Table des matières

Introduction Générale

Contexte

Depuis plusieurs années, les systèmes de gestion de bases de données relationnelles ont pris l'ascendant sur les autres modèles de données (modèle hiérarchique et réseau). Ils sont devenus un élément essentiel des organisations et un modèle de référence pour la gestion des données des systèmes d'informations. Ces architectures ont été développées comme une technologie pour stocker les données structurées et organisées sous forme de tableaux. Le modèle relationnel, apprécié par les entreprises pour sa structuration de données fortement cohérentes et le support de transaction ACID, est aujourd'hui remis en question par l'apparition de nouveaux besoins, liés à l'augmentation continue des données hétérogènes stockées et analysées, qui ont émergé de nouveaux concepts à savoir Big Data et Cloud Computing. Ce phénomène implique que certaines entreprises doivent maintenant gérer des volumes gigantesques de données en croissance exponentielle. Répondant à cette problématique, le NoSQL apparaît comme une solution viable mais dénuée de quelques défauts, citons principalement : l'absence de normalisation des solutions proposées, vu qu'elles étaient développées pour répondre à des besoins spécifiques des entreprises. Une autre contrainte relative à l'abandon des contraintes ACID pour favoriser la latence et la disponibilité. Cependant, le tout nouveau venu « NewSQL » semble promettre une architecture combinant les avantages du modèle relationnel et le NoSQL.

Problématique

Actuellement, plusieurs utilisateurs des SGBD classiques dits « SQL » veulent basculer vers les nouvelles technologies NewSQL. En revanche, l'apparition de différents modèles NewSQL qui fournissent de nouvelles fonctionnalités, d'une part, et l'absence de preuves tangibles qui justifient ce basculement, d'autre part, impose une question très pertinente : Quelle solution NewSQL à adopter ?

Contribution

Pour apporter les réponses nécessaires, nous allons développer, dans le cadre de ce projet de fin d'études, une analyse critique et comparative entre l'ancienne génération

relationnelle SQL et la nouvelle génération NewSQL, pour orienter les utilisateurs vers les solutions les plus appropriées selon leurs besoins.

Autrement dit, il s'agit de présenter une évaluation expérimentale de deux types de bases de données SQL (MySQL) et NewSQL (VoltDB), afin de fournir un ensemble de critères et d'indicateurs, aux acteurs intéressés pour des prises de décisions éventuelles sur les solutions adoptées pour leurs entreprises.

Organisation du mémoire

Notre mémoire est organisé comme suit :

- Dans le premier chapitre, on va présenter l'évolution des différentes générations des systèmes de gestion de données.
- Le deuxième chapitre va se focaliser sur la nouvelle technologie NewSQL, en recensant les solutions disponibles dans le marché, pour s'accentuer sur VoltDB.
- Le troisième chapitre fera l'objet de notre étude comparative MySQL vs VoltDB.

-

Chapitre I

Systèmes de gestion de données

I - Introduction

Un système de gestion de données est un outil permettant de stocker et de retrouver l'intégralité de données brutes ou d'informations en rapport avec un thème ou une activité, celles-ci peuvent être de natures différentes et plus ou moins reliées entre elles. Dans la très grande majorité des cas, ces informations sont structurées, et la base est localisée dans un même lieu et sur un même support généralement informatisé. Ce chapitre sera consacré à l'évolution des différentes générations des systèmes de gestion de données, en commençant en premier lieu par les fichiers plats, pour rappeler ensuite les principes des bases de données relationnelles, qui ont pris le pas sur les autres modèles de données pendant plusieurs années. Par la suite, on va expliquer les concepts de base de la mouvance NoSQL, pour en finir, par la dernière génération des systèmes de gestion de données connue par « NewSQL » qui semble promettre une architecture combinant les avantages des deux modelés cités précédemment.

II – Fichiers plats

Les fichiers plats sont l'une des premières formes de fichiers électroniques stockés et sont encore en usage aujourd'hui. Les systèmes d'exploitation DOS, Macintosh et les premières versions de logiciels tels que FileMaker utilisaient une des premières formes de fichiers plats. Ce sont des fichiers de données contenant un enregistrement par ligne, et dont les champs peuvent être délimités les uns des autres par un caractère spécial.

La conception d'un fichier plat consiste à identifier certaines caractéristiques des champs dans chaque enregistrement : nombre de domaine, nom de domaine et une description de type de données stockées et sa taille maximale et minimale.

Les documents conservés dans un fichier plat sont singuliers et n'ont pas de relation avec d'autres documents dans le dossier, contrairement aux bases de données ou aux fichiers relationnelles.

Les fichiers plats prennent beaucoup moins d'espace lorsqu'ils sont stockés. Pour les entreprises qui détiennent de grandes quantités de données, il peut donner un sens plus économique d'utiliser des fichiers plats.

La récupération d'un enregistrement à partir d'un fichier plat se fait dans le code de programmation qui est lié à une interface utilisateur graphique, comme une forme de

saisie de données à l'écran. Les enregistrements peuvent également être récupérés via les écrans de commande ou à l'aide d'une requête. Les requêtes sont écrites en connaissant le format de fichier et en utilisant un langage de requête spécifique. Par exemple, dans un système UNIX un langage de requête appelé " cql " est utilisé [1].

III - Bases de données relationnelles

III.1 – Introduction

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 [2].

Dans ce modèle, les données sont représentées par des tables, sans préjuger de la façon dont les informations sont stockées dans la machine. Les tables constituent donc la structure logique du modèle relationnel [3], d'autre part ces données sont gérées par un système de gestion de bases de données relationnelle (Relational DataBase Management System, 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 [4].

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

IiI.2 – Principe de fonctionnement de SGBD relationnel

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 :

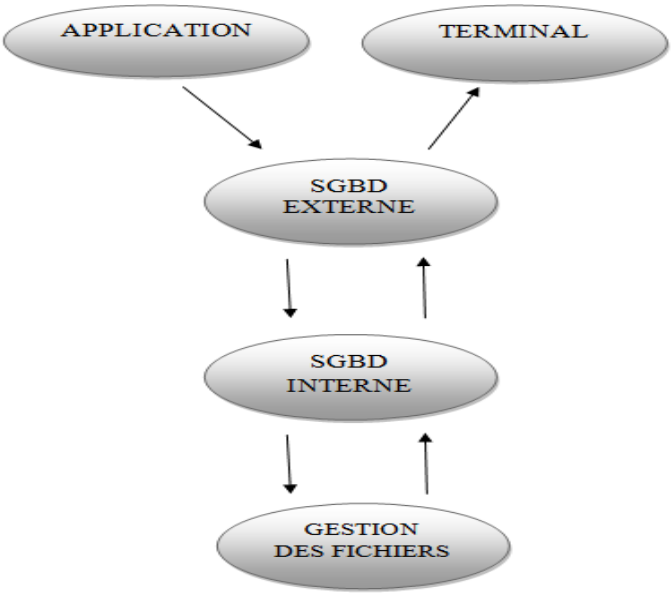


Figure 1-1 Sous-systèmes d'un SGBD

1. Le système de gestion de fichiers : permet de stocker les informations sur un support physique.
2. Le SGBD interne : gère l'ordonnancement des informations.
3. Le SGBD externe : présente l'interface de l'utilisateur.

IiI.3 – Propriétés ACID

Les propriétés ACID sont les quatre principaux attributs d'une transaction de données sur lesquelles s'appuie ce type de base de données. Il s'agit là d'un des concepts les plus anciens et les plus importants du fonctionnement des bases de données relationnelles, il spécifie quatre buts à atteindre pour toute transaction :

- **Atomicity (Atomicité)**

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

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

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

La notion de cohérence est importante, et c'est l'un des éléments les plus sensibles entre le monde du relationnel et le monde NoSQL (qu'on va voir par la suite). Les SGBDR imposent une règle stricte : d'un point de vue transactionnel, les lectures de données se feront toujours sur des données cohérentes. Les modifications en cours sont donc cachées, un peu comme sur la scène d'un théâtre : chaque acte d'une pièce de théâtre se déroule intégralement sous l'oeil des spectateurs. Si le décor va être changé, le rideau se baisse, les spectateurs doivent attendre, les changements s'effectuent derrière le rideau et lorsque le rideau se lève, la scène est de nouveau dans un état totalement cohérent. Les spectateurs n'ont jamais eu accès à l'état intermédiaire. Néanmoins, cet état intermédiaire peut durer longtemps, pour deux raisons : plusieurs instructions de modifications de données peuvent être regroupées dans une unité transactionnelle, qui peut être complexe. Ensuite, un SGBDR fonctionnant de façon ensembliste, une seule instruction de mise à jour, qui est naturellement et automatiquement transactionnelle, peut très bien déclencher la mise à jour de milliers de ligne de table. Cette modification en masse conservera des verrous d'écriture sur les ressources et écrira dans le journal de transaction ligne par ligne. Tout ceci prend, bien évidemment, du temps.

Ce respect strict de la cohérence est concevable si nous restons sur le même serveur, mais dès que nous adoptons une distribution de données, il commence à poser de sérieux problèmes. La modification des données qui résident sur plusieurs serveurs n'est aujourd'hui possible qu'à partir d'un mécanisme nommée transaction distribuée.

- **Isolation (Isolation)**

Chaque transaction voit l'état du système comme si elle était la seule à manipuler la base de données, c'est-à-dire si les transactions sont lancées en même temps, on n'aura jamais des interférences entre elles. Par exemple si pendant la transaction T1, une

transaction T2 est exécutée en même temps, T1 ne doit pas la voir, tout comme T2 ne doit pas voir T1.

Une transaction isole les données accédées pour éviter une corruption de donnée qui peut être causée par une modification simultanée de la même donnée par plusieurs utilisateurs concurrents.

Ce verrouillage provoque des attentes dans certains moteurs relationnels, comme Microsoft SQL Server. D'autres moteurs comme Oracle utilisent des techniques qui diminuent le verrouillage en maintenant la dernière version transactionnellement cohérente antérieure à la modification afin de permettre des lectures même en cas de modifications.

En revanche, deux tentatives d'écriture simultanée sont interdites. Cette isolation est maintenue par la pose de verrous sur les ressources accédées. La gestion des verrous est la responsabilité de moteurs de stockage. Les problématiques de verrouillage sont relativement complexes.

- **Durability (Durabilité)**

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

III.4 - Limites des bases de données relationnelles

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

distribuées sur plusieurs machines, de plus pour des raisons de fiabilité ces bases de données sont dupliquées pour que le service ne soit pas interrompu en cas de panne.
Malheureusement le modèle relationnel présente quelques problèmes liés à ce passage à l'échelle tel que [7] :

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

Une base de données relationnelle est construite en respectant les propriétés ACID (Atomicité, Cohérence, Isolation, Durabilité). Ces propriétés bien que nécessaires à la logique du relationnel nuisent fortement aux performances de cohérence.
En effet, la cohérence est très difficile à mettre en place dans le cadre de plusieurs serveurs (environnement distribué), car pour que ceux doivent être des miroirs les uns des autres. Deux problèmes qui émergent :
• Le coût en stockage est énorme car chaque donnée est présente sur chaque serveur.
• Le coût d'insertion/modification/suppression est très grand, car on ne peut valider une transaction que si on est certain qu'elle a été effectuée sur tous les serveurs, du coup le système fait patienter l'utilisateur durant ce temps [7].

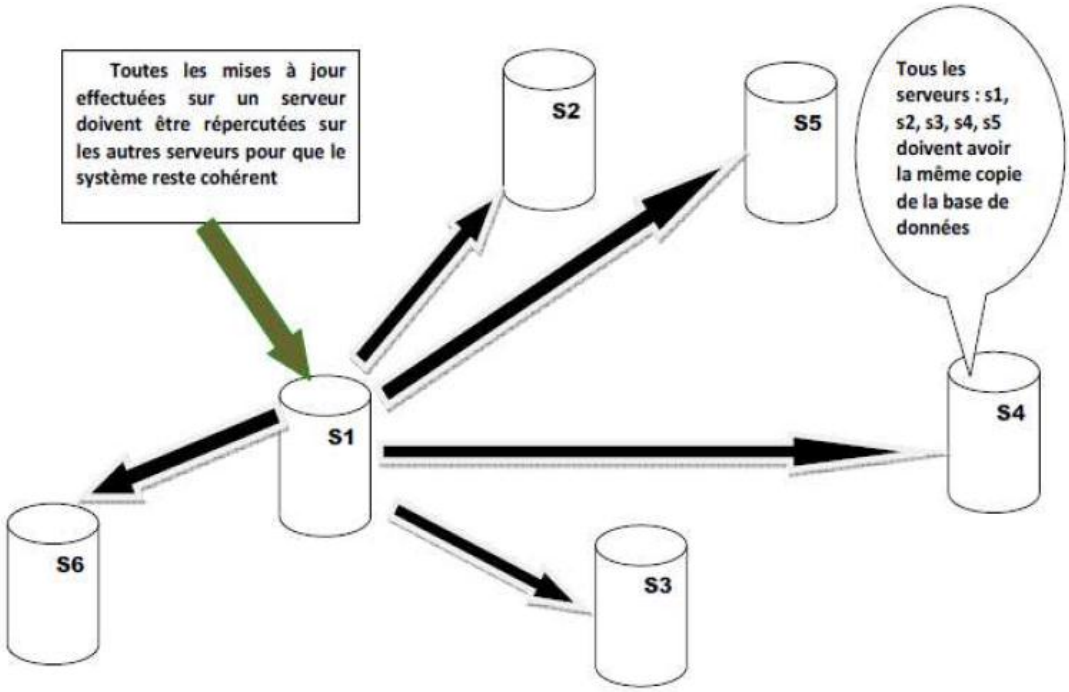


Figure I.2 : Problème lié aux propriétés ACID en milieu distribué [9]

- **Problème de requête non optimale suite à l'utilisation des jointures**

Imaginons une table contenant toutes les personnes ayant un compte sur Facebook, soit plus de 1 milliard, les données dans une base de données relationnelle classique sont

stockées par lignes, ainsi si on effectue une requête pour extraire tous les amis d'un utilisateur donné, il faudra effectuer une jointure entre la table des usagers (1 milliard) et celle des amitiés (chaque usager ayant au moins un ami donc au minimum on a 1 milliard), puis on va parcourir le produit cartésien de ces deux tables. De ce fait, on perd énormément en performances car il faut beaucoup de temps pour stocker et parcourir une telle quantité de données.

- **Type de données**

Les systèmes relationnels sont limités à des types simples (entiers, réels, chaines de caractères) ou des données composés comme dans le modèle Objet-Relationnel, les seules types étendus se limitant à l'expression de dates ou de données financières, ainsi que des conteneurs binaires de grande dimension (BLOB , Pour Binary Large OBjects) qui permettent de stocker des images ainsi que des fichiers audio ou vidéos .Ces BLOBs ne sont toutefois pas suffisants pour représenter des données complexes non structurés, les mécanismes de contrôles BD sont inexistants et le langages de requêtes SQL ne possède pas les opérateurs correspondant aux objets stockés dans ces BLOBs.

- **Langage de manipulation**

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

- **Requêtes récursives**

Il est possible, avec des requêtes SQL imbriquées, de travailler sur des relations de type parent-enfant, mais il est possible de travailler sur des relations de type ancêtre. Pour traiter ce type de problème, les requêtes doivent être insérées dans un langage de programmation.

- **Scalabilité limitée**

Atteinte par les poids lourds du Net comme Yahoo, Google, Facebook, Twitter ou LinkedIn, cette limite est la plus gênante pour une entreprise. Dans le cas de traitement 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 aussi gigantesques générés par ces opérateurs.

IV - Le NoSQL

IV.1 – Introduction

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

- Les gros volumes de données,
- Les montées en charge.

N'ayant pas trouvé de solution sur le marché répondant à ces problèmes, ils décidèrent de développer chacun à l'interne leurs propres SGBD. Ces produits développés de manière distincte sont connus sous le nom de base de données NoSQL ou de base de données non relationnelle. Les besoins en performance, lors de traitement de gros volumes de données ainsi que d'augmentation de trafic, ne touchent pas seulement les fameux sites mentionnés ci-dessus, mais aussi de nombreuses entreprises de tous types d'industries confondues, c'est de ce constat qu'est né le mouvement NoSQL [6].

NoSQL est un mouvement très récent, ce n'est qu'en 2009, lors d'un rassemblement de la communauté des développeurs des SGBD non-relationnels, que le terme NoSQL a été mis au goût du jour pour englober tous les SGBD de type non-relationnel. L'idée du mouvement est simple : proposer des alternatives aux bases de données relationnelles pour coller aux nouvelles tendances et architectures du moment, notamment le Cloud Computing et le Big Data. Les axes principaux du NoSQL sont une haute disponibilité et un partitionnement horizontal des données, au détriment de la cohérence. Alors que les bases de données relationnelles actuelles sont basées sur les propriétés ACID (Atomicité, Consistance ou Cohérence, Isolation et Durabilité).
En effet, NoSQL ne vient pas remplacer les BD relationnelles mais proposer une alternative ou compléter les fonctionnalités des SGBDR pour donner des solutions plus intéressantes dans certains contextes. Leurs principaux avantages sont leurs performances et leur capacité à traiter de très grands volumes de données [9].

IV.2 – Théorème de CAP

Avant d'aborder les différents types de base de données NoSQL, il est important d'expliquer le théorème CAP. Alors que le modèle relationnel obéit aux formes normales et aux règles ACID, les bases de données NoSQL suivent le théorème CAP. Le théorème de CAP est l'acronyme de « Coherence », « Availability » et « Partition tolerance », aussi connu sous le nom de théorème de Brewer. Ce théorème, formulé par Eric Brewer en 2000 et démontré par Seth Gilbert et Nancy Lych en 2002, énonce une conjecture qui définit qu'il est impossible, sur un système informatique de calcul distribué, de garantir en même temps les trois contraintes suivantes :

- **Cohérence**

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

- **Disponibilité**

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

- **Résistance au partitionnement**

  Le système doit être en mesure de répondre de manière correcte à toutes requêtes dans toutes les 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.

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

Les bases de données relationnelles implémentent les propriétés de Cohérence et de Disponibilité (système AC). Les bases de données NoSQL sont généralement des systèmes CP (Cohérent et Résistant au partitionnement) ou AP (Disponible et Résistant au Partitionnement). De ce fait, le NoSQL concentrent plus sur la résistance au morcellement, afin de garantir une disponibilité en tout temps et par conséquent abandonnent la cohérence.

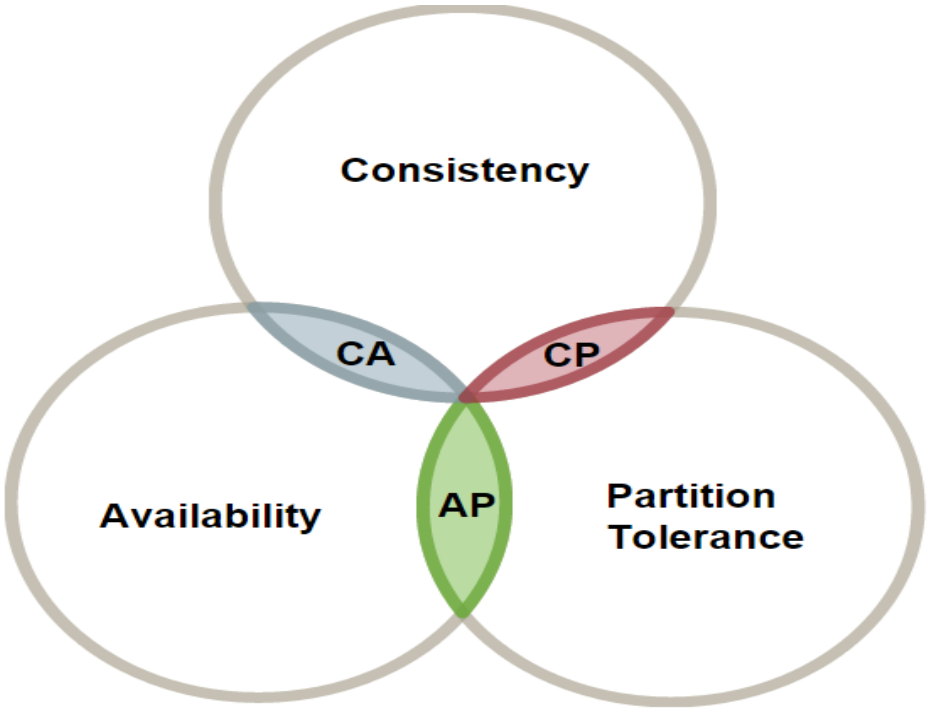


Figure I.3 : Théorème CAP [10]

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 [11].

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 [11].

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) [11].

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

IV.3 - Types de bases de données NoSQL

Il existe plusieurs types et architectures de bases de données NoSQL, cependant, elles ont toutes un point commun, c'est l'abandon des contraintes ACID. Ces bases stockent soit des paires clé-valeur soit des documents JSON. Elles ne possèdent pas de schémas et de contraintes sur les objets. Le développeur est libre de stocker et organiser les données dans la base comme il veut. C'est lui qui définit au travers de son application la structure et les règles des éléments qui y seront manipulés et sauvés [10].

Les quatre types de bases de données NoSQL sont présentés dans ce qui suit :

- **Les entrepôts clé-valeur**

Les bases de données NoSQL de type clé / valeur s'articulent sur une architecture très basique. On peut les apparenter à une sorte de HashMap, c'est-à-dire qu'une valeur, un nombre ou du texte est stocké grâce à une clé, qui sera le seul moyen d'y accéder. Leurs fonctionnalités sont tout autant basiques, car elles ne contiennent que les commandes élémentaires du CRUD (Create, Read, Update, Delete), cela permet aux données totalement hétérogènes entre elles d'être stockées. Ces modèles peuvent assurer une performance élevée en lecture/écriture.

Redis et Riak sont les principales bases de données de type entrepôt clé-valeur [17].

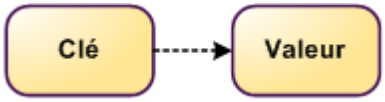


Figure I.4 : Une valeur identifiée par une clé

- **Bases orientées documents**

Les bases de données orientées documents se rapprochent du modèle clé-valeur. La valeur étant cette fois caractérisée par un document dans un format hiérarchique semi-structuré dont la structure est libre. Il est identifié par une clé unique. Ce type de modèle permet de stocker des données non-planes. Ce qui signifie des relations avec des jointures de type One to Many. C'est particulièrement approprié pour les applications web. On retrouve principalement MongoDB et CoucheBase comme solutions basées sur le concept de base documentaire [10].

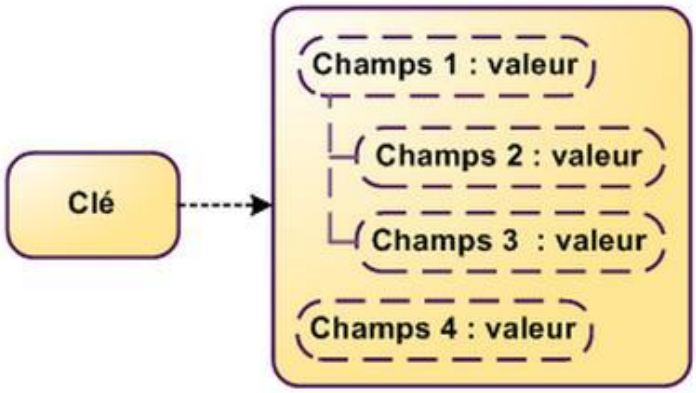


Figure I.5 Modèle orienté document

- **Bases orientées colonnes**

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

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

Le principe d'une base de données colonne consiste dans leur stockage par colonne et non par ligne, les données sont organisées 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. Autrement dit, et contrairement à une base de données relationnelle où les colonnes sont statiques et présentes pour chaque ligne, celles des bases de données orientées colonne sont dite dynamiques et présentes donc uniquement en cas de nécessité. De plus il n'y a pas de stockage de valeur « null » [16].

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

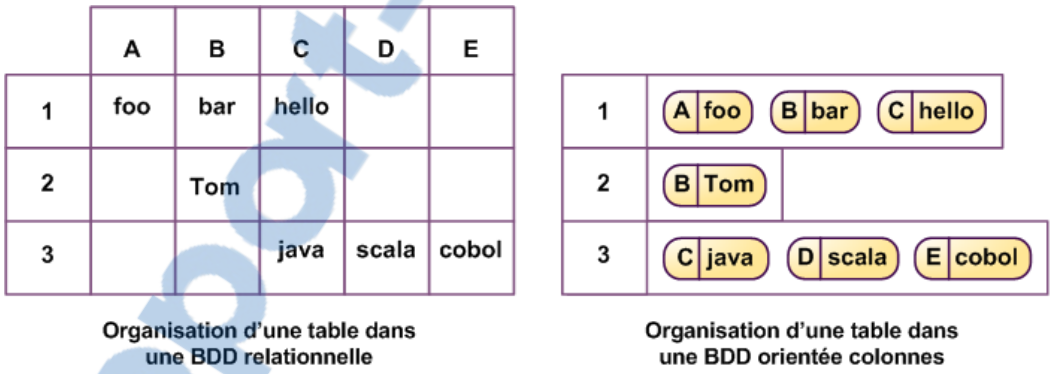


Figure I.6 Bases orientées colonnes

- **Bases orientées graphes**

Les bases de données NoSQL orientées graphes cherchent avant tout à répondre à des problèmes complexes voire impossibles pour des SGBDR. Elles offrent une réponse adéquate à un modèle de données très volumineux et fortement connecté [10].

Les réseaux sociaux (Facebook, Twitter,etc), où des millions d'utilisateurs sont reliés de différentes manières, constituent un bon exemple : amis, fans, famille etc. Le défi ici n'est pas le nombre d'éléments à gérer, mais le nombre de relations qu'il peut y avoir

entre tous ces éléments. Evidement et comme son nom l'indique, ces bases de données reposent sur la théorie des graphes.

Ce type de base de données NoSQL est le moins utilisé. Il répond à une problématique très spécifique qui intéresse encore peut les entreprises (hormis quelques grands acteurs des réseaux sociaux).

IV.4 - Requêtage NoSQL

Dans le monde du NoSQL il n'y a pas de langage standard comme SQL l'est dans le monde des bases de données relationnelles. L'interrogation des bases de données NoSQL se fait au niveau applicatif à travers, principalement, la technique dite de «MapReduce». MapReduce est une technique de programmation distribuée très utilisée dans le milieu NoSQL, qui vise à produire des requêtes distribuées. Cette technique se décompose en deux grandes étapes qui sont le Mapping et le Reduce.

IV.5 - Avantages du NoSQL

- **Plus évolutif**

NoSQL est plus évolutif. C'est en effet l'élasticité de ses bases de données NoSQL qui le rend si bien adapté au traitement de gros volumes de données.

- **Plus flexible**

N'étant pas enfermée dans un seul et unique modèle de données, les applications NoSQL peuvent donc stocker des données sous n'importe quel format ou structure, et changer de format en production. En fin de compte, cela équivaut à un gain de temps considérable et à une meilleure fiabilité.

- **Plus économique**

Les serveurs destinés aux bases de données NoSQL sont généralement bon marché, de plus, la très grande majorité des solutions NoSQL sont Open Source, ce qui reflète une économie importante sur le prix des licences.

- **Plus simple**

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

IV.6 – Inconvénients du NoSQL

La technologie NoSQL présente quelques désavantages. Contrairement aux bases de données relationnelles, la plupart des bases de données NoSQL ne se basent pas sur les principes ACID. En effet, ils offrent la disponibilité de grands volumes de données au détriment de la fiabilité et la cohérence des transactions. De plus, Ils sont souvent longs pour exécuter de grandes requêtes ou à supporter la fusion de plusieurs requêtes [12].

IV - Le NewSQL

Afin de remédier aux points faibles du NoSQL, le NewSQL a été développé pour mieux supporter les applications Big Data en jumelant les avantages des SGBDR et le NoSQL. Basé sur une architecture distribuée, NewSQL assure les principes ACID et supporte les requêtes SQL. Les bases de données NewSQL sont donc recommandées pour les applications des Big Data nécessitant un bon niveau de disponibilité de données et une performance d'exécution et fiabilité de transactions [12].

V - Conclusion

Dans ce premier chapitre, nous avons présenté les différents modèles de données qui ont été conçus et employés pour répondre aux besoins de stockage et gestion de données depuis les premières ères de l'informatique, en commençant par les fichiers plats jusqu'au NewSQL.

Dans la section suivante, nous allons accentuer sur cette nouvelle technologie NewSQL et analyser les solutions disponibles, en particulier VoltDB, qui va faire l'objet de notre étude comparative.

Chapitre II

Le NewSQL

I – Introduction

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. Cependant, il apparait que cette technologie, dans la majorité des cas, sacrifie la forte cohérence des données si chère au modèle relationnel, au bénéfice de la haute disponibilité et de la scalabilité horizontale. En revanche, cela ne plait pas à tout le monde, certaines entreprises ne peuvent pas abandonner les transactions ACID (ex : banques, bourses, etc…). D'autre part, l'instabilité et la pauvreté en fonctionnalités sont des effets marquants de cette mouvance NoSQL. Notons aussi que chaque système de gestion de base de données NoSQL parvient avec son propre langage de requête, contrairement aux systèmes relationnels qui adoptent le langage normalisé SQL. On déplore aussi un manque de standardisation des interfaces entre les applications.

Dans l'optique de trouver de meilleures solutions, en 2011, un groupe de recherche nommé "451 Research" a analysé avec brio le problème évoqué ci-dessus et propose le concept "NewSQL" pour définir une base de données regroupant les avantages du NoSQL et du SQL (SGBDR).

Dès lors, plusieurs implémentations de ce nouveau type d'architecture sont apparues très récemment paraissant dans la figure suivante :

Figure II.1 : le NewSQL et ses implémentations

II – La technologie NewSQL

NewSQL est un modèle de stockage distribué potentiellement en mémoire qui peut être requêté classiquement par une interface SQL. Le NewSQL, désigne une catégorie de bases de données modernes, qui a émergé du monde NoSQL mais reste différent sur plusieurs aspects. Comme NoSQL, il s'agit d'une nouvelle architecture logicielle qui propose de repenser le stockage des données pour prendre en charge les masses d'informations. Elle profite des architectures distribuées et des évolutions sur le plan du matériel pour coller aux nouvelles tendances. Contrairement à NoSQL, le New SQL permet de conserver le modèle relationnel au cœur du système.

En fait, le NewSQL est né du croisement de 3 types d'architectures, relationnelle, non-relationnelle et grille de données appelée également cache distribué, comme indiqué dans la Figure II.2. Le NewSQL s'appuie sur un stockage distribué issu des architectures NoSQL, pour supporter des accès transactionnels à fort débit, au moyen d'une interface SQL.

D'un point de vue évolutivité, il se situe en tant que concurrent direct des solutions NoSQL, mais contrairement à ces solutions, il conserve une interface relationnelle via le SQL, ce qui est l'une de ses forces.

Notons enfin, que la plupart des solutions NewSQL proposent un stockage en mémoire. Ce stockage en mémoire distribué sur plusieurs machines sous forme de grille de données est largement utilisé depuis une dizaine d'années dans les environnements où une faible latence est requise, notamment dans certaines applications des banques d'investissements. Les solutions NewSQL se situent ainsi dans une position intermédiaire entre les solutions NoSQL et les grilles de données [13].

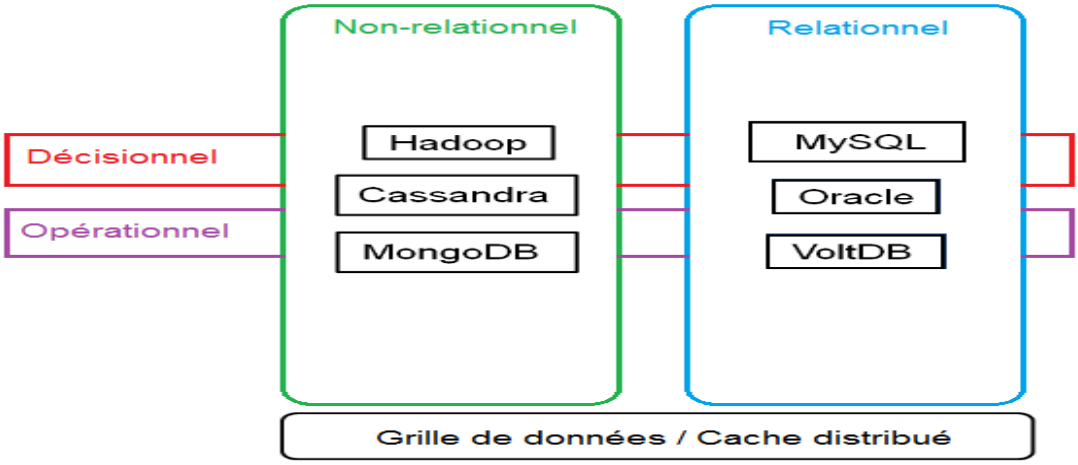


Figure II.2 : Naissance du NewSQL à partir de 3 architectures

III – Caractéristiques du NewSQL

Le NewSQL est une architecture qui reprend les avantages du NoSQL et comble son principal désavantage par une éventuelle cohérence des données grâce au support de transactions ACID via un langage unique de requêtage SQL [10].

Ce modèle est très récent et très prometteur. Voici ci-dessous quelques-unes de ces caractéristiques :

- Se base sur le SQL comme langage commun de requêtage.
- Supporte les contraintes ACID .
- Adopte un mécanisme qui évite d'imposer de verrous lors d'opérations concurrentes de lecture avec les opérations d'écritures. La lecture en temps réel en est ainsi facilitée.
- Une architecture distribuée fournissant de meilleures performances que les solutions classiques de type SGBDR.
- Scalabilité horizontale, architecture sans maître et capable de tourner sur un nombre important de nœuds sans souffrir de goulot d'étranglement.
- Base de données en mémoire.

IV – Architecture du NewSQL

L'architecture NewSQL n'est pas totalement nouvelle, elle reprend des expériences antérieures plusieurs caractéristiques tout en faisant des choix qui lui sont propres.

- Le choix d'une interface SQL.
- Un schéma relationnel avec des limitations pour faciliter la distribution des données et des traitements.
- Utiliser la distribution et la réplication des données pour assurer la scalabilité et la disponibilité des données [8].

V – Les catégories de bases de données NewSQL

Comme le NoSQL, il existe plusieurs catégories de systèmes NewSQL, basés sur les différentes approches adoptées par le propriétaire pour préserver l'interface SQL et aborder les problèmes de performance et d'évolutivité.

V.1 - New design

Dans ce modèle, les bases de données sont souvent écrites à partir de zéro avec une architecture distribuée selon le choix de l'utilisateur. Elles comprennent des composants tels que le contrôle distribué de la concurrence, le contrôle de flux et le traitement de requêtes distribuées. Ces bases sont utilisées pour fonctionner dans un cluster de nœuds distribués sans partage (shared-nothing), dans lequel chaque nœud possède un sous ensemble de données [8]. Citons par exemple : VoltDB, NuoDB, MemSQL, ClusstrixDB

V.2 - MySQL Engines

La deuxième catégorie est fortement orientée moteur de stockage optimisé pour SQL. Ces systèmes fournissent la même interface de programmation que SQL, mais adaptent mieux cette intégration [8]. Citons par exemple : ScaleDB, Tokutek

V.3 - Middleware

Ces systèmes fournissent une couche "sharding" qui permet de diviser automatiquement les bases de données sur plusieurs nœuds [8]. Citons par exemple : ScaleBase, DbShards

La figure ci-dessous illustre les différentes catégories des solutions NewSQL :

Figure II.3 : Les 3 catégories de base de données NewSQL

VI – Les leaders de la Technologie NewSQL

Les bases de données NewSQL sont essentiellement destinées aux entreprises qui gèrent des données sensibles et stratégiques, qui requièrent une certaine évolutivité, mais aussi une cohérence supérieure à ce qu'apportent les bases NoSQL. Voici quelques exemples de bases de données NewSQL, les plus utilisées dans le marché des entreprises :

VI.1 – Spanner

Spanner est une base de données hautement évolutive et globalement distribuée, conçue, construite et déployée par Google. Spanner fournit le niveau d'abstraction le plus élevé par rapport aux autres systèmes. C'est une base de données qui coupe les données sur plusieurs ensembles de machines dans des centres de données répartis dans le monde entier. Spanner prend en charge la réplication qui est utilisée pour une disponibilité globale et la localisation géographique et rétablit automatiquement les données dans les machines au fur et à mesure que la quantité de données ou le nombre de serveurs change, ainsi que la migration automatique des données à travers les machines pour 0équilibrer la charge et en réponse aux échecs [18].

	CLOUD SPANNER	TRADITIONAL RELATIONAL	TRADITIONAL NON-RELATIONAL
Schema	✓ **Yes**	✓ Yes	✗ No
SQL	✓ **Yes**	✓ Yes	✗ No
Consistency	✓ **Strong**	✓ Strong	✗ Eventual
Availability	✓ **High**	✗ Failover	✓ High
Scalability	✓ **Horizontal**	✗ Vertical	✓ Horizontal
Replication	✓ **Automatic**	↻ Configurable	↻ Configurable

Figure II.4 : Comparaison de Google Spanner avec une BDD SQL et NoSQL

VI.2 – ClustrixDB

ClustrixDB est une base de données SQL peer-to-peer très importante, conçue pour des applications à grande échelle, haute disponibilité et en croissance rapide. ClustrixDB est utilisé principalement à la fois pour exécuter un volume massif de transactions et aussi effectuer des analyses rapides en temps réel. ClustrixDB propose une base de données SQL échelonnée conçue à partir de la base de l'architecture scale-out du Cloud. ClustrixDB gère un volume de transaction massif et vous permet d'exécuter des analyses en temps réel sur vos données opérationnelles en direct sans les déplacer dans un autre système et élimine pratiquement les tâches d'opérations d'un administrateur de base de données. [18]

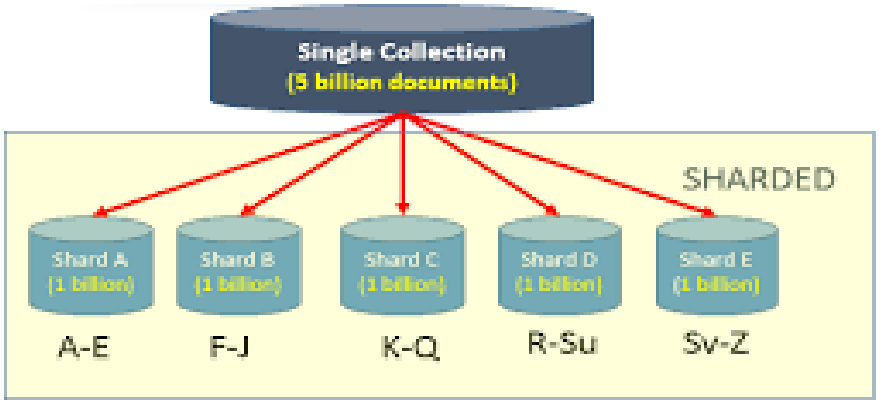


Figure II.5 : Exemple de partage de ClustrixDB

VI.3 – NuoDB

NuoDB est un système de gestion de base de données distribué avec une implémentation SQL supportant les transactions ACID. NuoDB fournit une plate-forme à l'échelle mondiale de gestion de données avec une garantie d'une haute disponibilité, des mises à niveau progressives, une redondance de données intégrée et une reprise après panne. NuoDB est une base de données multi-locataires qui s'échelle élastiquement offrant également des informations opérationnelles en temps réel [18].

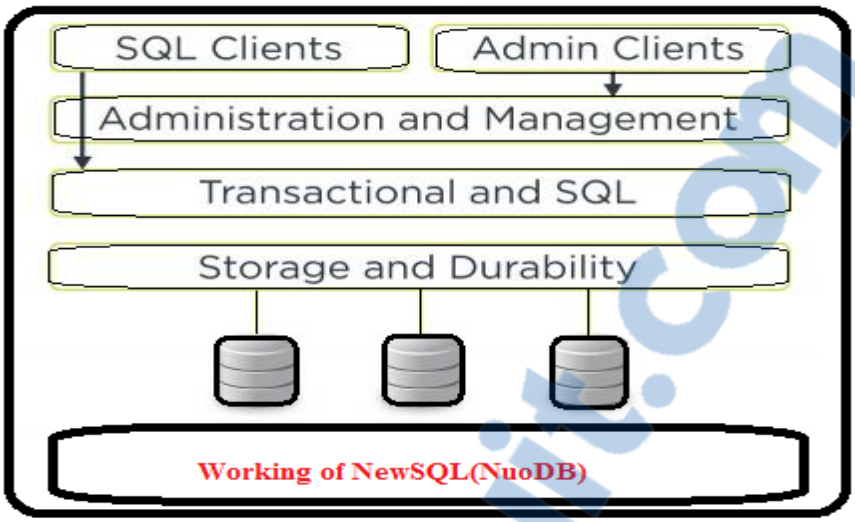


Figure II.6 : Architecture de NuoDB

VI.4 - TransLattice Elastic Database

Elastic TransLattice (TED) est un système de gestion de base de données évolutif et hautement disponible. C'est un serveur de base de données relationnel (SQL) géographiquement distribué conçu pour le traitement des transactions en ligne (OLTP) qui prend en charge les diverses charges d'application avec une haute disponibilité et des performances exceptionnelles pour des utilisateurs distants. La base de données Elastic TransLattice (TED) prend en charge certaines fonctionnalités importantes telles que la résilience, la scalabilité, la performance de l'utilisateur final, la migration vers le Cloud et des coûts nettement plus faibles.

Figure II.7 : Distribution géographique d'une base de données TED

VI.5 – SQLFire

SQLFire est une plate-forme de données en mémoire offrant une vitesse, une évolutivité dynamique, ainsi que la fiabilité et des capacités de gestion de données d'une base de données traditionnelle. Elle permet la résolution du problème de la transmission de données vers le cloud en mettant en œuvre une architecture partagée comme elle offre une haute disponibilité et une reprise après panne. SQLFire peut recevoir des données provenant de n'importe quelle application capable de lancer des appels comme le C++, C # et Java ainsi qu'elle achemine les appels de fonction vers les noeuds concernés sur le cluster sans interférence de l'appelant [18].

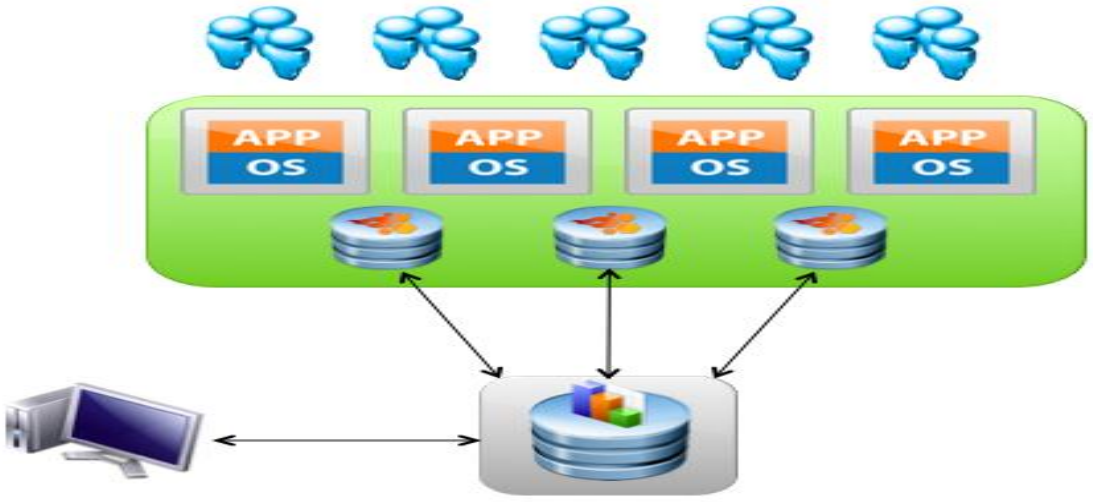


Figure II.8 : Amélioration des performances d'une application par SQLFire [14]

VI.6 – GridGain IMDB

GridGain IMDB prend en charge les fonctionnalités telles que les ensembles de données local, répliqués et partitionnés et permet de traverser de manière directe entre ces ensembles de données en utilisant la syntaxe standard SQL. GridGain IMDB prend en charge MapReduce, SQL distribué, MPP, MPI, RPC, un système de fichiers et fournit également des API fonctionnelles pour la plupart des opérations de base de données en mémoire telles que les projections de cache, les opérations basées sur les prédicats, etc. La cohérence dans GridGain IMDB est basée sur la mise en œuvre du contrôle simultané MultiVersion. GridGain IMDB prend pleinement en charge les transactions distribuées, la connectivité client distante via la connexion TCP directe ainsi que le protocole REST sur http [18].

La figure ci-dessous montre un IMDG classique avec un ensemble de clés de {k1, k2, k3} où chaque clé appartient à un nœud différent. Le composant de la base de données externe est facultatif. Si présent, les produits IMDG seront habituellement lus et écrit d'une façon automatique à partir de la base de données.

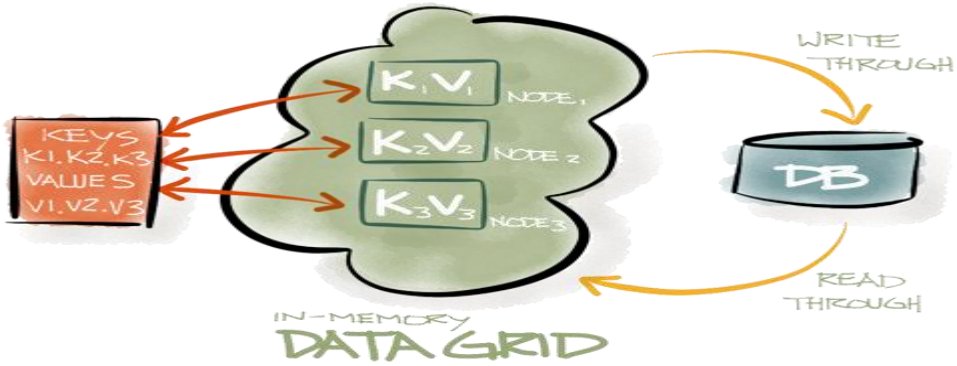


Figure II.9 : Lecture et écriture à partir d'une base de données GRID

VI.7 – Drizzle

Drizzle est une base de données très importante optimisée pour l'infrastructure Cloud et les applications Web. Elle est également conçue pour une simultanéité massive sur l'architecture multi-processeurs moderne. Drizzle est open source, Open community, & Open design possédant une architecture client / serveur et utilisant SQL comme langage principal. Drizzle est une base de données relationnelle compatible ACID, qui prend en charge les transactions via une conception MVCC et des points de plugin qui supportent la réplication, les moteurs de stockage, la réécriture des requêtes, les fonctions définies par l'utilisateur, les adaptateurs de protocole et les caches de requêtes multiples. [18]

VI.8 – MemSQL

MemSQL est une base de données distribuée en mémoire, un SGBDR conforme aux propriétés ACID, qui convertit les commandes SQL en générant du code C++. MemSQL s'appuie sur les structures de données sans verrouillage, le contrôle de concurrence multi-versions et un moteur d'exécution de requêtes conçu pour une vitesse et une efficacité maximale. MemSQL distribue facilement plusieurs copies de données sur des nœuds séparés dans des centres de données toujours sécurisées.

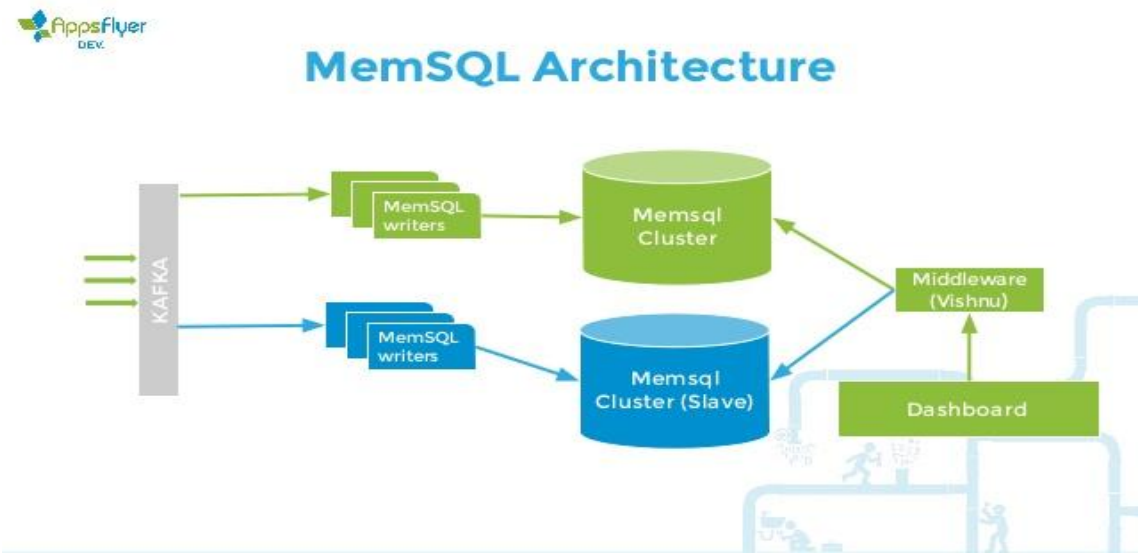


Figure II.10 : Architecture MemSQL [15]

VI.9 - VoltDB

VI.9.1 – Description

VoltDB est le dernier projet de Michael Stonebraker, qui a multiplié les créations dans le domaine des bases de données. Il a participé au développement d'Ingres (1983), Postgre (1989), PostgreSQL (1997). Il a fondé d'autres projets adressant des usages particuliers des bases de données comme StreamBase (2003), Vertica (2005) et finalement VoltDB qui a été lancé au marché le 28 janvier 2015. C'est donc un spécialiste des BD et un innovateur qui est à la base de VoltDB. Il a critiqué le mouvement NoSQL qui est une autre évolution du domaine, auquel il reproche le risque d'utilisation des langages de bas-niveau pour accélérer le traitement. Il préfère le mouvement NewSQL qui évolue le fonctionnement des bases de données, mais reste centré autour du langage SQL [19].

VoltDB est un nouveau produit de base de données révolutionnaire, conçu initialement pour être une meilleure solution pour des applications hautement performantes pour les entreprises. L'architecture VoltDB peut atteindre un débit 45 fois plus important que celui des produits de base de données actuels. L'architecture VoltDB permet également aux bases de données de s'étaler facilement en ajoutant des processeurs au cluster à mesure que le volume de données et les exigences de transaction augmentent [20]. VoltDB est conçu pour profiter pleinement de l'environnement informatique moderne :

- VoltDB utilise le stockage en mémoire pour maximiser le débit, en évitant un accès coûteux au disque.
- D'autres gains de performance sont obtenus en sérialisant tout accès aux données, en évitant de nombreuses fonctions dans les bases de données traditionnelles ralentissant le processus telles que le verrouillage, la maintenance des journaux de transactions.
- L'évolutivité, la fiabilité et la haute disponibilité sont obtenues grâce au regroupement et à la réplication entre les serveurs multiples [20].

VoltDB est une base de données transactionnelle entièrement compatible à la norme ACID, permettant au développeur d'applications de développer un code pour effectuer des transactions et gérer des renversements dans sa propre application. En utilisant SQL standard pour la définition de schéma et l'accès aux données, VoltDB réduit également la courbe d'apprentissage pour les concepteurs de bases de données expérimentés.

VI.9.2 – Domaine d'utilisation de VoltDB

VoltDB n'est pas destiné à résoudre tous les problèmes de base de données mais destiné seulement à un segment spécifique de l'informatique commerciale.

VoltDB se concentre spécifiquement sur les applications qui doivent traiter de grands flux de données rapidement. Cela comprend les applications financières, les applications des médias sociaux et les champs en plein essor de l'Internet. Les principales exigences pour ces applications sont l'évolutivité, la fiabilité, la haute disponibilité et un rendement exceptionnel.

VoltDB est utilisé aujourd'hui pour les applications traditionnelles à haute performance telles que les flux de données sur les marchés de capitaux, le commerce financier, les flux de données de télécommunications et les systèmes de distribution par capteurs. Il est également utilisé dans des applications émergentes comme les jeux en ligne, la détection de fraude, les échanges d'annonces numériques et les systèmes de micro-transaction. Toute application nécessitant un haut débit de base de données, une mise à l'échelle linéaire et une précision de données sans compromis bénéficieront immédiatement de VoltDB.

Cependant, VoltDB n'est pas optimisé pour tous les types de requêtes. Par exemple, VoltDB n'est pas le choix optimal pour la collecte et le rassemblement d'ensembles de données historiques extrêmement importants qui doivent être interrogés sur plusieurs tables. Ce type d'activité se retrouve généralement dans les solutions de business

intelligence et d'entreposage de données, pour lesquelles d'autres produits de base de données sont mieux adaptés.

Pour aider les entreprises qui exigent à la fois des performances exceptionnelles des transactions et des rapports ad hoc, VoltDB offre des fonctions d'intégration afin que les données historiques puissent être exportées vers une base de données analytique pour l'exploration de données à plus grande échelle.

Même si les diverses bases de données NewSQL diffèrent par leurs architectures internes, elles exploitent toutes le modèle de données relationnel et s'exécutent toutes en langage SQL [20].

VI.9.3 – Architecture et fonctionnalités du VoltDB

VoltDB est différent d'un produit de base de données traditionnelle ou chaque base de données VoltDB est optimisée pour une application spécifique en divisant les tables de base de données et les procédures stockées qui accèdent à ces tables sur plusieurs "sites" ou partitions sur une ou plusieurs machines hôtes pour créer la base de données distribuée. Puisque les données et les traitements sont partagés, plusieurs requêtes peuvent être exécutées en parallèle d'une part. D'autre part, chaque site fonctionne de façon indépendante, chaque transaction peut se terminer sans la surcharge de verrouillage des enregistrements individuels qui consomment une grande partie du temps de traitement des bases de données traditionnelles. Enfin, VoltDB équilibre les exigences de performance maximale avec la flexibilité pour s'adapter à moins d'intensité mais aussi avec le nombre important de requêtes qui traversent les partitions. Les sections suivantes décrivent ces concepts plus en détail.

VI.9.3.1 – Partitionnement

Dans VoltDB, chaque procédure stockée est définie comme une transaction. La procédure stockée (c'est-à-dire la transaction) réussit ou échoue dans son ensemble, assurant la cohérence de la base de données.

En analysant et précompilant la logique d'accès aux données dans les procédures stockées, VoltDB peut distribuer à la fois les données et le traitement associé aux partitions individuelles sur le cluster. De cette façon, chaque partition contient une «tranche» unique des données et de traitement de ces données. Chaque noeud du cluster peut prendre en charge plusieurs partitions [20].

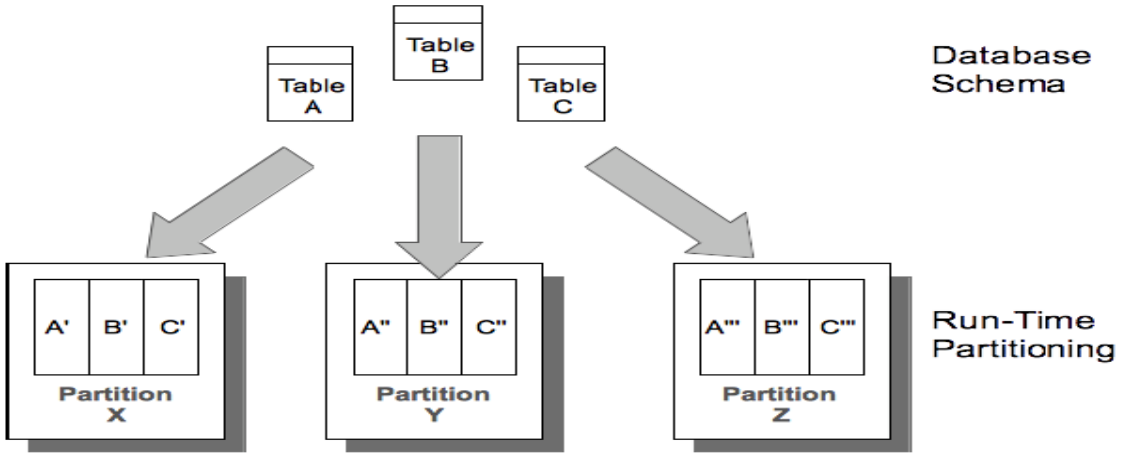


Figure II.11 : Partitionnement des tables dans VoltDB

VI.9.3.2 – Traitement de sérialisation

En cours d'exécution, les appels vers les procédures stockées sont transmis à la partition appropriée. Lorsque les procédures sont «partiellement partitionnées» (c'est-à-dire qu'elles fonctionnent sur des données dans une seule partition), le processus du serveur exécute la procédure par elle-même, libérant le reste du cluster pour traiter d'autres demandes en parallèle.

En utilisant le traitement sérialisé, VoltDB assure la cohérence transactionnelle sans surcharge des journaux de verrouillage et de transaction, tandis que le partitionnement permet à la base de données de gérer plusieurs requêtes à la fois. En règle générale, plus il y'as de processeurs (et donc plus de partitions) dans le cluster, plus les transactions VoltDB sont complétées par seconde, fournissant un chemin simple et presque linéaire pour la mise à l'échelle de la capacité et des performances d'une application.

Lorsqu'une procédure nécessite des données provenant de plusieurs partitions, un nœud agit en tant que coordinateur et distribue le travail nécessaire aux autres nœuds, collecte les résultats et termine la tâche. Cette coordination rend les transactions multi-partitionnées légèrement plus lente que les transactions à partition unique. Cependant, l'intégrité transactionnelle est maintenue et l'architecture de plusieurs partitions parallèles garantit que le débit est maintenu au maximum [20].

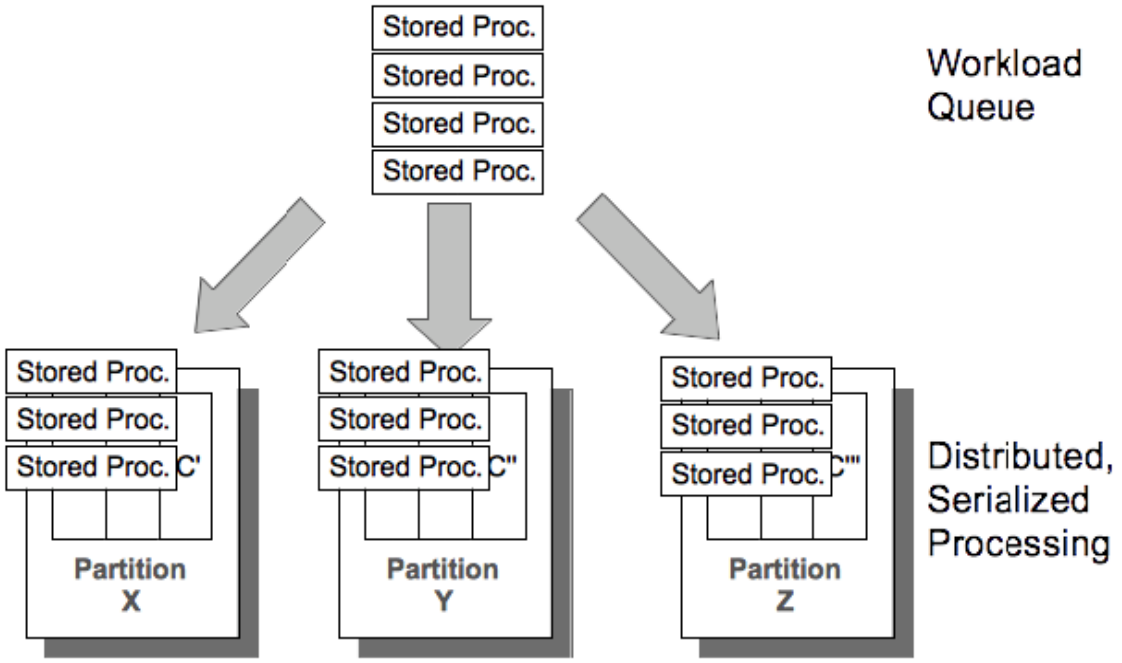


Figure II.12 : Traitement de sérialisation dans VoltDB

Il est important de noter que l'architecture VoltDB est optimisée pour le débit par rapport à la latence. La latence d'une seule transaction (le moment du début de la transaction jusqu'à la fin du traitement) est similaire dans VoltDB à d'autres bases de données, cependant, le nombre de transactions qui peuvent être effectuées en une seconde (débit) est d'ordre de grandeur plus élevé parce que VoltDB réduit le temps d'attente des demandes pour être exécutées. VoltDB atteint ce débit amélioré en éliminant les coûts requis pour le verrouillage et d'autres tâches administratives.

VI.9.3.3 – Partitionnement vs Tableaux répliqués

Les tables sont partitionnées dans VoltDB en fonction d'une colonne spécifiée par le développeur ou le concepteur. Lorsqu' on choisit des colonnes de partitionnement qui correspondent à la manière dont les données sont accessibles par les procédures stockées, elles optimisent l'exécution au moment de l'exécution.

Pour optimiser davantage les performances, VoltDB permet de reproduire certaines tables de base de données dans toutes les partitions du cluster. Pour les petites tables qui sont largement en lecture seule, ce qui permet aux procédures stockées de créer des jointures entre cette table et une autre table plus grande tout en restant une transaction à partition unique. Par exemple, une base de données de ventes en détail qui utilise des codes de produit comme clé primaire, peut avoir une table qui ne fait que coordonner le

code de produit avec la catégorie et le nom complet du produit, puisque ce tableau est relativement petit et ne change pas fréquemment (contrairement à l'inventaire et aux commandes), il peut être répliqué à toutes les partitions. De cette façon, les procédures stockées peuvent récupérer et renvoyer des informations de produit conviviales lors de la recherche par code de produit sans impact sur la performance des mises à jour et des recherches d'ordre et d'inventaire [20].

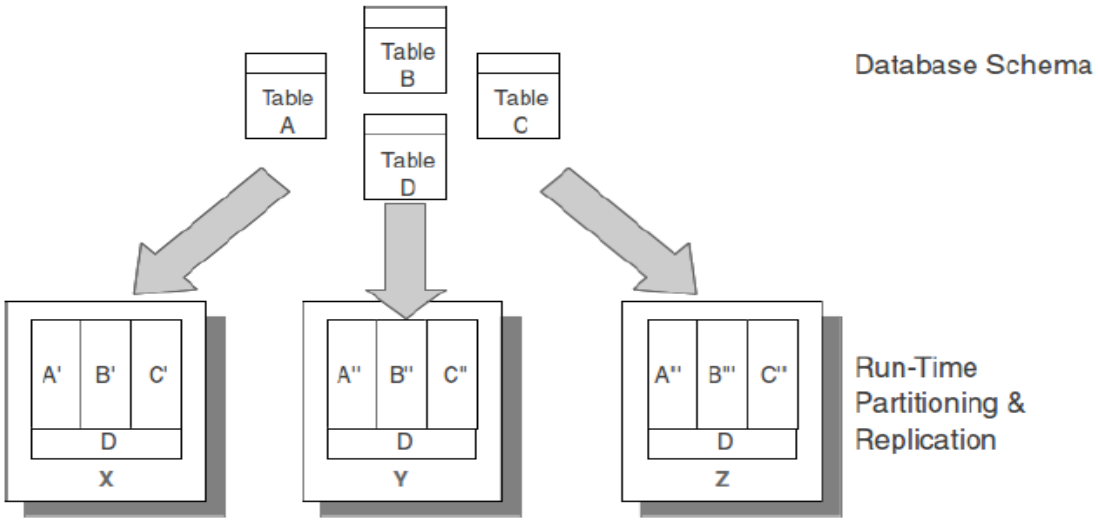


Figure II.13 : Tables de réplication VoltDB

VI.9.3.4 – Mise à l'échelle

L'architecture VoltDB est conçue pour simplifier le processus de mise à l'échelle de la base de données pour répondre aux besoins changeants de l'application. L'augmentation du nombre de nœuds dans un cluster VoltDB augmente le débit (en augmentant le nombre de files d'attente simultanées en fonctionnement) et augmente la capacité de données (en augmentant le nombre de partitions utilisées pour chaque table).

La mise à l'échelle d'une base de données VoltDB est un processus simple qui ne nécessite aucune modification du schéma de la base de données ou du code de l'application. On peut soit:

- Enregistrez la base de données (à l'aide d'une capture instantanée), puis redémarrez la base de données en spécifiant le nouveau nombre de nœuds pour

le cluster redimensionné et en utilisant la restauration pour recharger le schéma et les données.

- Ajoutez des nœuds "à la volée" pendant que la base de données est en cours d'exécution. [20].

VII – Les avantages du NewSQL

Cette nouvelle génération vise donc de régler tous les problèmes rencontrés à la fois avec le NoSQL, mais aussi avec les bases relationnelles. Voici quelques avantages :

- Le NewSQL propose un système aussi extensible que le NoSQL, tout en garantissant les mêmes propriétés qu'une base de données relationnelle traditionnelle via le support des requêtes SQL, ce qui est l'une de ces forces.
- Il conserve les propriétés ACID tout en ayant la performance et la scalabilité.
- La plupart des solutions NewSQL proposent un stockage en mémoire distribué sur plusieurs machines, sous forme de grille de données, ce qui permet d'augmenter la taille du stockage, et permet aussi à faire survivre la donnée à une coupure électrique d'une machine, vu que l'information est répliquée sur plusieurs machines.
- Le NewSQL offre une scalabilité horizontale simple en permettant une gestion beaucoup plus légère.
- Il existe à présent un nombre important de bases de données NewSQL open source qui permettent aux sociétés de choisir la base de données qui correspond le mieux à leurs besoins.

VIII – Les inconvénients du NewSQL

Aucun système n'est parfait, même les solutions NewSQL ont leurs inconvénients. Les principaux inconvénients sont les suivants :

- Les systèmes NewSQL ne sont pas encore normalisés autant que les systèmes SQL traditionnels.
- Les architectures en mémoire peuvent être inappropriées pour des volumes dépassant quelques téraoctets.
- Offre un accès partiel aux outils riches des systèmes SQL traditionnels [21].

IX – Conclusion

Le NewSQL est donc utilisé pour des stockages distribués. Il stocke potentiellement en mémoire, utilise en partie SQL comme langage de requête et offre des capacités compétitives et une architecture moderne qui correspond aux architectures récentes. NewSQL est plutôt destiné aux entreprises qui gèrent beaucoup de données (comme les grands groupes Google ou Amazon) nécessitent une certaine évolutivité mais aussi plus de cohérence de leurs données. A noter aussi que beaucoup d'entreprises ne font pas vraiment le choix entre ces nouvelles technologies NewSQL ou NoSQL ou même SQL. Ils utilisent différentes technologies de stockage de données pour les différents besoins Par exemple Facebook utilise MySQL pour stocker les posts ou les informations de l'utilisateur et Cassandra (NoSQL) pour les messages privés.

Chapitre III

Etude Comparative MySQL vs VoltDB

I - Introduction

Ce projet a pour objet de développer une étude comparative entre deux systèmes de gestion de base de données de générations différentes à savoir MySQL qui fait partie de l'ancienne génération SQL et VoltDB qui fait partie de la nouvelle génération NewSQL. La finalité de cette d'étude est d'offrir des éléments d'informations et des indicateurs pour orienter les utilisateurs pour des prises de décisions éventuelles pour le choix des solutions appropriées à leurs besoins et au contexte d'utilisation des données.

Donc ce chapitre, on va présenter les différents outils qui ont été employés dans notre étude, en plus particulier le benchmark YCSB et les solutions utilisées MySQL et VoltDB, suivi d'une analyse des résultats expérimentaux obtenus afin d'évaluer les performances de chaque modèle de base de données par rapport aux charges de travail exécutés sur ces SGBD.

Cette étude a été faite dans une seule machine PC portable DELL inspirion 15 3521 avec un processeur intel core i5-3337U CPU @ 1.80GHz, et une RAM de 6 Go sur un système d'exploitation Linux Mint 18.1 Cinnamon 64-bit.

II - Présentation et installation du benchmark

II.1 - Présentation d'YCSB

Avec l'apparition de nouvelles générations de bases de données, il était nécessaire de développer des outils qui permettent de confronter ces différentes solutions disponibles pour décider sur le système adéquat pour une application particulière, d'une part à cause de l'absence de normalisation de ces systèmes, et d'autre part à cause des différentes fonctionnalités développées selon les besoins spécifiques du domaine d'utilisation.

Le leader de ces outils de comparaison est YCSB (Yahoo!Cloud Serving Benchmark), qui est un framework open source conçu, pour évaluer et comparer les performances des différents types de systèmes de gestion de données.
Le noyau d'YCSB contient :

- Un générateur de charge de travail extensible
- Un ensemble 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 est configurée par un ensemble de paramètres comme le nombre d'enregistrements à charger, le nombre d'opérations à effectuer ainsi que la proportion de lecture, écriture et mise à jour des opérations. Voici quelques exemples :

- Workload A: 50% Read, 50% Update.
- Workload B: 95% Read, 5% Update.
- Workload C: 100% Read.
- Workload D: 50% Read, 50% Read-Modify-Write.
- Workload E: 5% Read, 95% Read-Modify-Write.
- Workload F: 100 % Read-Modify-Write [22].

Pour évaluer le temps de chargement, on va générer 600 000 enregistrements, chaque enregistrement est identifié par une clé d'enregistrement.

L'exécution des charges de travail consiste à lancer 1000 opérations sur la base de données sous forme de requêtes.

II.2 - Installation

Pour installer YCSB, une version récente de JAVA est pré-requise.

II.2.1 - JAVA

Il existe deux manières pour installer les archives binaires d'oracle JAVA JDK et JRE :

- La première c'est d'aller directement au site d'oracle pour télécharger le JDK :

http://www.oracle.com/technetwork/java/javase/downloads/index.html

- La deuxième c'est à l'aide des commandes suivantes exécutées dans le terminal:

```
sudo apt-add-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer
```

Après l'installation on peut vérifier la version du JDK avec cette commande :

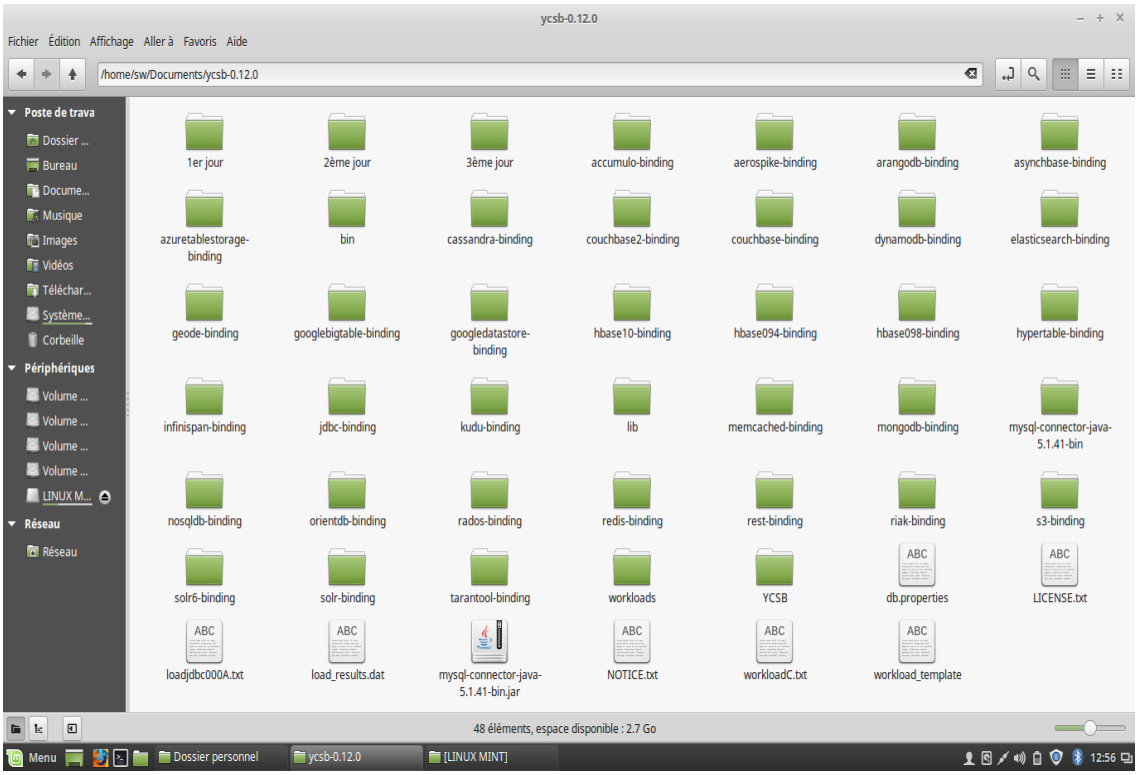
```
Java -version
```

II.2.2 - YCSB

Le programme YCSB est téléchargé à partir du lien suivant :

https://github.com/brianfrankcooper/YCSB/releases/tag/0.12.0

Après avoir téléchargé le fichier tar.gz, on l'a décompressé et voici ce qu'on a obtenu :



Notons que la version utilisée est 0.12.0.

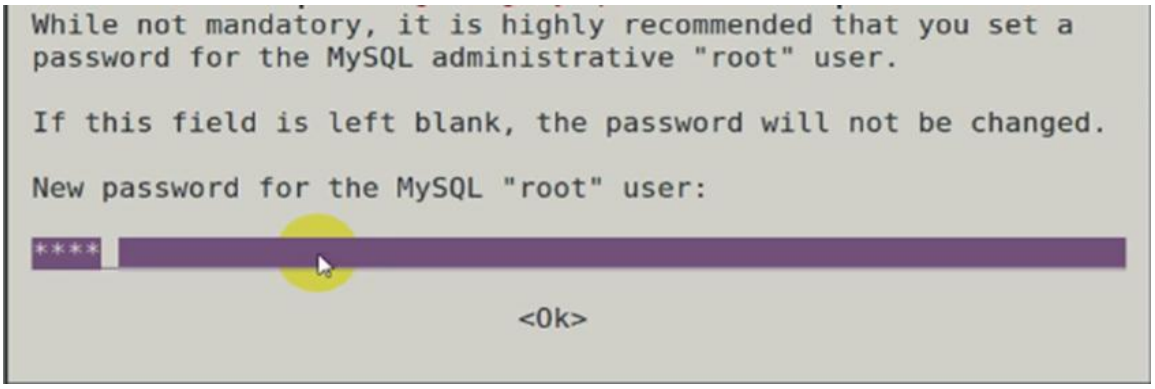
III - Installation et configuration des SGBD

III.1 - MySQL

- L'installation de MySQL a été faite à l'aide de la commande suivante :

```
Sudo apt-get install mysql-server
```

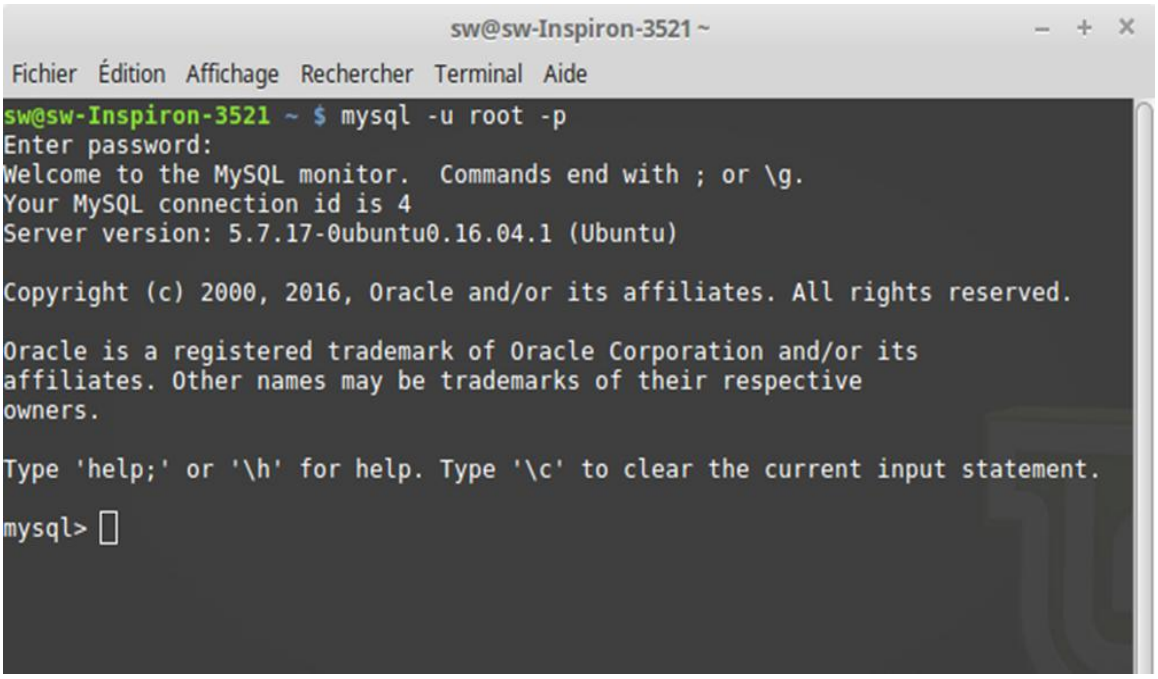
- Un mot de passe est requis pendant l'installation, pour l'utilisateur par défaut « root ».



- On lance MySQL à l'aide de cette commande :

```
mysql –u root -p
```

- Après l'insertion du mot de passe, voici le résultat :



- A la fin du processus d'installation, on doit configurer MySQL pour qu'il soit pris en charge par YCSB, pour cela il faut télécharger le mysql connector java qui est un pilote JDBC nécessaire pour communiquer avec les serveurs MySQL.
- Le téléchargement a été effectué à partir du lien suivant :

  https://dev.mysql.com/downloads/connector/j/

- La version utilisée de MySQL est la 5.7.17

III.2 - VoltDB

- Tout d'abord nous avons téléchargé VoltDB de son site officiel, la version utilisée est entreprise édition 7.1.1 :

  https://www.voltdb.com/try-voltdb/download-enterprise/

- Ensuite nous avons extrait le fichier téléchargé puis ouvrir un terminal dans le dossier obtenu après l'extraction.
- Puis on a utilisé des commandes pour créer le répertoire d'emplacement sur le disque et configurer la base de données.

VoltDB est une base de données qui s'exécute principalement en mémoire. Une gestion de mémoire appropriée est essentielle au bon fonctionnement des bases de données VoltDB.

(THP) Transparent Huge Pages est une autre fonctionnalité du système d'exploitation Unix qui optimise l'utilisation de la mémoire pour les systèmes exigeant une grande capacité de mémoire. THP modifie la cartographie de la mémoire pour une utilisation optimale d'un nombre important de pages physiques. Cela peut être utile pour l'informatique à usage général en exécutant plusieurs processus. Toutefois, pour les applications à forte intensité de mémoire telles que VoltDB, THP peut avoir un impact négatif sur les performances [20].
Par conséquent, il est important de désactiver les énormes pages transparentes (THP) sur les serveurs exécutant VoltDB.

- Les commandes suivantes, exécutées en tant que root ou depuis un autre compte privilégié, désactivent THP :

```
$ echo never>/sys/kernel/mm/transparent_hugepage/enabled
$ echo never>/sys/kernel/mm/transparent_hugepage/defrag
```

- Pour définir un répertoire racine pour toute activité sur disque requise au moment de l'exécution, il faut utiliser cette commande :

```
bin/voltdb create
```

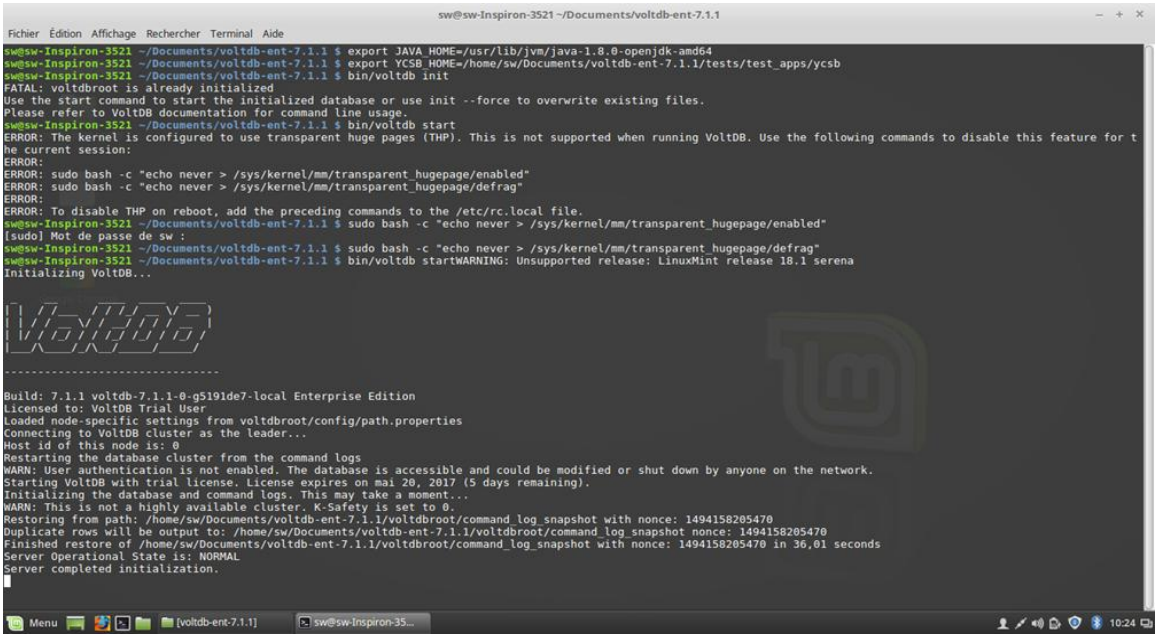
- Avant de lancer une base de données VoltDB, on doit initialiser le répertoire racine dans lequel VoltDB va stocker ses données de configuration, ses journaux et d'autres informations sur le disque. Une fois qu'on a initialisé le répertoire racine, on doit lancer la base de données. Les commandes pour l'initialisation et le démarrage de la base de données sont les suivantes :

```
bin/voltdb init
bin /voltdb start
```

- Il faut toujours exporter le chemin de l'environnement JAVA.

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64
```

L'image suivante résume ces différentes étapes :



- Après avoir créé et configuré le serveur on a eu le droit à une interface graphique en tapant cet URL : http://localhost:8080

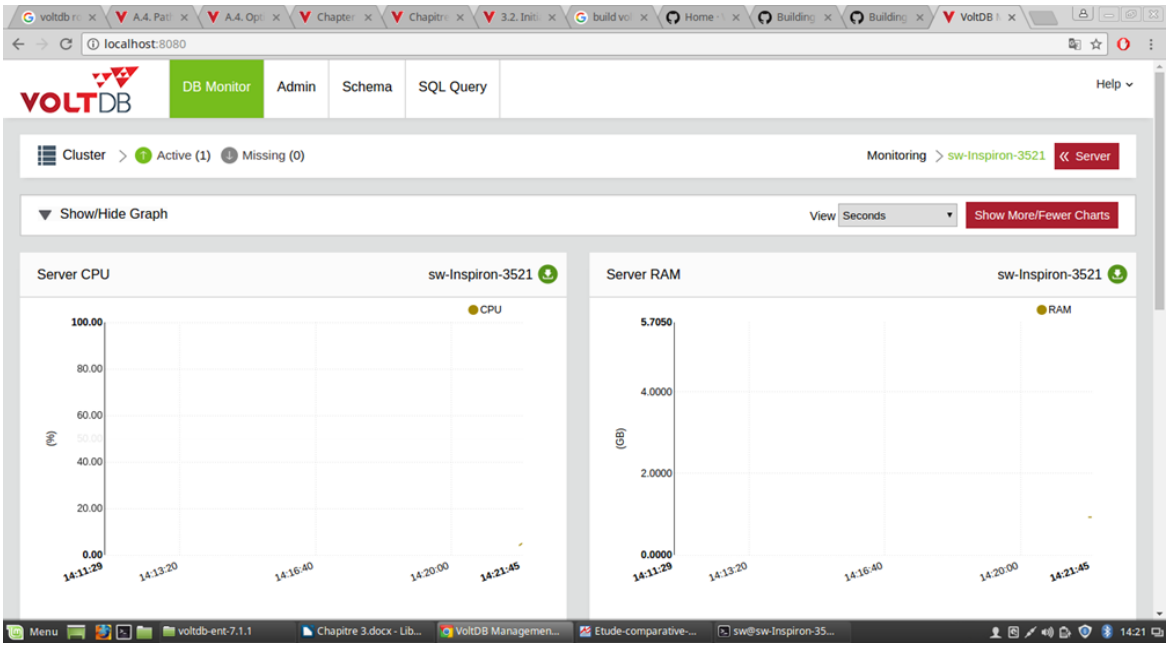


Figure III.1 : Interface graphique de VoltDB

- Configuration VoltDB avec YCSB :

YCSB est conçu initialement pour comparer entre les bases de données NoSQL, ce qui fait qu'il ne prend pas en charge le NewSQL. Pour cela on a téléchargé des fichiers JAVA publiés par John Hugg, un ingénieur fondateur chez Voltdb qui nous ont permis de configurer VoltDB avec YCSB.

En suivant ce lien, on trouve les différentes étapes à suivre :

https://github.com/VoltDB/voltdb/tree/master/tests/test_apps/ycsb

IV- Résultats expérimentaux

Dans cette section, nous allons présenter et analyser les résultats de chargement de 600 000 enregistrements générés par YCSB, ainsi que les différents tests effectués sous forme de charges de travail (Workloads) composées de 1000 opérations. Le principal critère de comparaison est le temps d'exécution, qui sera estimé par la moyenne des temps d'exécutions des Workloads obtenus pendant 3 jours différents.

IV.1 - Chargement des données

On va exposer dans ce qui suit les résultats de chargement des deux bases de données :

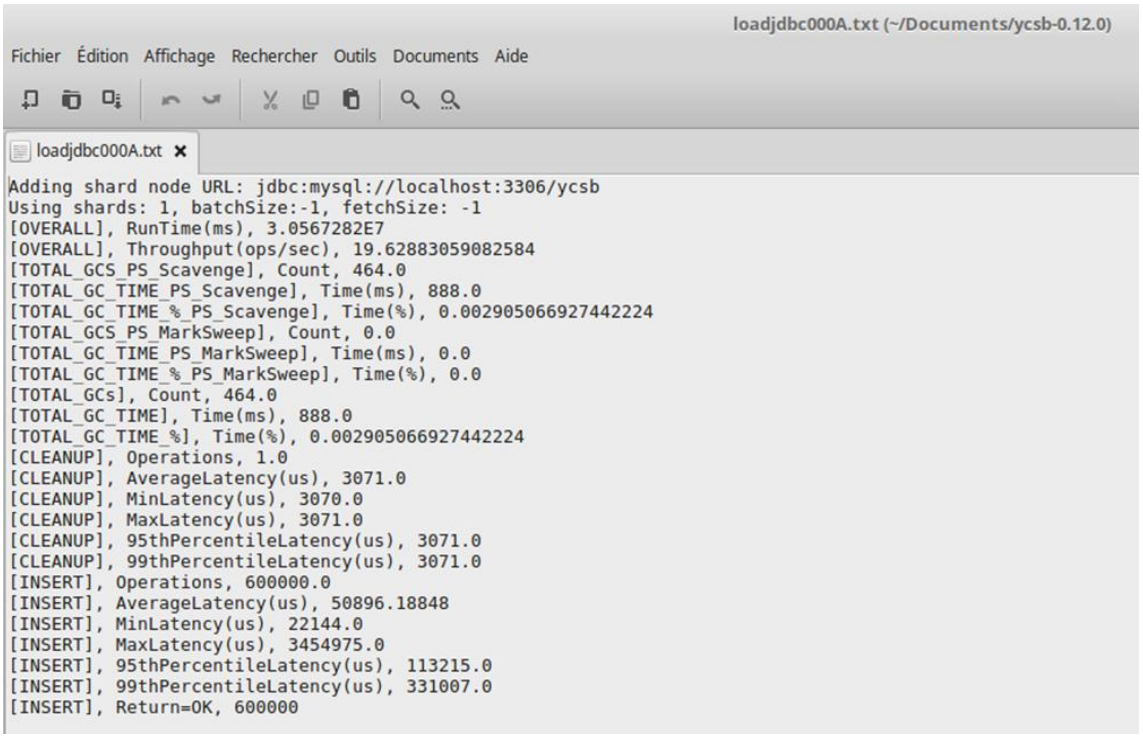
IV.1.1 - MySQL

Après avoir copié ce connecteur dans le dossier d'YCSB, on peut lancer la commande du chargement des 600 000 données :

```
bin/ycsb load jdbc -P workloads/workloada -p recordcount=600000 -P

db.properties -cp mysql-connector-java-5.1.41-bin.jar-s > loadjdbc000A.txt
```

Le résultat du chargement est obtenu dans un fichier texte « loadjdbc000A.txt » :



IV.1.2 - VoltDB

Après avoir lancé VoltDB, on ouvre un terminal dans le dossier YCSB puis exporter les deux chemins de JAVA et d'YCSB :

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64
export YCSB_HOME=/home/sw/Documents/voltdb-ent-7.1.1/tests/test_apps/ycsb
```

La commande de chargement pour VoltDB est la suivante :

```
./run.sh load
```

Le résultat du chargement donné par YCSB est le suivant :

Le tableau suivant montre le temps de chargement de chaque SGBD :

SGBD	VoltDB	MySQL
Temps (s)	110.039	30576.282

Tableau III.1 : Temps de chargement des deux SGBD

Dans ce premier test, nous pouvons constater que les meilleurs temps sont été inscrits par VoltDB qui était largement plus rapide par rapport à MySQL dans cette première phase de chargement.

IV.2 - Execution des workloads

Les charges de travail vont contenir 1000 opérations exécutées sur les 600 000 enregistrements, dont voici les résultats :

Workload A (50% read / 50% update) :

- **MySQL**

La commande est la suivante :

```
bin/ycsb run jdbc -P workloads/workloada -p recordcount=600000 -P
db.properties -cp mysql-connector-java-5.1.41-bin.jar-s > loadjdbc000A.txt
```

Voici le journal donné par YCSB :

workA.txt

```
Adding shard node URL: jdbc:mysql://localhost:3306/ycsb
Using shards: 1, batchSize:-1, fetchSize: -1
[OVERALL], RunTime(ms), 35431.0
[OVERALL], Throughput(ops/sec), 28.223871750726765
[TOTAL_GCS_PS_Scavenge], Count, 0.0
[TOTAL_GC_TIME_PS_Scavenge], Time(ms), 0.0
[TOTAL_GC_TIME_%_PS_Scavenge], Time(%), 0.0
[TOTAL_GCS_PS_MarkSweep], Count, 0.0
[TOTAL_GC_TIME_PS_MarkSweep], Time(ms), 0.0
[TOTAL_GC_TIME_%_PS_MarkSweep], Time(%), 0.0
[TOTAL_GCs], Count, 0.0
[TOTAL_GC_TIME], Time(ms), 0.0
[TOTAL_GC_TIME_%], Time(%), 0.0
[READ], Operations, 509.0
[READ], AverageLatency(us), 15399.888015717092
[READ], MinLatency(us), 497.0
[READ], MaxLatency(us), 825855.0
[READ], 95thPercentileLatency(us), 32767.0
[READ], 99thPercentileLatency(us), 65535.0
[READ], Return=OK, 509
[CLEANUP], Operations, 1.0
[CLEANUP], AverageLatency(us), 2761.0
[CLEANUP], MinLatency(us), 2760.0
[CLEANUP], MaxLatency(us), 2761.0
[CLEANUP], 95thPercentileLatency(us), 2761.0
```

- **VoltDB**

```
./run.sh workload workloada
```

Le résultat est le suivant :

```
[OVERALL], RunTime(ms), 655.0
[OVERALL], Throughput(ops/sec), 1526.7175572519084
[TOTAL_GCS_PS_Scavenge], Count, 4.0
[TOTAL_GC_TIME_PS_Scavenge], Time(ms), 39.0
[TOTAL_GC_TIME_%_PS_Scavenge], Time(%), 5.9541984732824424
[TOTAL_GCS_PS_MarkSweep], Count, 2.0
[TOTAL_GC_TIME_PS_MarkSweep], Time(ms), 54.0
[TOTAL_GC_TIME_%_PS_MarkSweep], Time(%), 8.244274809160306
[TOTAL_GCs], Count, 6.0
[TOTAL_GC_TIME], Time(ms), 93.0
[TOTAL_GC_TIME_%], Time(%), 14.198473282442748
[READ], Operations, 506.0
[READ], AverageLatency(us), 30131.045454545456
[READ], MinLatency(us), 1887.0
[READ], MaxLatency(us), 150911.0
[READ], 95thPercentileLatency(us), 130559.0
[READ], 99thPercentileLatency(us), 139903.0
[READ], Return=OK, 506
[CLEANUP], Operations, 150.0
[CLEANUP], AverageLatency(us), 156.57333333333332
[CLEANUP], MinLatency(us), 2.0
[CLEANUP], MaxLatency(us), 14543.0
[CLEANUP], 95thPercentileLatency(us), 7.0
[CLEANUP], 99thPercentileLatency(us), 6995.0
[UPDATE], Operations, 494.0
[UPDATE], AverageLatency(us), 31577.937246963564
[UPDATE], MinLatency(us), 1322.0
[UPDATE], MaxLatency(us), 145407.0
[UPDATE], 95thPercentileLatency(us), 130047.0
[UPDATE], 99thPercentileLatency(us), 140543.0
[UPDATE], Return=OK, 494
sw@sw-Inspiron-3521 ~/Documents/voltdb-ent-7.1.1/tests/test_apps/ycsb $
```

SGBD	VoltDB	MySQL
Temps (s)	3.33	32.16

Tableau III.2 : Temps d'exécution des deux SGBD pour le Workload A

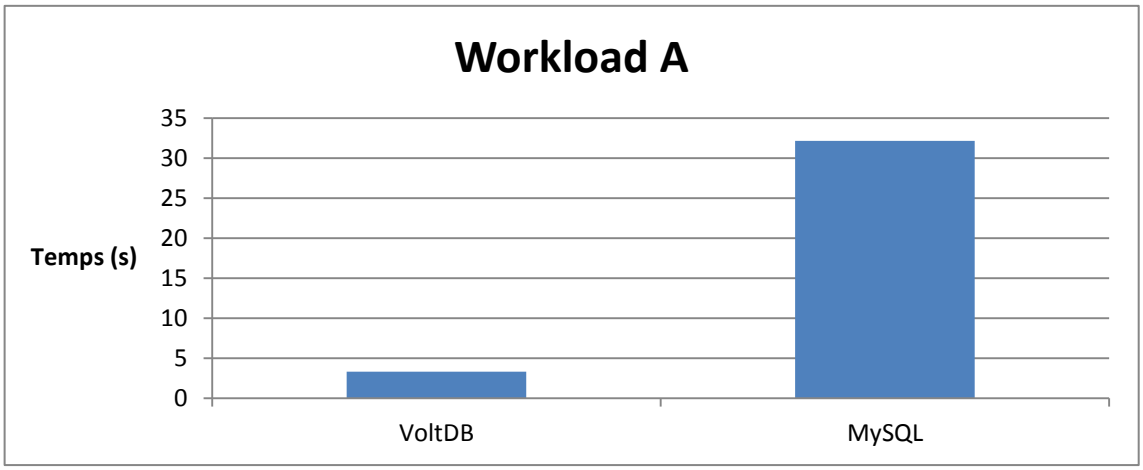


Figure III.2 : Workload A

Workload B (95% Read, 5% Update) :

SGBD	VoltDB	MySQL
Temps (s)	1.81	12.23

Tableau III.3 : Temps d'exécution des deux SGBD pour le Workload B

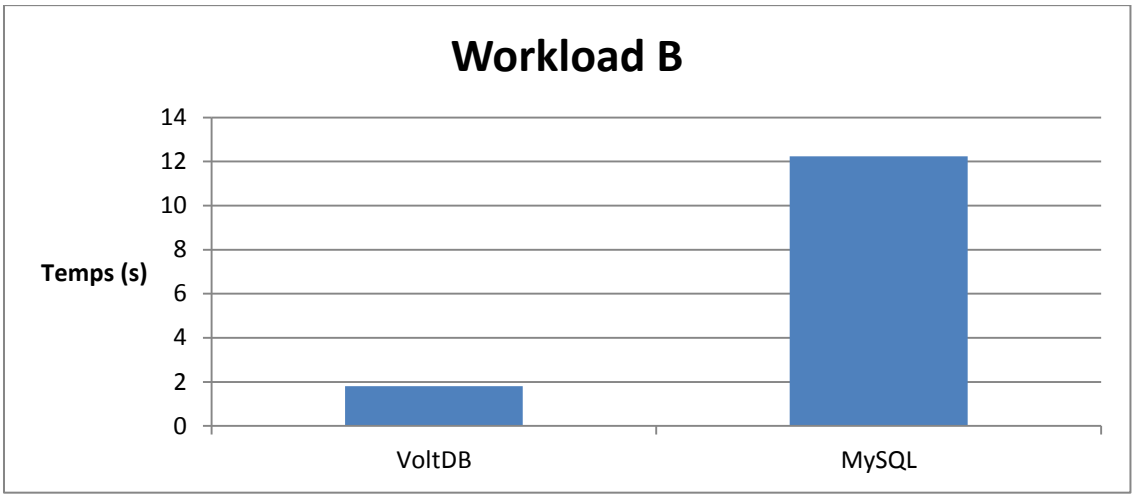


Figure III.3 : Workload B

Workload C (100% Read) :

SGBD	VoltDB	MySQL
Temps (s)	1.08	8.47

Tableau III.4 : Temps d'exécution des deux SGBD pour le Workload C

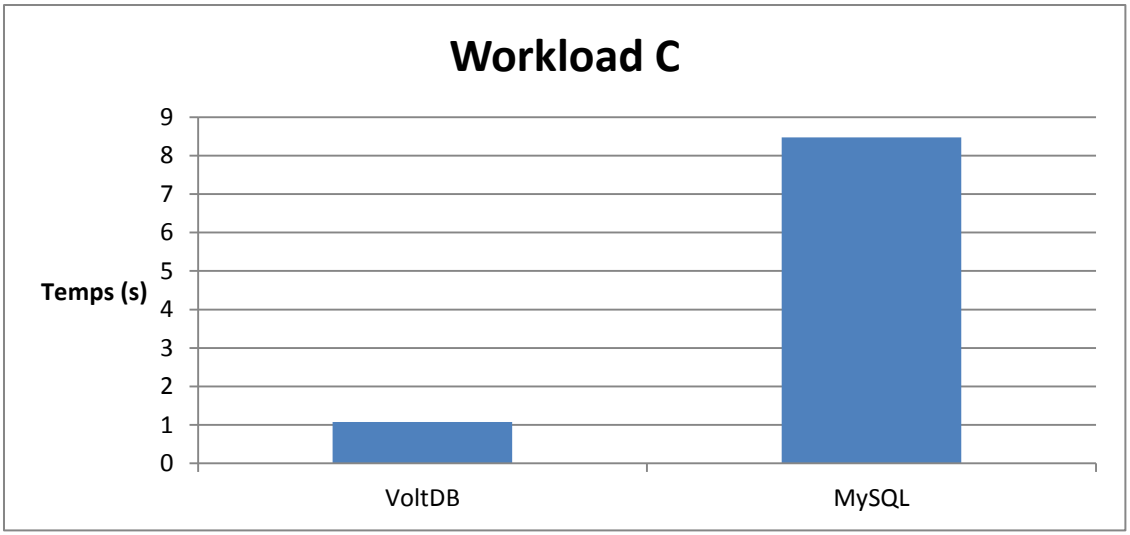


Figure III.4 : Workload C

Workload D (50% Read, 50% Read-Modify-Write):

SGBD	VoltDB	MySQL
Temps (s)	0.95	28.02

Tableau III.5 : Temps d'exécution des deux SGBD pour le Workload D

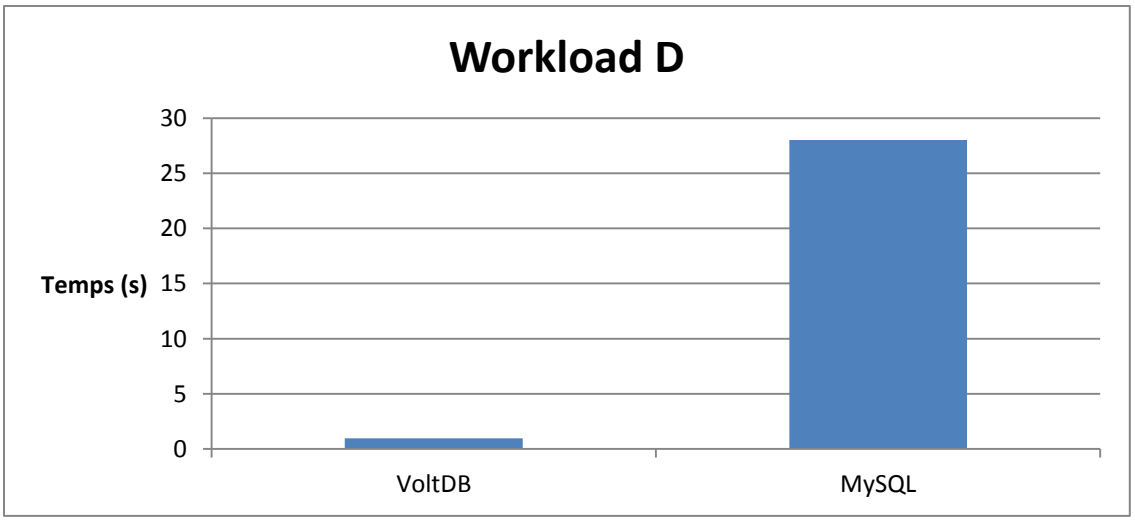


Figure III.5 : Workload D

Workload E (5% Read, 95% Read-Modify-Write) :

SGBD	VoltDB	MySQL
Temps (s)	10.26	48.65

Tableau III.6 : Temps d'exécution des deux SGBD pour le Workload E

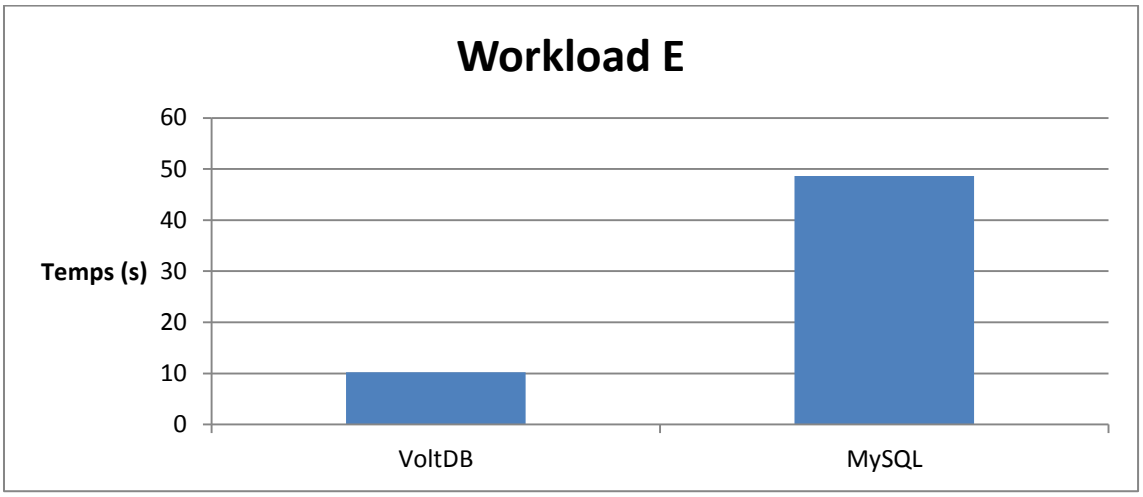


Figure III.6 : Workload E

Workload F (100 % Read-Modify-Write):

SGBD	VoltDB	MySQL
Temps (s)	3.9	52.85

Tableau III.7 : Temps d'exécution des deux SGBD pour le Workload F

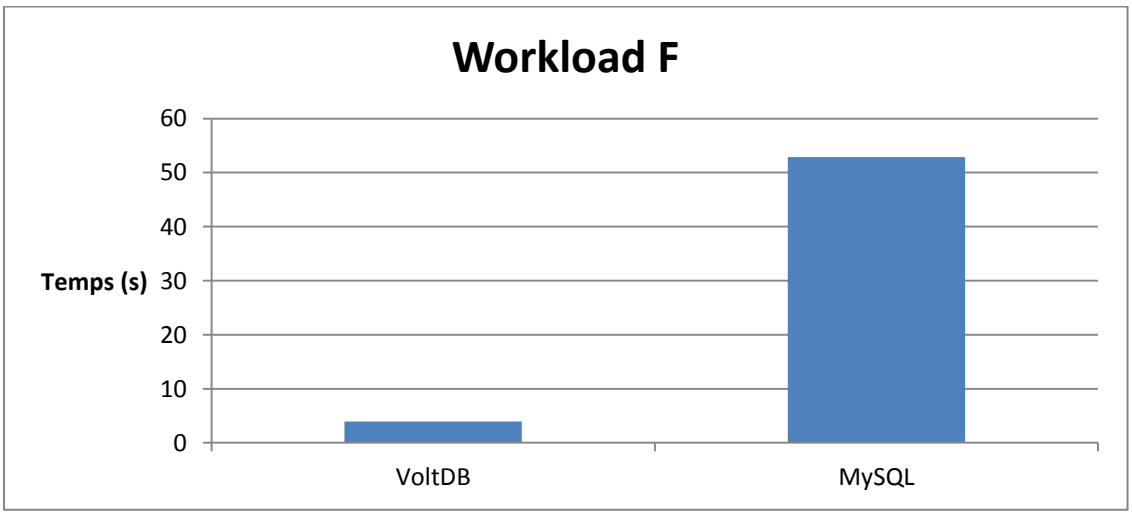


Figure III.7 : Workload F

Les résultats des tests ont été effectués pendant trois journées différentes, on a pris la moyenne et voilà le schéma global des résultats :

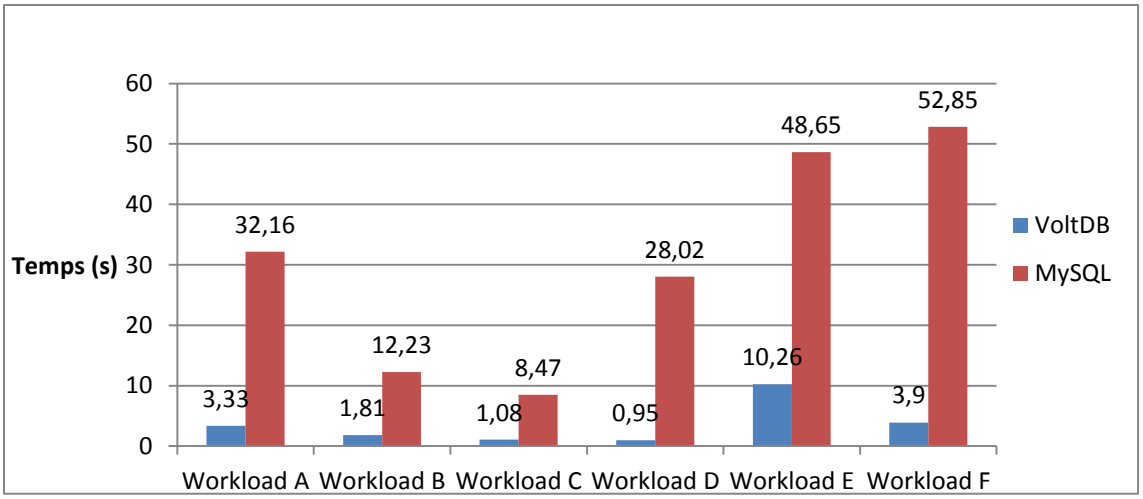


Figure III.8 : Temps d'exécution global des charges A, B, C, D, E, F,

IV.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 SQL (MySQL) et NewSQL (VoltDB) en s'appuyant sur le temps d'exécution des différents Workloads, composées de 1000 opérations chacune.

Sur le premier test de chargement de données, VoltDB a été très performante en inscrivant de grands écarts par rapport à MySQL avec 110 secondes pour VoltDB contre 9 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 VoltDB qui s'appuie sur une gestion de mémoire spécifique.

Les différents résultats obtenus après exécution des Workloads, ont montré que VoltDB est très performante par rapport à MySQL dans toutes les charges de travail, notamment celles qui contiennent des opérations d'écriture lourdes dans la base de données à savoir : A (50% mise à jour), D (50% lecture-modification-écriture), E (95% lecture-modification-écriture), F (100 % lecture-modification-écriture). Nous pouvons justifier cette performance de VoltDB ainsi que la lenteur de MySQL par les exigences strictes 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 NewSQL qui proposent un stockage en mémoire distribué sur plusieurs machines en respectant les propriétés ACID d'une manière raisonnable avec une cohérence éventuelle.

En revanche, MySQL a produit ses meilleurs résultats dans l'exécution du Workload C comportant seulement des opérations de lecture.

Après la lecture des résultats obtenus en exécutant les six charges de travail, nous pouvons conclure que la solution SQL MySQL peut être 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 NewSQL VoltDB.

Notons que VoltDB a consommé plus de 80% de mémoire vive pour l'exécution des différentes charges de travail.

V - Conclusion

Dans ce chapitre, nous avons présenté en premier lieu les différentes étapes pour mettre en œuvre les deux solutions de gestion de données à savoir VoltDB représentant la nouvelle génération NewSQL et MySQL représentant les systèmes relationnels traditionnels, puis on a exécuté une série de tests de chargement, de lecture ou d'écriture sur les deux bases de données qui nous ont permis de tirer un ensemble de renseignements sur les deux modèles.
On peut affirmer que VoltDB est plus performante que MySQL dans tous les Workloads exécutées dans un environnement à grande échelle composé de 600 000 enregistrements.

Conclusion générale et perspectives

L'explosion des quantités d'informations numériques, qui a induit à un changement d'échelle des volumes, nombres et types, a imposé aux différents acteurs ces dernières années, de nouveaux challenges et les a incités à concevoir et développer de nouvelles technologies pour mieux 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 des mouvements NoSQL et NewSQL.

Beaucoup d'entreprises et compagnies utilisant des SGBD relationnels classiques dits «SQL», depuis plusieurs années, veulent basculer vers ces nouvelles générations de systèmes de gestion de données, pour coller aux nouvelles tendances et anticiper l'explosion de leurs données dans le futur.

La mouvance NoSQL s'est proposée comme alternative aux systèmes transactionnels traditionnels qui ont atteint leurs limites, néanmoins avec beaucoup de renoncements sur les principes du paradigme relationnel, notamment le non-respect des propriétés ACID.

Pour y remédier, la mouvance NewSQL a émergé en se plaçant entre ces 2 générations et essaye de tirer les atouts des deux rivaux SQL et NoSQL.

Pour justifier et motiver un éventuel choix ou un basculement vers un nouveau système, 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 VoltDB en tant que nouveau modèle, représentant la nouvelle ère NewSQL.

Les différents tests effectués sous forme de charges de travail étaient basées sur le temps d'exécution pour évaluer les performances, à l'aide du benchmark développé par Yahoo YCSB (Yahoo! Cloud Serving Benchmark).

Les tests effectués s'initient par l'insertion de 600 000 enregistrements dans chaque base de données, puis se poursuit par l'exécution de six charges de travail composées de 1000 opérations, chacune, de nature différente : mise à jour ou lecture.

L'analyse des résultats obtenus affirme que VoltDB était très performante par rapport à MySQL, dans toutes les charges de travail, avec une différence remarquable. Ceci peut

justifier par la dimension choisie (600 milles enregistrements), une échelle favorable pour les solutions NewSQL qui ont été conçu initialement pour ce passage à l'échelle.

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

Rappelons enfin, que notre objectif majeur était d'explorer le domaine du NewSQL, et comparer une solution très répondue VoltDB avec un système classique très répondue aussi MySQL, néanmoins cette étude peut être développé et prolongé sur plusieurs aspects :

- Etaler notre étude à d'autres solutions NewSQL et SQL (Oracle, SQL Server,..).
- Développer d'autres études comparatives avec les systèmes NoSQL.
- Accroître et diminuer le nombre d'enregistrements à charger.
- Augmenter le nombre d'opérations à exécuter.
- Varier les charges de travail en créant d'autres.

Références Bibliographiques

[1] Connaissances informatique – Définition des fichiers plats. http://www.ordinateur.cc/syst%C3%A8mes/Comp%C3%A9tences-informatiques-de-base/203366.html

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

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

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

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

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

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

[8] Vincent MESNIER, Présentation de NewSQL, Octobre 2015. http://air.imag.fr/index.php/NewSQL#Enseignants_:

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

[10] Furrer Hadrien, SQL, NoSQL, NewSQL stratégie de choix, Mémoire présenté en vue de l'obtention du diplôme Bachelor HES, Filière Informatique de gestion ; Haute École de Gestion de Genève (HEG-GE).Février 2015.

[11] Meyer Léonard, L'AVENIR DU NoSQL Quel avenir pour le NoSQL ?, 2014. onardmeyer.com/wp-content/uploads/2014/06/avenirDuNoSQL.pdf

[12] Fatima-Zahra Benjelloun, Ayoub Aitlahcen, Samir Belfkih , BIG DATA: LE QUOI? LE POURQUOI ET LE COMMENT, LGS, ENSA, Université Ibn Tofail, Kénitra, Maroc, 2015.

[13] Hadi Hashem, Modélisation intégratrice du traitement BigData, Thèse de Doctorat de l'Université Paris Saclay, 19 septembre 2016.

[14] Using SQLFire to improve application performance and scalability, Mai 2012. http://cliffdavies.com/blog/vmware/using-sqlfire-to-improve-application-performance-and-scalability/

[15] Journey to the realtime – Analytics in extreme growth. https://www.slideshare.net/MemSQL/journey-to-the-realtime-analytics-in-extreme-growth

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

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

[18] Rakesh Kumar et al, International Journal of Computer Science and Mobile Computing, Vol.3 Issue.5, May- 2014, pg. 434-438

[19] SCRIPTOL – Voltdb rend oracle obsolète. https://www.scriptol.fr/logiciel/voltdb.php

[20] VoltDB Community Edition - Using VoltDB.PDF, V7.2, 21 avril 2017. https://docs.voltdb.com/

[21] Dataconomy Media GMBH – SQL vs NoSQL vs NewSQL trouver la bonne sulution http://dataconomy.com/2015/08/sql-vs-nosql-vs-newsql-finding-the-right-solution/

[22] NEWS & MEDIA – Yahoo Cloud Serving Benchmark, avril 2010 https://research.yahoo.com/news/yahoo-cloud-serving-benchmark#link

Liste des Abréviations

BDDR : Base De Données Relationnelles.

SQL : Structured Query Language.

SGBD : Système de Gestion de Base de Données.

SGBDR : Système de Gestion de Base de Données Relationnelles.

ACID : Atomicity, Consistancy, Isolation, Durability.

NoSQL : Not only SQL.

CAP : Consistency, Availability, Partition tolerance.

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

CRUD : Create, Read, Update, Delete.

JSON : JavaScript Object Notation.

BSON : Binary JSON.

HDFS : Hadoop Distributed File System.

MPP : Massively Parallel Processing

MPI : Master Patient Index

RPC : Remote Procedure Call

IMDB : Internet Movie Database

Liste des Figures

Liste des Tableaux

Résumé

Les SGBDs relationnels ont pris une place importante sur le marché des SGBD. Ce modèle bien que très puissant, a atteint ses limites pour les environnements distribués. Les nouvelles solutions NewSQL proposent une alternative ou complètent les fonctionnalités des SGBDR pour offrir des solutions plus adéquates dans les systèmes à grandes échelles. Beaucoup d'utilisateurs des SGBD relationnels classiques dits « SQL » veulent basculer vers ces nouvelles solutions «NewSQL» pour répondre à ces nouvelles exigences.

Dans le cadre de notre PFE, nous développons une étude comparative sur les performances de deux solutions très répondues de l'ancienne génération SQL (MySQL) et la nouvelle NewSQL (VoltDB) à l'aide de l'outil YCSB. Le premier objectif à atteindre était d'explorer le monde du NewSQL, le deuxième était de justifier le choix d'une solution par rapport à une autre et 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, NewSQL, MySQL, VoltDB, YCSB, Charge de travail

Abstract

Relational DBMSs have taken an important place in the DBMS market. This model, although very powerful, has reached its limits for distributed environments. NewSQL databases complement RDBMS features to provide more appropriate solutions in large scale systems. Many users of traditional SQL relational databases want to switch to these new "NewSQL" solutions to meet these new requirements.

As part of our ESP, we are developing a comparative study on the performance of two popular solutions of the old generation SQL (MySQL) and the new NewSQL (VoltDB) using the tool YCSB end of studies' project. The first objective was to explore the world of NewSQL, the second was to justify the choice of a solution in relation to another and to provide the necessary assistance to the interested parties for possible decisions on The choice of solutions to be adopted.

Keywords: SQL, NewSQL, MySQL, VoltDB, YCSB, Workload

ملخص

لقد اتخذت انظمة تسيير قواعد البيانات العلائقية مكانة هامة في سوق انظمة تسيير قواعد البيانات. هذا النموذج بالرغم من أن اداءه الجيد، قد وصل حدوده في المجالات الموزعة. قواعد البيانات NewSQL تقدم بدائلا أو تكمل وظائف انظمة تسيير قواعد البيانات العلائقية لتقديم حلول أكثر ملائمة للأنظمة ذات المقاييس الكبيرة. العديد من مستخدمين نظم إدارة قواعد البيانات SQL يريدون الميل أو التحول إلى الحلول الجديدة NewSQL لمواكبة الاحتياجات المعاصرة.

في اطار مشروعنا لنهاية الدراسة، قمنا بإنجاز دراسة مقارنة بين الأنظمة القديمة SQL (MySQL) و الأنظمة الحديثة NewSQL (VoltDB) باستخدام اداة YCSB. الهدف الأول المسطر من هذه الدراسة هو استكشاف عالم NewSQL، الهدف الثاني هو تبرير اختيار حل من الحلول، وتقديم المساعدة اللازمة إلى الجهات المعنية لاتخاذ القرارات المحتملة لاختيار الحلول المناسبة لاستعمالاتهم في مجال تسيير المعلومات.

الكلمات المفتاحية: عبء العمل , SQL, NewSQL, MySQL, VoltDB, YCSB, Workload