opter pour un système SQL au lieu d'un système NoSQL. En revanche, dans un contexte BigData et Cloud Computing avec des applications en temps réel

produisant et échangeant d'énormes quantités de données, le NoSQL sera un choix plus judicieux.

- La redondance : En utilisant une forme structurée des données et en focalisant sur la cohérence des données, la redondance est très limitée dans les bases de données SQL car la nature relationnelle de la base de données permet d'éviter la redondance, comme c'est le cas de MySQL. En revanche, la redondance des données dans les différents serveurs dans les environnements distribués, pour assurer la disponibilité de l'information, est un aspect déterminant dans les bases de données NoSQL.
- La variété de données : Dans un domaine applicatif ou un schéma plus structuré de la base de données est requis, on préfère un système SQL pour sa capacité de prise en charge des données structurées. Par contre, si les données manipulées sont semi-structurées ou non structurées, le choix d'un modèle NoSQL est plus convenable.
- La flexibilité **:** La possibilité de stocker des types hétérogènes provenant de diverses sources de données comme les smartphones, les capteurs ou les satellites, est un avantage essentiel offert par les bases de données NoSQL, comme c'est le cas de MongoDB, qui peut combiner plusieurs types de données dans un seul document.
- La vitesse : Les solutions NoSQL et en particulier MongoDB, offrent un accès efficace et rapide aux données, mieux que les bases de données SQL et en particulier MySQL, en raison de l'application des jointures pour lier les différentes tables relationnelles et le respect imposé des contraintes d'intégrité, avant d'y répondre à une requête.

En effet, Il n'y a pas de solution parfaite et il n'y a pas de mauvaises solutions de gestion de données, NoSQL ou SQL. Le contexte dans lequel les données sont utilisées, les exigences du système, la dimension de l'environnement et la nature des opérations effectuées, fournissent un certain nombre de critères, qui peuvent déterminer ou orienter vers le modèle de base de données à adopter. Si l'application exige une par exemple, des propriétés ACID, on est amené vers les SGBDR, si on ne s'intéresse pas beaucoup à ces contraintes et particulièrement à la cohérence, une solution NoSQL est recommandée. Une autre alternative peut être envisagée, c'est combiner entre les deux solutions, y compris les bases de données relationnelles SQL et NoSQL à la fois, en utilisant un système hybride.

Enfin, on peut estimer que l'objectif ciblé a été largement atteint, néanmoins ce travail pourrait être prolongé sur plusieurs aspects, citons :

- Etaler notre étude à d'autres solutions NoSQL comme : CouchBase, Cassandra, Hbase, Redis ou OrientDB et d'autres solutions SQL comme Oracle et SQL Server.

- Élever le nombre d'enregistrements pour atteindre ou dépasser un million.
- Créer d'autres charges de travail en variant les types d'opérations.
- Etendre l'étude comparative à la nouvelle génération NewSQL.
- Etendre notre travail d'un environnement monoposte à un environnement distribué.

## Références bibliographiques

**[1]** V. Raja Raman, Introduction to Information Technology, PHI Learning Pvt. Ltd., 2004 (ISBN 9788120324022)

**[2]** Michel Grech, Bases de données relationnelles & SQL, Infotique, PARIS. Support de cours : http://wiki.humanum.ephe.fr/lib/exe/fetch.php/tutoriaux/supportbdrsql.pdf

**[3]** Olivier Losson, Introduction aux Systèmes de Gestion de Bases de Données Relationnelles, cours Master Sciences et Technologies, Université Lille1 :http://lagis-vi.univ-lille1.fr/~lo/ens/commun/bd/bd_cours.pdf

**[4]** Georges Gardarin, Bases de données, Editions Eyrolles, 5e tirage 2003.

**[5]** Christian Soutou, apprendre SQL avec MySQL, Editions Eyrolles, 2006.

**[6]** R Ferrere, initiation aux bases de données modélisation et langage SQL Master2 :http://docplayer.fr/2958727-Initiation-aux-bases-de-donnees-modelisation-et-langage-sql.html

**[7]** Xavier Maletras, Le NoSQL- Cassandra, thèse professionnelle, université Paris, 2012.

**[8]** Kouedi Emmanuel, Approche de migration d'une base de données relationnelle vers une base de données NoSQL orientée colonne, Mémoire master informatique, Université de Yaoundé I, mai 2012.

**[9]** http://www.supinfo.com/articles/single/1483-base-donnees-NoSQL par Omar SENHAJI Publié le 31/10/2015.

**[10]** Matteo DI MAGLIE, Adoption d'une solution NoSQL dans l'entreprise, réalisée en vue de l'obtention du Bachelor HES, Carouge, 12 septembre 2012

**[11]** Meyer Léonard, L'AVENIR DU NoSQL, 2014.

**[12]**http://sql-vs-NoSQL.blogspot.com/2013/10/the-base-difference-between-sql-and.html publier le 25-10-2013.

**[13]** https://blog.developpez.com/sqlpro/p10383/langage-sql-norme/sgbdr_et_repartion_de_charge_scalabilite publié le 10 octobre 2011par SQL pro

**[14]** https://www.couchebase,com/NoSQL-resources/whay-NoSQL consulté le 03Mars2017

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

**[16]**Lionel HEINRICH, Not Only SQL, en vue de l'obtention du Bachelor HES en Informatique de Gestion, Genève, le 26 octobre 2012

# Liste de figures

**Résumé**

Les bases de données NoSQL ou «Not Only SQL» sont des types de bases de données qui ont commencé à émerger depuis 2009, pour répondre aux nouveaux besoins du Cloud et du Big Data ainsi que proposer des alternatives aux bases de données relationnelles. Beaucoup d'utilisateurs des SGBD relationnels classiques dits « SQL » veulent basculer vers ces nouvelles solutions «NoSQL» pour anticiper l'explosion de la volumétrie et coller aux nouvelles tendances. Dans le cadre de notre PFE, nous développons une étude comparative sur les performances de deux solutions très répandues des deux générations MySQL (SQL) et MongoDB (NoSQL) 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. La finalité est d'apporter l'assistance et l'aide nécessaire aux acteurs intéressés pour des éventuelles prises de décision sur le choix des solutions à adopter.

**Mots clés**: SQL, NoSQL, MySQL, MongoDB, YCSB, Workload.

**Abstract**:

Not Only SQL databases are database types that have begun to emerge since 2009, to meet the new needs of Cloud and Big Data as well as to offer alternatives to relational databases. Many users of traditional SQL relational databases want to switch to these new "NoSQL" solutions to anticipate the explosion of volumetry and stick to new trends. As part of our PFE, we develop a comparative study on the performance of two widely used solutions of the two generations MySQL (SQL) and MongoDB (NoSQL) using the YCSB tool. The latter is a leader tool known for its test power and used in several evaluations of NoSQL databases. The finality is to provide the necessary help and assistance to the actors concerned for possible decision-making on the choice of solutions to be adopted

**Keywords** :SQL, NoSQL, MySQL, MongoDB, YCSB, Workload.

ملخص

قواعد البيانات NoSQL التي بدأت في الظهور منذ عام 2009 هي نوع من الحلول التي حلت لتلبية الاحتياجات الجديدة من الحوسبة السحابية والبيانات الضخمة واقتراح بدائل لقواعد البيانات العلائقية. العديد من المستخدمين لأنظمة إدارة قواعد البيانات العلائقية التقليدية "SQL" يريدون التحول إلى هذه الحلول الجديدة "NoSQL" لاستباق ارتفاع حجم البيانات والتطلع نحو هذا التوجه الجديد. كجزءمن مشروع نهاية الدراسة، نقومبدراسة مقارنة على أداء نوعين من الحلول المستخدمة على نطاق واسع و هما MySQL(SQL) و (NoSQL) MongoDB باستخدام أداة YCSB. هذا الأخير هو أداة اختبارمعروفة و جد فعالة لتقييم أداءالعديد من قواعد البيانات NoSQL. والغرض من هذا العمل هو تقديم التوجيه والمساعدة اللازمة للمستخدمين في اتخاذ قراراختيار الحلول المناسبة.

**الكلمات المفتاحية** : SQL, NoSQL, MySQL,MongoDB,YCSB, عبء العمل.