REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE ABOU-BEKR BELKAID - TLEMCEN

THÈSE

Présentée à :

FACULTE DES SCIENCES – DEPARTEMENT D'INFORMATIQUE

Pour l'obtention du diplôme de :

DOCTORAT EN SCIENCES

Spécialité : Informatique

Par :

Mr MATALLAH Houcine

Sur le thème

Vers un nouveau modèle de stockage et d'accès aux données dans les Big Data et les Cloud Computing

---

Soutenue publiquement en Septembre 2018 à Tlemcen devant le jury composé de :

Mr CHIKH Med Amine	Professeur	Université de Tlemcen	Président
Mr BELALEM Ghalem	Professeur	Université d'Oran1	Directeur de thèse
Mr BOUAMARNE Karim	Professeur	Université d'Oran1	Co-Directeur de thèse
Mr ATMANI Baghdad	Professeur	Université d'Oran1	Examinateur
Mme HAMDADOU Djamila	Maître de Conférences A	Université d'Oran1	Examinatrice
Mr BENMAMMAR Badr	Maître de Conférences A	Université de Tlemcen	Examinateur

Laboratoire de Recherche Informatique Tlemcen (LRIT)
BP 119, 13000 Tlemcen - Algérie

A mon père et mon frère Abdelhadi

A mon beau-frère Professeur Boutiba Zitouni

Que Dieu les accueille dans son vaste paradis

A ma chère mère

A ma chère femme

A mes enfants Moncif, Rihab, Oussama et Lokmane

A mes sœurs et mon frère et leurs petites familles

A ma belle famille

A mes amis et collègues

A toutes les personnes qui m'aiment

Remerciements

Aucune œuvre humaine ne peut se réaliser sans l'aide de Dieu. Je le remercie en premier lieu de m'avoir donné la santé, le courage ainsi qu'une grande volonté pour aboutir à ce travail qui au début paraissait une mission difficile, vu toutes mes diverses responsabilités assumées.

Comme toute thèse, ce mémoire est le fruit de longues heures de lecture, de recherches, de réflexion et le résultat d'un effort constant, cet effort n'aurait pu aboutir sans la contribution d'un nombre de personnes que je tiens à remercier.

Tout d'abord, j'exprime ma double gratitude à Mr BELALEM Ghalem, en tant que directeur de thèse pour ses conseils, orientations, son assistance permanente et pour tous les efforts fournis durant ces années de travail. Je réitère les mêmes gratitudes exprimées à Mr BOUAMARANE Karim Professeur et chef département Informatique à l'université d'Oran en tant que co-directeur de thèse.

J'exprime ma profonde reconnaissance à Mr CHIKH Mohammed Amine de me faire l'honneur de présider le jury et mes plus sincères remerciements à l'égard de Mr ATMANI Baghdad, Mme HAMDADOU Djamila et Mr BENMAMMAR Badr pour l'intérêt qu'ils ont bien voulu porter à ce travail en acceptant de l'examiner et d'en être rapporteurs.

Je remercie chaleureusement ma famille pour son soutien et ses encouragements tout au long de cette thèse.

Enfin, ce travail de recherche n'aurait jamais été terminé sans le soutien de plusieurs personnes qui n'ont pas hésité à me donner le courage, l'endurance et le dynamisme pour l'accomplir. Qu'ils trouvent ici l'expression de mes sincères remerciements.

Merci à tous et à toutes.

Résumé

La révolution technologique intégrant de multiples sources d'informations, la vulgarisation de l'informatique dans les différents secteurs et domaines ont amené à l'explosion de la volumétrie des données, qui reflète le changement d'échelle des volumes, du nombre et de types. Ces accroissements massifs ont poussé à l'évolution des manières de gestion, de stockage, de localisation et d'accès aux données. Les dernières étapes de cette évolution informatique ont émergé de nouvelles technologies : Cloud Computing et Big Data.

Le Big Data est un ensemble de technologies basées sur les bases de données NoSQL « Not Only SQL » permettant le passage à grande échelle en volumes, en nombres et en types de données. Les grandes entreprises du domaine informatique voient dans les nouveaux systèmes NoSQL, de nouvelles solutions permettant de répondre à leurs besoins d'évolutivité. Plusieurs solutions open-source et payantes de modèles NoSQL sont disponibles sur le marché.

Dans la première contribution, nous développons une étude comparative sur les performances de six solutions NoSQL très répandues dans le marché, à savoir : MongoDB, CouchBase, Cassandra, HBase, Redis, OrientDB à l'aide du Benchmark 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 à savoir YCSB. La finalité est d'apporter l'assistance et l'aide nécessaire aux acteurs intéressés de Big Data et de Cloud Computing pour d'éventuelles prises de décision sur le choix de la meilleure solution appropriée pour leurs entreprises.

Notre deuxième contribution consiste à proposer des remaniements internes dans l'architecture de l'implantation de référence des Clouds de stockage et de Big Data, à savoir HDFS d'Hadoop. Ce dernier adopte un service de métadonnées séparé aux données avec isolation et centralisation des métadonnées par rapport aux serveurs de stockage de données. Nous proposons une approche qui permet d'améliorer le service de métadonnées d'HDFS, afin de maintenir la cohérence sans compromettre les performances des métadonnées en employant un parallélisme modéré des métadonnées, entre centralisation et distribution des métadonnées, dans le but d'accentuer la performance et l'extensibilité du modèle.

Mots clés : Cloud Computing, BigData, NoSQL, YCSB, Hadoop, HDFS, MapReduce, Métadonnées.

Abstract

The technological revolution integrating multiple information sources and extension of computer science in different sectors led to the explosion of the data quantities, which reflects the scaling of volumes, numbers and types. These massive increases have resulted in the development of new locations techniques and access to data. The final steps in this evolution have emerged new technologies : Cloud Computing and Big Data.

Big Data is a set of technologies based on NoSQL databases allowing scalability of volumes, numbers and types of data. The important companies in the IT sector find these NoSQL systems, new solutions to respond to scalability needs. Multiple open-source and proprietary models of NoSQL are available on the market.

In the first contribution, we develop a comparative study about the performance of six solutions widely employed MongoDB, Couchbase, Cassandra, HBase, Redis and OrientDB, using the YCSB tool. The latter being a very powerful test tool, used in multiple evaluation works of NoSQL databases. The purpose is to provide assistance and support to actors interested of Big Data and Cloud Computing for eventual decisions for the choice of solutions to be adopted.

Our second contribution consists in proposing internal revisions in the architecture of the reference implementation of storage clouds and Big Data, namely HDFS of Hadoop. This latter is based on the separation of metadata to data that consists in the centralization and isolation of the metadata of storage servers. We propose an approach to improve the service metadata for Hadoop to maintain consistency without much compromising performance and scalability of metadata by suggesting a mixed solution between centralization and distribution of metadata to enhance the performance and scalability of the model.

Keywords: Cloud Computing, BigData, NoSQL, YCSB, Hadoop, HDFS, MapReduce, Metadata.

ملخـــص

أدت الثورة التكنولوجية التي أدمجت المصادر المتعددة للمعلومات، وكذا تعميم تكنولوجيا المعلومات في مختلف القطاعات والمجالات إلى انفجار حجم البيانات، مما يعكس تغير السلم عل مستوى الأحجام، الأعداد والأنواع. وقد أدت هذه الزيادات الهائلة إلى تغييرات في طرق تسيير، تخزين والتوصل إلى موقع البيانات. المراحل الأخيرة من تطور المعلوماتية أفرز تقنيات جديدة: الحوسبة السحابية والبيانات الضخمة.

البيانات الضخمة هي مجموعة من التقنيات المبنية على قواعد بيانات NoSQL "ليس فقط SQL"، مما يسمح بقياس واسع النطاق في الأحجام والأرقام وأنواع البيانات. ترى الشركات الكبيرة في مجال تكنولوجيا المعلومات، في أنظمة NoSQL الجديدة، حلولًا جديدة لتلبية احتياجات قابلية التوسع، حيث أن العديد من الحلول المفتوحة المصدر والمدفوعة الثمن من طراز NoSQL، متوفرة بقوة في السوق.

في أول إسهام لنا، قمنا بإنجاز دراسة مقارنة لأداء ستة حلول NoSQL جد معروفة في المجال: MongoDB، CouchBase، Cassandra، HBase، Redis، OrientDB باستخدام YCSB Benchmark. هذا الأخير هو أداة معروفة جيدا لقوة اختبارها حيث تستخدم في العديد من أعمال التقييم لقواعد البيانات NoSQL، ألا وهي YCSB. الهدف المرجو هو توفير التوجيهات والمساعدة اللازمة للمهتمين في مجال البيانات الكبيرة والحوسبة السحابية لاتخاذ قرارات محتملة بشأن اختيار أفضل الحلول لشركاتهم.

يتمثل إساهمنا الثاني في اقتراح مراجعات داخلية في بنية التنفيذ المرجعي للحوسبة السحابية والبيانات الضخمة HDFS ل Hadoop، حيث يعتمد هذا الأخير خدمة للبيانات الوصفية منفصلة عن المعطيات المخزنة مع عزل وتمركز البيانات المصنفة عن خوادم تخزين البيانات. نقترح نهجًا يحسِّن خدمة البيانات الوصفية لنظام HDFS للمحافظة على التناسق دون المساس بأداء البيانات الوصفية عن طريق استخدام التوازي المعتدل للبيانات الوصفية، بين المركزية وتوزيع البيانات الوصفية، للتأكيد على أداء وقابلية تطوير النموذج.

الكلمت المفتاحية : Hadoop , YCSB , NoSQL ، البيانات الضخمة ، الحوسبة السحابية ، البيانات الوصفية ، HDFS, MapReduce .

Tables des matières

Introduction générale

Contexte

La pléthore de sources de création de données numériques (Internet, Ordinateurs, Téléphones portables, Appareils photos numériques, Appareils électroménagers, Capteurs,..) et la vulgarisation de l'informatique dans les différents secteurs et domaines (E-Commerce, E-Administration, E-Gouvernement, Astrologie, Météorologie, MultiMedia, Cinéma et Télévision Numérique, Archives Numériques, Bibliothèques Numériques, Réseaux sociaux,…) ont fait exploser la volumétrie des données, qui reflète le changement d'échelle sur trois dimensions : en volumes, en nombres et en types de ces données. Plusieurs qualifications ont été employées dans la littérature pour caractériser ce phénomène comme « Déluge de données », « Tsunami de données », « Explosion de données », « Boom des données ».

Il est très difficile d'estimer les quantités de données numériques produites chaque jour dans le monde des entreprises, des administrations et des particuliers, qu'il s'agisse de photographies, de vidéoclips, de textes, de tweets ou d'e-mails.

Différentes approches de traitement des données existent selon qu'il s'agit de données en mouvement ou de données au repos. Les données en mouvement seraient utilisées si une entreprise est capable d'analyser la qualité de ses produits pendant le processus de production pour éviter des erreurs coûteuses. Les données au repos seraient utilisées pour une analyse commercialle pour mieux comprendre les habitudes d'achat des clients en fonction de tous les aspects de la relation client, y compris les ventes, les données sur les médias sociaux et les interactions avec le service client.

Les conceptions informatiques depuis les années quatre-vingt-dix avaient recours aux entrepôts de données (Data Warehouse) qui sont généralement centralisés dans des serveurs connectés à des baies de stockage. Ces architectures s'avéraient difficilement scalable (ajout de puissance à la demande). Alors face à l'extension du volume des données et face à la grande variété des données multi structurées (Vidéo, image, texte, web, etc.) ainsi que la fréquence dont ces données sont générées, les SGBD traditionnels et même les Data Warehouses ont eu du mal à s'adapter.

L'emplacement des données vis-à-vis de l'ordinateur qui la traite, et la manière d'y accéder ont constamment évolué depuis les débuts de l'informatique. L'ordre de grandeur de l'exaoct et entraîne déjà plusieurs problématiques pour le stockage et l'accès aux données.

Introduction générale

Aucun système de gestion de fichiers traditionnel ni aucun système de gestion de donnés classique ne peut prendre en charge un tel nombre d'informations avec des temps de réponses acceptables. Alors, ce passage à l'échelle en volumétrie a imposé de nouvelles réflexions, de nouveaux besoinset de nouveaux challenges qui ont amené à chercher un changement d'échelle de performance et tenter de surpasser une performance en débit de plusieurs Go/S atteint par les systèmes de fichiers parallèles à un débit supérieur au To/s. Ce débit recherché sous-entend que les clients sont nombreux et éparpillés géographiquement et la relation entre utilisateurs et stockage repose sur les réseaux distants.

Plusieurs études spécialisées se sont focalisées sur cette expansion perpétuelle de l'univers numérique pour présenter des prévisions dans l'optique d'anticiper et inférer des indicateurs sous forme d'alertes pour les acteurs du marché de l'IT.

Ces nouvelles dimensions, qui reflètent le changement d'échelle des volumes, nombres et types, ont imposé aux différents chercheurs depuis quelques années, de nouveaux défis qui les ont poussés à concevoir de nouvelles technologies et infrastructures pour contenir et traiter ces volumes énormes de données. Il s'agit de découvrir de nouveaux ordres de grandeur concernant la capture, la recherche, le partage, le stockage, l'analyse et la présentation des données. Beaucoup de concepts « inséparables » dominent actuellement le marché de l'IT, on entend souvent de « Cloud Computing », hébergeant des « Big Data » sous forme de «NoSQL» et traité par un simple programme « MapReduce » dans des « Clusters » distribués partout dans le monde.

Problématique 1

Dans une architecture Cloud articulée sur plusieurs milliers de nœuds, les solutions de gestion de données traditionnelles relationnelles deviennent inexploitables, d'où la nécessité de créer de nouveaux systèmes nativement adaptés à un environnement distribué de type Cloud, qui peuvent gérer et répartir dynamiquement les données sur plusieurs nœuds et supporter la scalabilité horizontale.

Un autre problème lié aux systèmes de bases de données relationnelles apparaît avec l'expansion exponentielle des données qui se compte désormais en péta-octets avec une vitesse accrue est liée aux contraintes ACID du modèle relationnel. Dans ces gigantesques volumes de données, ces contraintes, bien qu'elles garantissent la cohérence des données, peuvent devenir un facteur de blocage contraignant à la performance. Un exemple typique c'est celui des recherches dans Internet où le consommateur d'informations s'intéresse plus à

avoir une réponse instantanée sans autant exiger que ces informations soient à jour instantanément.

Cette nouvelle ère informatique avec ses nouvelles exigences a obligé les différents acteurs dans le domaine à chercher les meilleures solutions pour s'adapter à ces changements forcés qui a ont conduit aux nouveaux concepts Big Data et ont concouru à l'émergence du mouvement NoSQL.

Plusieurs solutions NoSQL open-source et propriétaires ont été conçues, développés et déployés par les grands acteurs du domaine comme Google, Microsoft, Yahoo, Facebook, Twitter, Amazon, etc., pour héberger dans leurs serveurs ces grands volumes de données; néanmoins l'absence de normalisation est un aspect marquant de cette mouvance NoSQL et la panoplie de solutions existantes dans le marché, mettent les différents décideurs devant un embarras dans le choix du modèle approprié par rapport à leur environnement d'exploitation, qui amène à avancer une vraie problématique sur la meilleure solution NoSQL à adopter pour répondre à leurs besoins ?

Contribution 1

Notre contribution consiste à fournir un ensemble de critères et indicateurs, aux acteurs intéressés, pour des prises de décisions éventuelles sur les solutions appropriées pour leurs entreprises, en développant une étude comparative sur un ensemble de solutions NoSQL très déployées dans le marché.

Notre étude accentue sur six systèmes de gestion de données NoSQL caractérisées par l'implémentation dans leurs noyaux du même algorithme « MapReduce », il s'agit des modèles MongoDB, Cassandra, HBase, Redis, CouchBase et OrientDB. Pour évaluer et comparer les solutions NoSQL disponibles, plusieurs benchmarks ont été conçus, le plus couramment utilisé est le YCSB. Plusieurs études comparatives dans le même contexte, utilisant YCSB, ont été développées récemment. L'étude avec laquelle on va comparer nos résultats a été menée dans une seule machine [1].

Problématique 2

Les difficultés potentielles de mise en œuvre des Clouds est l'éparpillement géographique en raison de l'externalisation des données dans un autre pays, voire un autre continent, ce qui amène à transporter des données et des métadonnées sur des réseaux distants. Si la difficulté de localisation des données est prise en compte dès le concept des Clouds, la localisation des

métadonnées est plus difficile à décider. Ainsi la contrainte technique principale à l'extensibilité des Clouds demeure la gestion des métadonnées.

Principalement, il existe deux approches de localisation des métadonnées centralisées ou distribuées. Le modèle P2P applique une distribution totale des métadonnées en employant un parallélisme très fort, à tous les niveaux de données et de métadonnées. Le deuxième modèle, considéré comme l'implantation de référence des Clouds de stockage et des Big Data ces dernières années, à savoir le HDFS d'Hadoop, opte pour un service de métadonnées centralisé afin de garantir la fiabilité du service et l'intégrité des métadonnées, avec un parallélisme massif pour le transport des données. Néanmoins, la centralisation de ce service dans un seul serveur, peut contraindre le fonctionnement du système HDFS en provoquant un goulot d'étranglement qui peut affecter les performances du processus vu l'accroissement massif très rapide des données qui engendre automatiquement un autre accroissement massif des métadonnées.

Contribution 2

Le modèle de référence dans le domaine du Cloud Computing et du Big Data à savoir le système de gestion de fichiers distribués d'Hadoop HDFS, centralise toutes les métadonnées dans une seule machine, cependant la montée en charge peut affecter le fonctionnement du système. Pour remédier au goulot d'étranglement qui peut être causé par le serveur unique de métadonnées d'HDFS, nous proposons une solution mixte entre centralisation et distribution des métadonnées, une approche qui permet de rétablir le service de métadonnées d'Hadoop afin d'améliorer la performance et l'extensibilité du modèle.

Le modèle conçu propose des remaniements internes dans l'architecture HDFS, en employant un parallélisme modéré des métadonnées. L'idée est de scinder les métadonnées en sous-ensembles qui sont gérés différemment, celles qui sont nécessaires à la navigation des utilisateurs et qui renseignent sur les données proprement dites, peuvent être distribuées sur plusieurs machines contrairement à l'architecture initiale, pour les rapprocher à leurs données respectives et alléger le trafic vers le serveur principal de métadonnées.

Organisation de la thèse

Cette thèse est constituée de deux parties principales A et B :

La première est intitulée « L'ère de Big Data et Cloud Computing ». Cette partie A est organisée en 4 chapitres :

Introduction générale

1. Dans le premier chapitre, on va explorer l'univers du Big Data et le Cloud Computing.
2. Les nouvelles exigences imposées par le passage aux nouvelles dimensions qui ont amené aux deux concepts précédents et les limites des systèmes traditionnels seront évoquées dans la première partie du chapitre suivant. Dans la deuxième partie du même chapitre, on va présenter les nouveaux systèmes de gestion de données conçus pour répondre aux nouveaux besoins exigés par le passage à l'échelle, à savoir le NoSQL et le NewSQL.
3. Dans le chapitre 3, nous allons accentuer sur les six solutions NoSQL : MongoDB, Cassandra, HBase, Redis, CouchBase et OrientDB, qui feront l'objet de notre étude comparative.
4. Après l'évaluation des performances de chaque base de données, les différents résultats expérimentaux de cette étude comparative seront synthétisés et confrontées aux résultats obtenus par Veronika Abramova et al. [1] dans Le chapitre 4.

   Nous terminons partie A par une synthèse et quelques perspectives de nos travaux futurs.

La seconde partie B du document est intitulée « Hadoop, HDFS et Mapreduce ». Elle est organisée en 3 chapitres :

1. Un état de l'art des différents systèmes de gestion fichiers va être décrit dans le premier chapitre.
2. Dans le chapitre 2, Les différentes architectures d'implantation des Clouds de stockage seront évoquées en premier lieu, l'écosystème Hadoop, ainsi que son système de gestion de fichiers HDFS seront présentés en deuxième lieu. Dans la dernière partie du chapitre, nous allons accentuer sur le concept de métadonnées dans un environnement à grand échelle puisqu'elles seront au cœur de notre modèle proposé.
3. Dans le troisième et dernier chapitre, nous proposons une implantation hybride basée sur une nouvelle distribution des métadonnées.

PartieA

L'ère du Big Data et Cloud Computing

Partie A : L'ère du Big Data et Cloud Computing

Introduction

Les dernières études montrent que la croissance de la volumétrie des données stockées informatiquement est environ 30% par an (Giga, Tera, Peta, Exa, Zetta...) [2, 3], générées par les multiples sources d'informations. Selon l'étude de l'université de Berkley en 2003, 5 exaoctets ($5x10^{18}$ oct) de données ont été créés au cours de l'année 2002 parmi lesquelles 92% ont été stockées informatiquement. Dans son rapport d'expertise établi en 2010 (Active Archiving Survey), IDC estime que le volume global des données numériques devrait atteindre 35 zettaoctets ($35x10^{21}$ oct) en 2020 [4]. En 2010, le PDG de Google déclare que : « Tous les 2 jours nous nous créons autant de données que ce qui a été créé jusqu'au 2003 », c'est des chiffres qui s'appliquent aux données produites par le grand public. Pour avoir une idée de l'accroissement exponentiel de la masse de données, selon d'autres études, on considère que 90 % des informations ont été produites durant les années où l'usage d'internet et des réseaux sociaux a connu une forte croissance. L'ensemble de toutes les données engendrées depuis le début de l'informatique jusqu'à la fin de l'année 2008, conviendrait maintenant aux volumes de ceux qui sont générés chaque minute.

Sur le plan de connectivité, Intel affirme que le nombre d'objets connectés en 2013 était de l'ordre de 5 milliards, ce chiffre va se multiplier par 10 d'ici 2020 pour atteindre 50 milliards d'objets connectés. En 2013, le nombre d'utilisateurs connectés à internet depuis un smartphone a dépassé celui des utilisateurs connectés via un PC.

Sur le plan d'utilisation d'internet, les derniers chiffres en 2017 affirment que le nombre d'internautes est d'environ 3.81 Milliards, soit 51% de la population, le nombre d'inscrits sur les réseaux sociaux est 2.91 milliards, soit 39% de la population. Chaque minute sur internet, il y a 216 millions de photos aimées sur Facebook, 7 millions de Snaps envoyés sur Snapchat, 2.4 millions de photos aimées sur Instagram, 400 heures de vidéos téléchargées sur YouTube, 70 millions de mots traduits sur Google Translate, 110 000 appels sur Skype, 830 000 fichiers téléchargés sur Dropbox.

En Juin 2012, Amazon a évalué la croissance du nombre de fichiers enregistrés sur son site de stockage Cloud Amazon S3 de 1000 milliards d'entités de données par rapport à 10 milliards hébergées en Octobre 2007. On peut illustrer ce phénomène de « déluge de données » par un exemple du réseau social Twitter : à l'occasion d'un événement sportif (classico espagnol), +7000 messages ont été générés par seconde et même si la taille de chaque message est faible, leur vélocité se doit d'être importante. A raison de 140 car/Tweet, la fréquence des échanges de Twitter a produit plus de 8 To/jour [4].

L'amplitude de cette volumétrie recouvre 2 modes d'usage opposés : d'un côté le nombre d'accès aux données ne fait qu'augmenter ; d'un autre côté, l'habitude humaine est de conserver les données sans savoir si elles seront relues ou non. La gestion des données évolutives et distribuées qui a été la vision de la communauté des chercheurs dans le domaine des bases de données depuis plus de trois décennies, s'est accéléré remarquablement ces dernières années pour pallier aux rudes contraintes liées aux serveurs, aux réseaux et aux applications évoluant dans des environnements distribués imposés par les nouvelles tendances informatiques. Les dernières étapes de l'évolution informatique ont émergé de nouvelles technologies : Cloud et Big Data qui bouleversent l'industrie avec l'arrivée des acteurs issus des sociétés de l'internet.

Le changement d'échelle en volumétrie, la répartition géographique des données sont déjà des problèmes d'actualité, à lesquelles s'ajoute une autre difficulté liée à la diversité des types de ces masses de données crées puis accumulées, sans organisation ni de structure. La part importante et croissante des données non structurées parmi l'ensemble des données numériques, est un des facteurs qui a conduit au nouveau concept Big Data ; les dernières études montrent que le pourcentage des données non structurées est de 90% [4]. Ainsi étendre la gestion et les traitements aux données multi structurées, à savoir : données structurées, semi-structurées ou non structurées, est considéré comme problématique fondamentale autant pour les Big Data que pour les Clouds. Pour coller à ces nouvelles tendances et remédier aux rudes contraintes imposées par cette montée en charge, de nouvelles approches de conception, gestion, organisation et stockage de ces masses d'information sont été conçues, développées et déployées ces dernières années. Ces nouvelles exigences ont été l'objectif fondateur des Clouds de stockage « Storage-as-a-Service » qui offrent le stockage en tant que service [5, 6, 7]. Sur le plan organisationnel et gestion de données, les solutions NoSQL adoptées pour les Big Data, se présentent comme des architectures extensibles et flexibles pour la prise en charge de ces masses hétérogènes de données. Ces nouveaux systèmes se considèrent comme des nouveaux modèles de stockage et d'accès aux données en remplacement des SGBD relationnels qui étaient considérés presque parfaits jusqu'à ces dernières années.

Chap I : Big Data et Cloud Computing

Ce chapitre se focalise sur les deux nouvelles technologies : Cloud Computing et BigData.

I.1 Big Data

L'évolution des architectures des différents systèmes de gestion de données qui ont amené au Big Data, ainsi que les concepts de base de cette dernière génération, vont être évoqués dans les sections suivantes.

I.1.1 Evolution des systèmes de gestion de données

Chaque innovation dans le domaine de la gestion des données se considère comme un nouveau départ isolé de ce qui a été fait auparavant. Cependant, que ce soit révolutionnaire ou graduelle, la plupart des nouvelles étapes et vagues de développement dans le domaine de gestion des données s'appuient sur leurs prédécesseurs alors on ne renverse pas les outils, la technologie et les pratiques utilisés pour traiter de nouveaux et différents problèmes.

L'évolution des systèmes de gestion des données doit inclure les évolutions technologiques du matériel, le stockage, le réseautage et les nouveaux modèles informatiques tels que la virtualisation et le Cloud Computing. L'association des technologies émergentes et la réduction des différents coûts du stockage aux traitements informatiques, ont bouleversé les réflexions et les manières de gestion et exploitation des données, et ont permis de nouvelles possibilités.

Chaque naissance d'un nouveau modèle de gestion des données est justifiée par le besoin de résoudre un type spécifique de problème de gestion des données. Chacune de ces générations ou phases a évolué en raison de la cause et l'effet. Lorsqu'une nouvelle solution technologique est déployée sur le marché, on est amené à découvrir de nouvelles approches. Lorsque les bases de données relationnelles ont envahi le marché, on était obligé de concevoir un ensemble d'outils pour permettre aux gestionnaires d'étudier les associations et les relations entre les éléments de données. Lorsque les entreprises ont commencé à stocker des données non structurées, on a pu réaliser que les différents serveurs ont été submergés par les immenses quantités de données hébergées, alors pour tirer profit de ces données, il fallait de nouveaux outils et approches novatrices et la nécessité de développement de nouvelles fonctionnalités et outils d'analyse pour obtenir des informations qui seraient utiles aux entreprises.

Les différentes générations des technologies de gestion des données au cours des dernières décennies ont avancé un nouvel épisode intitulé : l'ère du Big Data. Cette dernière tendance

n’est pas une technologie indépendante ; plutôt, c'est une accumulation des 50 dernières années d'évolution technologique.

Avant de présenter les concepts fondamentaux du Big Data, on va faire un aperçu sur l'évolution des modèles de gestion des données [8].

I.1.1.1 Vague 1 : Création de systèmes de gestion de données

Vers la fin des années 60, les données ont été stockées dans des fichiers plats qui n'imposaient aucune structure. Les utilisateurs ont dû appliquer des modèles de programmation très détaillés pour créer une certaine valeur.

Plus tard dans les années 1970, avec l’apparition des systèmes de gestion des bases de données relationnelles (RDBMS), la structure était imposée et un niveau d'abstraction a été ajouté (Langage de requête SQL, les générateurs de rapports et les outils de gestion des données), alors il était plus facile pour les programmeurs de satisfaire les demandes croissantes de gestion de données. Le modèle relationnel offrait un écosystème d'outils issus d'un grand nombre de sociétés de logiciels émergentes.

Avec l’augmentation du nombre de requêtes sollicitant les données, les performances commencent à se dégrader, alors un besoin urgent de trouver de nouvelles techniques pour soutenir le modèle relationnel. Le modèle Entité-Relation (ER) est apparu, qui a ajouté une abstraction supplémentaire pour augmenter l'utilisabilité des données. Dans ce modèle, chaque élément a été défini indépendamment de son utilisation. Par conséquent, les développeurs pourraient créer de nouvelles relations entre les sources de données sans une programmation compliquée. C'était une avance énorme à l'époque qui a permis aux développeurs de repousser les limites de la technologie et créer des modèles plus complexes pour joindre les entités de données. Le marché des bases de données relationnelles a explosé depuis le temps et restera ainsi très dynamique jusqu’aujourd'hui. Il demeure toujours le modèle adéquat pour la gestion transactionnelle des données hautement structurées.

Avec la multiplication des applications informatiques dans le monde des entreprises dans les années 90, la volumétrie des données a augmenté considérablement. Le besoin de fusion, intégration des données issues de diverses bases de données pour en extraire celles qui ont plus d’importance pour des fins d’analyse et aide à la décision, a émergé les Data Warehouses.

Les entrepôts de données et les Datacenters ont résolu de nombreux problèmes des entreprises pour gérer leurs données transactionnelles d’une manière cohérente. Mais quand il s'agissait de gérer d'énormes volumes de données non structurées ou semi-structurées, l'entrepôt ne

pouvait pas évoluer suffisamment pour répondre aux différentes demandes qui ne cessaient de changer.

Pour prendre en charge l'accroissement des volumes de données non structurées, les fournisseurs des systèmes de gestion de bases de données relationnelles ont commencé à ajouter des fonctionnalités telles que BLOB (Binary Large OBjects). Essentiellement, un élément de données non structuré serait stocké dans une base de données relationnelle comme un segment contigu de données. C'est un objet étiqueté mais on ne pouvait pas voir ce qui se trouvait à l'intérieur de cet objet. Évidemment, cela ne résoudrait pas tous les besoins variables des clients ou des entreprises.

Avec l'avènement des systèmes de gestion de base de données objet (ODBMS), une base de données pouvait stocker le BLOB comme un ensemble adressable d'éléments afin qu'on puisse voir son contenu. Contrairement au BLOB, qui était une unité indépendante ajoutée à une base de données relationnelle traditionnelle, la base de données orientée objet fournissait une approche uniforme pour traiter les données non structurées. Les bases de données d'objets incluent un langage de programmation et une structure pour les données qui permet de manipuler facilement divers objets de données sans programmation et de complexes jointures. Ces nouveaux modèles ont introduit un nouveau niveau d'innovation qui a contribué à la deuxième vague des systèmes de gestion des données [8].

I.1.1.2 Vague 2 : Web et gestion de contenu

Ce n'est pas un secret que la plupart des données disponibles dans le monde d'aujourd'hui sont non structurées. Paradoxalement, les entreprises ont concentré leurs investissements dans les systèmes avec des données structurées qui étaient très associés à leurs revenus : Line-Of-Business transactional systems (LOB). Les systèmes de gestion de contenu d'entreprise ont évolué dans les années 1980 pour donner aux entreprises la capacité de mieux gérer les données non structurées, principalement des documents. Dans les années 1990, avec l'essor du Web, les organisations voulaient aller au-delà des documents et stocker et gérer le contenu Web, images, audio et vidéo.

Le marché est passé d'un ensemble de solutions déconnectées à un modèle plus uniforme qui a regroupé ces éléments en une plateforme qui intègre la gestion des processus métiers (Business Process Management BPM), la reconnaissance d'information, la gestion des textes et la collaboration. Cette nouvelle génération de systèmes a introduit le traitement des métadonnées (informations sur l'organisation et caractéristiques des informations stockées).

Entre temps, une nouvelle génération d'exigences a commencé à émerger qui va conduire à la prochaine vague. Ces nouvelles exigences ont été motivées, en grande partie, par une convergence de facteurs tels que le Web, la virtualisation et le Cloud Computing. Dans cette nouvelle vague, les organisations admettent qu'elles doivent gérer une nouvelle génération de sources de données qui créent une quantité et une variété de données sans précédent et qui doivent être traitées à une vitesse surprenante.

I.1.1.3 Vague 3 : Gestion de Big Data

Les Big Data sont-elles vraiment nouvelles ou s'agit-il d'une évolution des systèmes de gestion des données ? La réponse est oui - c'est en fait les deux. Comme pour les autres générations, les Big Data émergent de l'évolution des pratiques de gestion des données au cours des cinq dernières décennies. Ce qui est nouveau, c'est que pour la première fois, le coût des traitements et de stockage a atteint un niveau réduit, considéré comme un point de basculement. C'est important de rappeler qu'il y a seulement quelques années, les organisations commettaient généralement des compromis en stockant des clichés ou des sous-ensembles d'informations importantes, car le coût de stockage et les limites de traitement ne leur permettait pas de stocker tout ce qu'ils voulaient analyser.

Avec Big Data, il est désormais possible de virtualiser les données afin qu'elles puissent être stockées efficacement, et en utilisant un stockage en nuage, plus économique. En outre, les améliorations apportées à la vitesse et la fiabilité du réseau ont dégagé d'autres limitations physiques pour pouvoir gérer des quantités massives de données à une vélocité acceptable, ajoutez à cela l'impact des changements économiques et technologiques de la mémoire.

Mais aucune transition technologique ne se produit isolément ; Beaucoup de technologies au cœur des Big Data, telles que la virtualisation, le traitement parallèle et les systèmes de fichiers distribués, existent depuis des décennies. D'autres technologies telles que Hadoop et MapReduce ont jailli sur la scène seulement depuis quelques années. Cette combinaison d'évolutions technologiques vient pour résoudre des problèmes commerciaux importants. Toutes ces technologies nécessaires pour obtenir les réponses aux différents besoins de l'entreprise sont relativement isolées les unes des autres. Pour aboutir à l'état final souhaité, les évolutions de différentes technologies devront se rejoindre.

Les entreprises veulent être en mesure d'obtenir des informations pertinentes et des résultats utiles à partir de nombreux types de données à une vitesse convenable - peu importe la quantité de données impliquée.

Le passage aux Big Data ne concerne pas seulement les entreprises ; la science, la recherche et les activités gouvernementales ont également contribué à cette avancée.

I.1.2 Définition

Littéralement, Big Data signifie données massives ou méga données. C'est un ensemble d'entités de données hétérogènes en extensibilité permanente qui ne peuvent pas pris en charge par les systèmes de gestion de données classiques. Big Data est aussi une architecture distribuée et scalable pour le traitement et le stockage de grands volumes de données.

En effet, on crée environ 2,5 milliards de Giga octets de données tous les jours, émanant des différents domaines créés par les divers outils numériques : vidéos publiés, messages envoyés, signaux GPS, enregistrements transactionnels d'achats en ligne et bien d'autres encore. Ces volumes massifs de données sont baptisés Big Data. Les géants du Web, au premier rang comme Yahoo, Facebook, Amazon et Google, ont été les tous premiers à déployer ce type de technologie pour permettre à tout le monde d'accéder en temps réel à leurs bases de données géantes.

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

Cependant, aucune définition universelle ou précise ne peut qualifier le Big Data. Etant un concept polymorphe et complexe, son interprétation varie selon les communautés qui s'y intéressent en tant que fournisseur ou utilisateur de services. Le Big Data est aussi défini par rapport à la manière avec laquelle les grandes masses de données peuvent être traitées et exploitées de façon optimale.
L'émergence du Big Data est considérée comme une nouvelle révolution industrielle semblable à la découverte de la vapeur, de l'électricité, du téléphone et de l'informatique. D'autres, qualifient ce phénomène comme étant le dernier épisode de la troisième révolution industrielle, dite celle de « l'information ».
Depuis l'apparition de cette terminologie, beaucoup de bruit a été rouspété autour du Big Data, mais selon le Gartner, ce concept regroupe une famille d'outils qui répondent à une triple problématique dite règle des 3V.

Il s'agit notamment d'un **V**olume d'informations considérable à traiter, une grande **V**ariété de données (venant de diverses sources, non-structurées, organisées, Open…), et un certain

niveau de **V**élocité à atteindre, autrement dit vitesse des échanges ou fréquence de création, collecte et partage de ces données [8].

Une autre définition similaire dans [8] ou les auteurs qualifient les Big Data comme n'importe quel type de source de données qui a au moins trois caractéristiques communes :

- **V**olumes de données extrêmement volumineux ;
- **V**itesse de transmission extrêmement élevée ;
- Très grande **V**ariété de données.

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

I.1.2.1 Volume

Le Big Data est associé à un volume de données vertigineux, se situant actuellement entre quelques dizaines de téraoctets (1 To=2^{12} octets) et plusieurs pétaoctets (1 Po=2^{15} octets) en un seul jeu de données. Le volume correspond à la masse d'informations produite chaque seconde. Les entreprises issues de tous les secteurs d'activité gérant des données massives, se voient assujetties à trouver des techniques nécessaires et moyens capables pour gérer les volumes de données collectés chaque jour et d'une importance vitale pour leur survie.

I.1.2.2 Vélocité

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

I.1.2.3 Variété

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

classement des données de différents types comme des photos sur différents sites ou les messages échangés sur les réseaux sociaux, etc. Ce sont les différents éléments qui constituent la variété supportée par le Big Data.
Alex Popescu [9] parle aussi d'un autre V de **V**ariabilité qui l'a défini par le format et le sens des données qui peut varier au fil du temps.

I.1.2.4 Valeur

Toutes ces données massives ressemblent à une mine d'or potentielle, mais comme dans une mine d'or, vous avez seulement peu d'or et beaucoup plus de tout le reste. Parmi les défis les très sensibles de cette technologie est comment donner du sens à ces données pour tirer celles qui sont les plus pertinentes pour d'éventuelles prises de décisions dans une entreprise ? Comment une organisation traite-t-elle des quantités massives de manière significative ?

La notion de Valeur correspond à l'intérêt qu'on puisse tirer de l'utilisation de cette technologie. Selon les experts du domaine, les entreprises qui ne s'intéressent pas sérieusement au contenu de leurs volumes de données hébergées risquent d'être pénalisées et dépassées. Big Data désigne à la fois les grands volumes de données et la difficulté à extraire de cette masse de données celles ayant suffisamment de valeur pour justifier leur analyse. Big Data offre un ensemble d'outils d'analyse de données qui peuvent servir à préserver un privilège concurrentiel.

I.1.2.5 Véracité

L'aptitude à juger la crédibilité et la fiabilité du nombre indéfini de données collectées qualifie la Véracité du Big Data. Il est difficile de justifier l'authenticité et l'exactitude des contenus des différents volumes et variétés de données manipulées comme dans les conversations dans les réseaux sociaux avec les abréviations, le langage familier, les coquilles, les hashtags.

La vérité est que le Big Data, a bien trois significations et que les éditeurs n'en abordent qu'une à la fois. Il est important de connaître leur positionnement pour leur poser les bonnes questions.

I.1.3 Architecture Big Data

On distingue principalement les couches suivantes [8] :

- **Couche matériel (infrastructure Layer)**: Peut employer des serveurs virtuels VMware, ou des serveurs physiques ;

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

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

I.1.3.1 Infrastructure physique redondante

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

I.1.3.2 Sécurité d'infrastructure

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

I.1.3.3 Avantages de l'architecture Big Data

Plusieurs avantages peuvent être associés à une architecture Big Data [8], citons par exemple :

- **Extensibilité (scalabilité)** : Le concept Big Data apporte une architecture scalable qui peut prévenir la taille d'infrastructure et l'espace disque nécessaire.
- **Performance** : Grâce au traitement parallèle des données et à son système de fichiers distribué, le concept Big Data est hautement performant en diminuant la latence des requêtes.

- **Coût faible** : On n'aura plus besoin de centraliser les données dans des baies de stockage souvent excessivement cher, grâce à unsystème de fichiers distribués, les disques internes des serveurs suffiront.
- **Disponibilité** : On a plus besoin des RAID disques, souvent coûteux. L'architecture Big Data apporte ses propres mécanismes de haute disponibilité.

I.1.4 Sources et types de données

Les données collectées, stockées et traitées dans les Big Data peuvent être issues de différents domaines et crées par plusieurs sources de données hétérogènes, ce qui génère une masse de données de types différents structurés et non structurés [8].

I.1.4.1 Sources de données structurées

Le terme « données structurées » désigne généralement des données dont la longueur et le format sont définis. Les exemples de données structurées comprennent des nombres, des dates et des chaînes (par exemple, le nom d'un employé, son poste de travail, etc.). On rappelle que la plupart des études déclarent que ce type de données représente environ de 10 à 20 pour cent des données globales. Les données structurées sont les données qu'on a l'habitude de traiter, généralement stockées dans une base de données relationnelle ayant un schéma, interrogées à l'aide du langage de requête structuré (SQL). Elles sont recueillies à partir de sources traditionnelles.

Dans le monde des Big Data, les données structurées prennent un nouveau rôle avec l'évolution de la technologie qui offre de nouvelles sources de données structurées produites souvent en temps réel et en grands volumes. Les sources de données sont divisées en deux catégories :

- **Générées par ordinateur ou par machine** : Les données générées par machine se réfèrent généralement à des données créées par une machine sans intervention humaine.
- **Générées par l'homme :** Ce sont les données créées par les humains, en interaction avec les ordinateurs.

Dans la première catégorie, plusieurs sources existent actuellement :

- **Données de capteurs** : Données GPS, Radiofréquence, compteurs intelligents, dispositifs médicaux, journaux des appels en téléphonie, etc.

- **Données Web** : Le fonctionnement des serveurs, des applications et des réseaux permet de capturer toutes sortes de données comme les textes, les images, les vidéos, et tous ce qui peut figurer sur les pages Web.
- **Données commerciales** : Lorsque le caissier glisse le code-barres d'un produit acheté, toutes les données associées au produit sont générées. Ilsuffit de penser à tous les produits acquis par toutes les personnes, pour imaginer le volume de sonnées crée.
- **Données financières** : Beaucoup de systèmes financiers sont automatisés ; Ils sont exploités sur la base de règles prédéfinies qui automatisent les processus.

Voici quelques exemples de données structurées générées par l'homme :

- **Données de saisie** : Il s'agit de toute donnée, qu'un être humain peut introduire dans un ordinateur, comme le nom, l'âge, le salaire, les sondages et ainsi de suite. Ces données peuvent être analysées pour comprendre le comportement des clients.
- **Données de flux de clics** : les données sont générées chaque fois qu'on clique sur un lien sur un site Web. Ces données peuvent être analysées pour déterminer le comportement des clients et les modèles d'achat.
- **Données liées aux jeux** : Chaque mouvement fait dans un jeu peut être enregistré. Cela peut être utile pour comprendre le comportement des utilisateurs.

I.1.4.2 Sources de données non structurées

Les données non structurées sont des données qui n'ont pas un format spécifié. Elles représentent 80 à 90 pour cent de l'ensemble des données disponibles. Jusqu'à quelques années auparavant, les outils disponibles n'offraient pas des traitements particuliers à ce type de données, mise à part le stockage ou l'analyse manuelle. Des données non structurées sont un peu partout : *pdf*, *doc*, email ou post dans un réseau social. En fait, un grand nombre d'individus et organisations mènent leurs activités, leurs revenus autour de ces types de données.

Tout comme pour les données structurées, les données non structurées sont générées soit par la machine ou par l'humain. Voici quelques exemples de données non structurées générées par une machine :

- **Images satellites** : Incluent les données météorologiques ou les données images de surveillance par satellite capturées par les services gouvernementaux. Un exemple typique de ces systèmes est Google Earth.

- **Données scientifiques** : Cela comprend l'imagerie sismique, les données atmosphériques, les données astronomiques, l'environnement, la génomique, la physique subatomique et la physique des hautes énergies.
- **Photographies et vidéo** : Concerne la sécurité, la surveillance et la vidéo de trafic.

Voici d'autres exemples de données non structurées générées par l'homme :

- **Textes et courrier interne d'une entreprise** : Inclue tous les textes contenus dans les documents, les journaux, les rapports, les procés verbaux, les bilans, les résultats d'enquêtes, les sondages et les courriers. L'information d'entreprise représente en fait un grand pourcentage des données textuelles dans le monde d'aujourd'hui.
- **Données des médias sociaux** : Ces données sont générées à partir des plateformes des réseaux sociaux tels que YouTube, Facebook, Twitter, LinkedIn et Flickr.
- **Données mobiles** : Cela inclut des données telles que les messages texte et les informations de localisation.
- **Contenu du site Web** : Ceci provient de n'importe quel site offrant des contenus non structurés, tels que YouTube, Flickr ou Instagram [8,10].

Notons à la fin de cette section qu'il y a une autre catégorie de données qualifiée comme semi-structurée qui se place entre les deux catégories structurée et non structurée. Les données semi-structurées ne sont pas nécessairement conformes à une structure fixe prédéfinie mais peuvent être auto-descriptives et définies par des simples couples marque/valeur. Par exemple, ces couples peuvent inclure :<Famille> = Matallah, <Père> = Abdelkader, et <fils> = Oussama. Des exemples de données semi-structurées incluent XML (Extensible Markup Language), CSV file (Comma-Separated Values), EDI (Electronic Data Interchange) et SWIFT (Langage de script implicitement parallèle).

I.1.5 Processus de collecte et chargement de données

Une couche responsable de chargement de données dans une architecture Big Data, est capable de gérer d'énormes volumes de données hétérogènes ou on doit valider, épurer, transformer, réduire (compression) et intégrer les données dans la grande pile de données en vue de leur traitement. Elle collecte les informations pertinentes et les charge dans la couche de stockage Big Data [11].

La figure A.1 illustre le processus et les composants présents dans la couche de chargement de données.

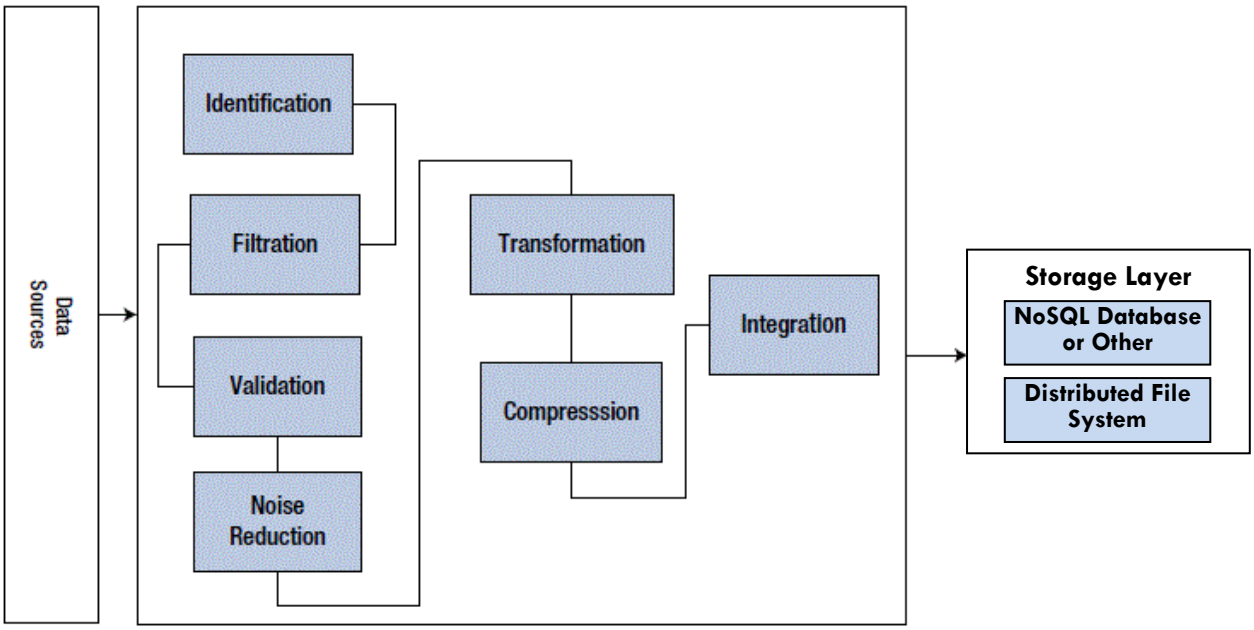


Figure A.1 : Processus de chargement de données Big Data [11]

Cette couche doit inclure les composants suivants :

- **L'identification** des différents formats de données.
- **La filtration et sélection** de l'information entrante pertinente.
- **La validation** et analyse des données en permanence.
- **La réduction** du bruit qui consiste au nettoyage des données.
- **La transformation** peut entraîner le découpage, la convergence, la normalisation ou la synthèse des données.
- **La compression** consiste à réduire la taille des données sans perte de pertinence.
- **L'intégration** consiste à intégrer l'ensemble des données collectées dans la couche de stockage Big Data (Généralement NoSQL DataBase).

I.1.6 Principaux acteurs

Les acteurs sont les organismes qui ont conçu, développé, déployé et utilisé du Big Data, ce sont principalement les fondateurs de ce nouveau concept et les « créateurs » des technologies Big Data.

Les principales innovations du domaine trouvent leur origine chez les leaders du Web [10] : Google (MapReduce et BigTable), Amazon (Dynamo, S3), Yahoo! (Hadoop, S4), Facebook (Cassandra, Hive), Twitter (Storm, FlockDB), LinkedIn (Kafka, SenseiDB, Voldemort), LiveJournal (Memcached), etc.

La fondation Apache est particulièrement active dans ce domaine, en lançant ou en recueillant plus d'une dizaine de projets, matures ou en incubation : Hadoop, Lucene/Solr, Hbase, Hive, Pig, Cassandra, Mahout, Zookeeper, S4, Storm, Kafka, Flume, Hama, Giraph, etc.

Outre les sociétés du Web, le secteur scientifique et plus récemment les promoteurs de l'Open Data et de sa variante, l'Open Linked Data, issu du Web Sémantique, ont également effectué des contributions importantes dans le domaine du Big Data [10].

I.1.7 Big Data et Informatique distribuée

L'informatique distribuée n'est pas un nouveau concept technologique mais il existe depuis près de 50 ans. Initialement, la technologie a été utilisée dans la recherche en informatique dans le but d'augmenter les performances informatiques pour résoudre des problèmes assez complexes. En revanche, s'il n'y a pas une importante contrainte de temps, un traitement complexe peut être effectué à distance par un service spécialisé qui n'est pas forcément distribué.

Il n'existe pas un seul modèle informatique distribué parce que les ressources informatiques peuvent être distribuées de différentes manières. Par exemple, vous pouvez distribuer un ensemble de programmes sur le même serveur physique en utilisant les services de messagerie pour leur permettre de communiquer et de transmettre des informations. Il est également possible d'avoir de différents systèmes ou serveurs, qui peuvent être utilisés ensemble pour résoudre un problème particulier.

Les percées majeures en matière de matériel et de logiciels ont révolutionné le marché de gestion des données. Ces innovations ont pu augmenter les performances et diminué les prix du matériel. De nouveaux logiciels ont émergé, qui ont bien exploité les évolutions technologiques sur l'aspect matériel, en automatisant des processus comme l'équilibrage de charge, le traitement parallèle et l'optimisation à base d'un cluster composé de nombreux nœuds. En cas d'échec d'un nœud, les traitements sont déplacés vers un autre nœud pour assurer un service sans interruption, en utilisant la technologie de la virtualisation.

Les nouvelles technologies Big Data se sont appuyées sur des anciennes technologies de l'informatique distribuée employées depuis plusieurs années [8].

I.1.8 Big Data et Data Warehouse

En informatique décisionnelle traditionnelle (aussi nommée DSS pour Decision Support System ou encore BI pour Business Intelligence), toutes les données de l'entreprise sont regroupées dans un serveur central de l'entrepôt de données.

Il s'agit majoritairement de données internes à l'entreprise, mais réparties, puisqu'elles sont stockées dans plusieurs bases de données opérationnelles transactionnelles qui manipulent en grande partie des données structurées souvent des années antérieures.

Les données dans la BI traditionnelle sont généralement analysées en mode déconnecté. En outre, les entrepôts de données sont généralement alimentés en intervalles discontinus, généralement hebdomadaires ou quotidiens. C'est parfait pour la planification, les rapports financiers et les campagnes de marketing traditionnelles, mais c'est trop lent pour les environnements d'un nombre très important de clients exigeant des réponses de plus en plus en temps réel comme pour les moteurs de recherche ou les sites de commerce électronique.

Une solution Big Data, peut être distinguée d'une BI traditionnelle sur les aspects suivants :

- Les données sont conservées dans un système de fichiers distribué et scalable plutôt que sur un serveur central dans lequel sont consolidées les données de l'entrepôt.
- Les types et les formats de données manipulées sont très variés, ce n'est pas le cas dans les entrepôts qui manipulent des données structurées.
- Les données Big Data sont analysées en ligne, par contre les données dans la BI traditionnelle sont généralement analysées en mode déconnecté.
- Les données sont analysées en temps réel contrairement aux entrepôts qui sont alimentés en intervalles discontinus, hebdomadaires ou quotidiens.
- La technologie Big Data emploie un traitement massivement parallèle, les Data Warehouse s'articulent sur des architectures Client/Serveur.

Avec l'avènement du Big Data, les concepteurs des entrepôts de données sont en train de réfléchir à une approche complémentaire avec le Big Data ou on pourrait concevoir un modèle hybride. Dans ce modèle hybride, les données opérationnelles très structurés seront stockées et analysées dans l'entrepôt de données, tandis que les données qui sont fortement distribuées et non structurées seront gérées par une solution Big Data [8].

I.1.9 Stockage de données et virtualisation

Les difficultés de gestion et stockage de données dans les entrepôts de données en raison de leurs volumes, types et formes contrarient leur exploitation adéquate. Les nouvelles architectures développées pour répondre favorablement à ces nouveaux besoins ont émergé les bases de données NoSQL.

Les bases NoSQL visent pour passer à l'échelle de manière horizontale en relâchant partiellement ou totalement les conditions fortes de transactionnalité (ACID) jugées comme

fondamentales pour les systèmes traditionnels, et en renonçant au modèle relationnel (Voir Chapitre II de la partie A).

Pour mieux contrôler l'utilisation et la performance des ressources informatiques et en particulier le stockage, on applique de la virtualisation. C'est une technologie fondamentale mise en œuvre à la fois dans les Big Data et le Cloud Computing. La virtualisation des données peut être utilisée pour créer une plate-forme pour les services de données liés dynamiquement. Cela permet de rechercher facilement les données et de les relier par une source de référence unifiée. Par conséquent, la virtualisation des données fournit un service abstrait qui présente des données sous une forme cohérente quelle que soit la base de données physique sous-jacente. De plus, la virtualisation des données expose les données mises en cache à toutes les applications pour améliorer les performances. La virtualisation de stockage combine les ressources de stockage physique afin de les partager plus efficacement, cela réduit le coût du stockage et facilite la gestion de données requises pour l'analyse. La virtualisation des données et du stockage joue un rôle important en facilitant et en réduisant le coût de stockage, de recherche et d'analyse des données, structurées et surtout non structurées, hébergées dans les environnements Big Data. Avec la virtualisation, la base de données peut être stockée en tant qu'image virtuelle et appelée chaque fois qu'elle est nécessaire sans consommer davantage des ressources de Data Center [8].

I.2 Cloud Computing

Depuis sa création, la technologie de l'internet se développe d'une manière exponentielle, et en particulier les pages web qui sont de l'ordre de cinq milliards pour plus d'un milliard de sites en 2016. Actuellement, une nouvelle « tendance » est entrain de dominer le marché informatique, il s'agit du Cloud Computing. Le Cloud s'est démocratisé à partir du début du nouveau millénaire, même s'il est assez récent, son histoire est directement liée au développement de l'internet. Cette technologie, permet aux utilisateurs de réduire leurs coûts d'exploitation des logiciels et du matériel directement en ligne, de gagner en performance et en extensibilité. Le Cloud Computing évolue actuellement dans un contexte très favorable dans le monde professionnel, puisque la majorité des grandes entreprises utilisent ou envisagent d'utiliser des services Cloud au cours des prochaines années.

I.2.1 Définition

L'organisme NIST (National Institute of Standards and Technology) définit le Cloud Computing comme un modèle fournissant, à la demande et au travers d'un réseau, un

ensemble partagé de ressources informatiques incluant desserveurs, des espaces de stockage, des applications, des traitements et des plates-formes de déploiement qui peuvent être rapidement mises en service avec un effort minimum de gestion et d'interaction avec le fournisseur de ce service.

Le Cloud Computing permet de rendre une infrastructure matérielle et logicielle dynamique et flexible en exposant les capacités des Data-Centers comme étant un "réseau de services virtuels". Dans cette infrastructure, les utilisateurs peuvent accéder et déployer des applications à partir d'Internet suivant leurs demandes et la qualité de service exigée [12].

Le Cloud représente l'avenir des infrastructures software et hardware. Cette tendance est justifiée par les coûts réduits de plus en plus (Service mesurable et facturable : Ordinateur loué dans le Cloud coûte moins de 5€/mois), les offres attractives, et les débits des connexions modernisés. Actuellement un nombre très important d'entreprises ont déjà migré certaines de leurs services vers le Cloud ou envisagent de le faire dans le proche futur. Les offres des services de type Cloud ne cessent de s'accroitre à tous les niveaux, à partir des applications spécifiques, embarqués, CRM, ERP, services web, librairies, jusqu'aux infrastructures complètes. Le marché du Cloud Computing devrait dépasser 241 milliards de dollars en 2020.

Le Cloud Computing possède trois modèles de service, quatre modèles de déploiement et, présentés brièvement dans la section suivante.

I.2.2 Modèles de services

Le Cloud se décline sous la forme de 3 offres : SaaS, PaaS et IaaS (Figure A.2) [12]

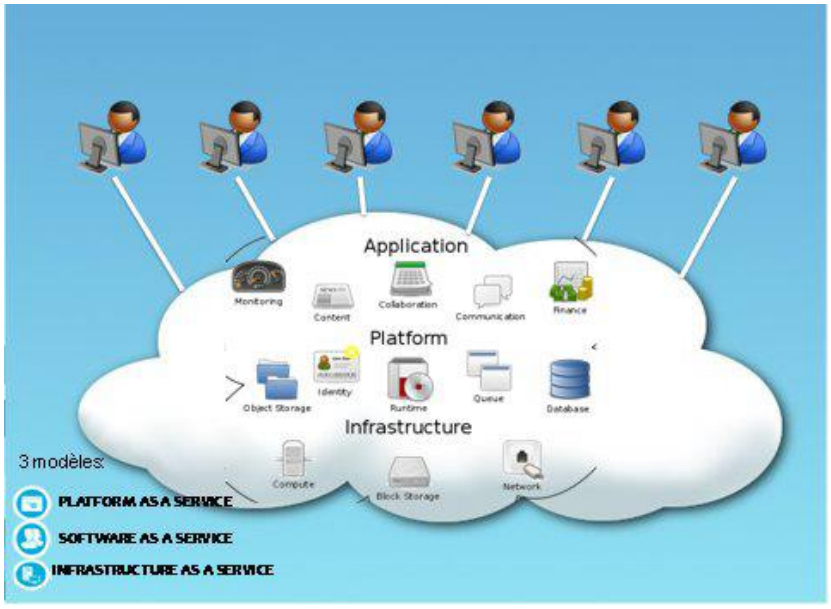


Figure A.2 : Les 3 modèles du Cloud Computing

I.2.2.1 SaaS (Software as a Service)

Dans ce modèle, le logiciel est offert sous la forme d'un service. Le fournisseur de Cloud de type SaaS gère entièrement sa plateforme matérielle et logicielle, et les clients du Cloud utilisent ainsi le logiciel fourni sans s'occuper de la pile en dessous (plateforme applicative, matériel, …) ni de l'installation du logiciel en question.

Exemples de SaaS : La messagerie électronique, logiciels de type CRM, etc.

I.2.2.2 PaaS (Plateforme as a Service)

Dans ce second modèle, le fournisseur offre une plateforme sous forme d'environnement complet sur laquelle des développeurs ou éditeurs de logiciels des clients peuvent déployer des applications. La pile en dessous de cette plateforme à savoir le socle applicatif, le système d'exploitation, le matériel et le réseau, sont gérés par le fournisseur de service. Notons que certaines offres PaaS exigent un langage de programmation spécifique.

Exemple de PaaS : Google AppEngine (PaaS où les développeurs peuvent réaliser leurs programmes en Python ou Java), Engine Yard (avec Ruby on Rails), force.com deSalesforce.com (langage propriétaire) et Coghead SAP (langage propriétaire).

I.2.2.3 IaaS (Infrastructure as a Service)

Infrastructure as a Service (IaaS) est la proposition d'un ensemble de services informatiques comprenant le matériel, le réseautage et le stockage. Le fournisseur offre une plateforme sur laquelle les clients vont pouvoir déployer de ressources d'infrastructures dont la plus grande partie est localisée à distance dans des Data Centers. Le client acquiert une ressource et est facturé pour cette ressource en fonction de la quantité utilisée et de la durée d'utilisation. On trouve des versions publiques et privées de IaaS. Dans le modèle IaaS publique, l'utilisateur utilise une carte de crédit pour acquérir ces ressources, dès que l'utilisateur cesse de payer, la ressource disparaît. Dans un service IaaS privé, c'est généralement la structure informatique qui crée l'infrastructure conçue pour fournir des ressources à la demande pour les utilisateurs internes et parfois les partenaires commerciaux.

Ce modèle considéré comme une évolution des Data Centers virtualisés, permet au client de faire abstraction du modèle physique (gestion des serveurs physiques, l'électricité, la climatisation, la sécurité physique des centres de données). Dans ce modèle, le fournisseur contrôle le matériel et la couche de virtualisation. Sur l'aspect de données, le contrôle est effectué au niveau de la machine virtuelle qui est stockée et sauvegardée par le fournisseur de Cloud IaaS.

Exemples d'IaaS : Amazon EC2 (Web Services Elastic Compute Cloud) et Amazon S3(Secure Storage Service), CloudSigma, RackSpace (Cloud Server et CloudFiles).

Enfin, on récapitule le partage de responsabilité entre le client (l'entreprise) et le fournisseur Cloud pour les 3 modèles dans la figure A.3 :

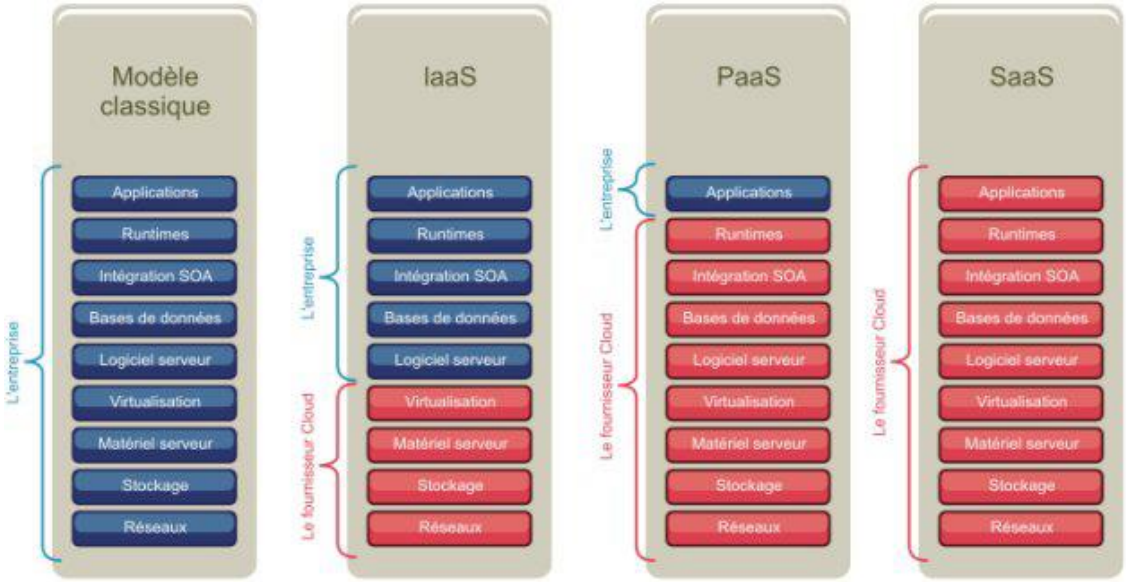


Figure A.3 : Partage de responsabilité dans les 3 modèles du Cloud [12]

Mise à part les 3 offres décrites précédemment, un autre service appelé *Data as a Service* ou les données sont fournies sur demande à l'entreprise. (DaaS) est proposé pour la gestion et traitement de données qui est étroitement lié au SaaS. DaaS est un service indépendant de la plate-forme qui permet de se connecter au Cloud pour stocker et récupérer les données. Par exemple, Google offre un service qui peut traiter une requête avec 5 téraoctets de données en seulement 15 secondes. Ce type de requête prendrait dix fois plus de temps dans un Data center ordinaire. Des centaines de services analytiques spécialisés ont été développés par des sociétés comme IBM et d'autres.

I.2.3 Modèles de déploiement

Nous distinguons deux formes principales de déploiements du Cloud Computing suivant le réseau dans lequel les services sont disponibles : Cloud public et Cloud privé, ainsi deux autres formes dérivées, à savoir Cloud communautaire et hybride [13].

I.2.3.1 Cloud Public

Le Cloud publique est un ensemble de matériel, réseautage, stockage, services, applications et interfaces dont un tiers est propriétaire et exploité par d'autres sociétés et individus. Ces fournisseurs externes possèdent des Data Center hautement extensibles masquant les détails

de l'infrastructure au client. Les services offerts sont accessibles via internet à tout le monde qui paye les services. Le principe est d'héberger des applications, en général des applications Web, sur un environnement partagé avec un nombre illimité d'utilisateurs. Les Clouds publiques sont viables parce qu'ils gèrent généralement des charges de travail relativement simples et répétitives comme par exemple, le courrier électronique. En outre, les entreprises peuvent choisir de stocker leurs des données où le coût par gigaoctet est relativement peu coûteux par rapport au stockage acheté. Ce modèle offre un maximum de flexibilité pour le client et requiert de lourds investissements pour le fournisseur de services. Néanmoins, les contraintes principales de ces plateformes sont les exigences de sécurité et un niveau de latence acceptable.

Les services offerts par les Cloud publiques ne sont pas identiques. Certains sont très extensibles avec un niveau de sécurité élevé et une meilleure qualité de services. D'autres sont moins robustes et moins sûrs, mais ils sont beaucoup moins chers à utiliser. Le choix dépendra de la nature et l'importance des données et le niveau du risque qu'on peut assumer.

Les fournisseurs du Cloud publique les plus connus sont Google, Microsoft et Amazon.

I.2.3.2 Cloud Privé

Le Cloud privé est un ensemble de matériel, réseautage, stockage, services, applications et d'interfaces appartenant à une organisation et exploité par lui pour l'usage de ses employés, partenaires et clients. Il peut être créé et géré par un tiers pour l'usage exclusif d'une entreprise. C'est un environnement déployé au sein d'une entreprise ou organisation qui en est la propriétaire, en utilisant ses infrastructures internes. Les ressources sont détenues et contrôlées par le département informatique de l'entreprise ou les services sont accessibles via le réseau privé. Le Cloud privé est un environnement hautement contrôlé se trouvant derrière un pare-feu qui n'est pas ouvert à la consommation publique. Si une organisation gère un projet Big Data qui exige un traitement de quantités massives de données, le Cloud privé pourrait être le meilleur choix en termes de latence et de sécurité. Ce modèle correspond aujourd'hui à une évolution des Data Centers virtualisés et à l'émergence de l'IT as a Service : le système d'information et les équipements informatiques qui se transforment en centre de services pour le reste de l'entreprise. Dans cette optique d'IT as a Service, on comprend parfaitement le principe du Cloud Computing en tant que service mesurable et facturable aux différentes divisions de l'entreprise.

Eucalyptus, OpenNebula et OpenStack sont des exemples de solutions proposées pour la mise en place d'un Cloud privé.

I.2.3.3 Cloud Communautaire

Le Cloud Communautaire est une extension du Cloud privé. Dans ce modèle, les ressources, services et la propriété sont partagés à l'échelle d'une communauté (ex. : à l'échelle d'un Etat, d'une ville, d'une académie, etc.). Les services sont accessibles via l'interconnexion de réseaux appartenant aux organisations de cette communauté.

I.2.3.4 Cloud Hybride

Un Cloud hybride est une combinaison d'un Cloud privé combiné à l'utilisation de services de Cloud public. Certains services sont accessibles via un réseau privé et d'autres via Internet. Le futur devrait confirmer l'émergence du Cloud hybride qui consiste à la cohabitation et la communication entre un Cloud privé et un Cloud publique dans une organisation partageant des données et des applications.

I.2.4 Secrets du succès du Cloud

Le Cloud représente la solution idéale permettant d'allier les bénéfices de l'internet et les technologies informatiques. On peut résumer le succès du Cloud dans les points suivants :

- Aucun investissement initial (Sans infrastructure propriétaire)
- Facturer à l'usage (Pay As You Go)
- Simplicité d'accès et d'utilisation de services complexes
- Réduction des coûts et du sous exploitation du matériel, licences, par consolidation et facturation réelle
- Puissance de calcul élastique
- Transfert de risques vers le fournisseur

I.2.5 Caractéristiques et avantages

On peut caractériser un Cloud sur plusieurs aspects, dont voici les plus importants [12] :

- **Accès réseau universel via le réseau**

Un environnement de type Cloud s'appuie obligatoirement sur Internet et est accessible via ce réseau, quel que soit le périphérique utilisé (PC, Mac, TV, Tablette, et Smartphone).

- **Mise en commun de ressources (Pooling)**

Dans un environnement de type Cloud, on ne pense pas en nombre de serveurs, taille de disques ou nombre de processeurs, mais surtout en puissance de calcul, capacité de stockage et bande passante disponible.

– **Elasticité**

Dans un environnement Cloud, on peut multiplier les ressources très rapidement pour soutenir une forte demande (ex : pour satisfaire le nombre important des demandes d'achats clients sur une plateforme Web de E-commerce durant les fêtes). Inversement, au-delà des ressources allouées, il est possible de diminuer ces ressources utilisées si elles sont supérieures à ce qui est réellement nécessaire (ex : en cas de baisse d'activité sur cette même plateforme Web de E-commerce dans d'autres saisons ou périodes).

– **Libre-Service (self-service)**

Dans un Cloud, il est possible à un utilisateur de consommer les services ou ressources sans pour autant devoir faire une demande d'intervention auprès de son fournisseur (équipe IT). (ex: un développeur qui souhaite tester son application sur une machine virtuelle représentative d'un poste standardisé de son entreprise peut, seul et au travers d'un portail Web, provisionner et utiliser une machine sans devoir solliciter l'équipe IT).

– **Service mesurable et facturable (Pay As You Go)**

Dans un environnement Cloud, le fournisseur de la solution est capable de mesurer de façon précise la consommation des différentes ressources (CPU, stockage, bande passante, etc.). Cette mesure lui permet ensuite de facturer le client selon l'usage. Pay As You Go est une option de facturation typique pour un fournisseur de Cloud Computing, ce qui signifie que le client est facturé en fonction des ressources consommées. Cela peut être très utile dans le cas où ce dernier n'est pas certain des ressources dont il a besoin.

– **Tolérance aux pannes**

Les architectures Cloud doivent être tolérantes aux pannes, fournissant des services continus malgré la défaillance d'un ou plusieurs composants du système.

– **Lieu de travail dynamique (Dynamic Workplace)**

Le Cloud favorise la nouvelle vision de l'entreprise sur le lieu de travail de l'employé et sa présence physique sur les sites de l'entreprise. Alors, on n'est pas obligé de travailler dans les locaux de l'entreprise mais on peut travailler n'importe où ? Le plus important est d'être rentable et efficace.

Le Cloud est caractérisé aussi par :

– La flexibilité et l'accès aux ressources de grandes échelles ;
– L'évolutivité ;
– L'agilité et la rapidité de mise en service, développement des produits, … ;
– Les données qui sont distribuées et répliquées sur de grandes distances géographiques.

I.2.6 Inconvénients majeurs

Pour une entreprise, l'emploi d'un Cloud quel que soit son modèle de service ou de déploiement, offre une panoplie d'avantages, cependant, des inconvénients peuvent en résulter particulièrement pour le Cloud non privé, dont les plus inquiétantes sont :

- Le problème de fiabilité de la bande passante vers le fournisseur.
- Le problème de sécurité et confidentialité.

I.2.7 Accord de niveau de service (SLA)

« Accord de niveau de service » ou « Service-Level Agreement » est un contrat par lequel un prestataire informatique s'engage à fournir un ensemble de services à un ou plusieurs clients. Autrement dit, il s'agit d'une clause contractuelle qui définit les objectifs précis et la qualité du service à attendre de la part du prestataire signataire.

Le SLA est intimement lié à l'univers du Cloud. Il permet de garantir aux clients certains niveaux de sécurité dans le stockage et la gestion de leurs données à caractère personnel. Il faut alors définir de façon très précise différents indicateurs de qualité pouvant être mesurés, analysés et contrôlés régulièrement. Il convient enfin de prévoir des sanctions qui seront appliquées si le prestataire ne répond pas à ses obligations mentionnées dans le SLA.

Les SLA calculent les performances en spécifiant les éléments de mesure ou indicateurs suivants :

- La disponibilité (Pourcentage de temps durant lequel les services sont disponibles) ;
- La fiabilité (Définit par exemple le nombre d'échecs par semaine) ;
- Le nombre d'utilisateurs pris en charge simultanément ;
- Le temps de réponse des applications ;
- La pérennité des Données ;
- Le périmètre de responsabilité ;
- Le dédommagement ;
- Un calendrier des notifications préalables à des modifications du réseau susceptibles d'affecter les utilisateurs ;
- Le délai de réponse du service d'assistance pour différentes catégories de problèmes ;
- Des statistiques d'utilisation.

I.2.8 Solutions Cloud existantes

Il y a actuellement beaucoup d'offres de solutions Cloud (SaaS, PaaS, IaaS) existantes sur le marché. Certaines sont propriétaires et donc payantes, et d'autres sont open source.

I.2.8.1 Solutions propriétaires

Voici quelques solutions propriétaires utilisées :

- Amazon Web Services AWS, et Elastic Compute Cloud EC2 (IaaS) depuis 2006.
- Google App Engine (PaaS), et Google App (SaaS) depuis 2008.
- Microsoft Azure et Office 365 de Microsoft depuis 2010.
- Salesforce (SaaS) de Salesforce.com depuis 1999.
- ES Citrix.
- VMwareCloud de VMware.

Dans la section suivante, une brève description de trois solutions développées et fournies par des poids lourds du marché Cloud, est présentée [8].

- **Amazon's Public Elastic Compute Cloud**

Actuellement, l'un des prestataires de services IaaS les plus prestigieux est Amazon Web Services avec son Elastic Compute Cloud (Amazon EC2). Initialement, Amazon n'avait pas une vision de construire une énorme infrastructure pour la mettre à la disposition de clients sous forme de services, mais c'était plutôt une infrastructure massive pour fonctionner son propre commerce de détail, sauf que l'entreprise a découvert que ses ressources étaient sous-exploitées. Au lieu de laisser cet actif s'abolir au ralenti, ils ont décidé de tirer parti de ces ressources en louant son infrastructure à d'autres entreprises ou utilisateurs. Le service EC2 d'Amazon a été lancé en 2006 et ne cesse d'évoluer depuis sa création.

Amazon EC2 offre une scalabilité contrôlée par l'utilisateur, qui paye les ressources utilisées à l'heure (à partir de quelques centimes € / heure). L'utilisation du terme élastique dans la désignation d'Amazon EC2 est significative, puisque les utilisateurs EC2 ont à augmenter ou à diminuer les ressources d'infrastructure affectées pour répondre à leurs besoins. Amazon offre également d'autres services Big Data aux clients comme :

- Amazon Elastic MapReduce
- Amazon DynamoDB
- Amazon Simple Storage Service (S3)
- Amazon High Performance Computing
- Amazon RedShift

- **Google Big Data services**

Le géant de la recherche sur Internet Google, offre également un nombre de services Cloud ciblé pour Big Data avec un abonnement annuel de 40 € par utilisateur. Il s'agit notamment des éléments suivants :

- Google Compute Engine : Un Cloud basé sur la capacité des machines virtuelles.
- Google Big Query : Permet d'exécuter des requêtes similaires à SQL à grande vitesse sur des gigantesques ensembles de données de plusieurs milliards de lignes.
- Google Prediction API : Un Cloud basé sur les outils d'apprentissage pour de grandes quantités de données. La prédiction est capable d'identifier des modèles de données qui peuvent être analysés à des fins diverses, y compris la détection de fraude, l'analyse de désabonnement et l'impression du client.

– **Microsoft Azure**

Microsoft a produit un ensemble d'outils de développement, de supports de machines virtuelles et de services de périphériques mobiles dans une offre PaaS. Pour les clients possédant une grande expertise en .Net, SQL Server et Windows, l'adoption de PaaS Azure n'est pas assez compliquée. Pour répondre aux nouvelles exigences d'intégration de Big Data dans les solutions Windows Azure, Microsoft a également ajouté Windows Azure HDInsight basé sur Horton works Data Platform (HDP), qui selon Microsoft, offre une compatibilité à 100% avec Apache Hadoop. HDInsight prend en charge la connexion avec Microsoft Excel et d'autres outils d'informatique décisionnelle (BI), comme il peut être également déployé sur Windows Server. Plusieurs systèmes d'exploitation sont supportés.
Microsoft a changé sa politique de fournisseur de licences à un fournisseur de services. Microsoft Azure représente un portail qui offre l'ensemble des services Cloud proposés par Microsoft, principalement : Office 365, qui est une suite d'applications (office, exchange, Sharepoint, ...), Skype, Bing, Drive et autres services.
Microsoft affirme que 2/3 des solutions installées dans son Cloud ne sont pas des solutions Microsoft. La société est entrain, aussi, de préparer sa propre solution Unix.
Plus de 90 000 nouveaux clients, chaque mois, optent pour les solutions Cloud Microsoft.

I.2.8.2 Solutions libres

Un ensemble de solutions libres ont émergé également, citons principalement :

- Eucalyptus : IaaS livré en 2007 par l'université de Californie.
- OpenStack : IaaS créé en juillet 2010 par la NASA et l'hébergeur américain Rackspace.
- OpenNubela : IaaS livré en 2005 par l'université Complutense de Madrid. Notons qu'une version commerciale nommée « OpenNebulaPro » est disponible depuis 2010.
- CloudStack.

Toutes ces solutions se basent sur des outils de virtualisation tels que Xen, KVM, VMware ou Hyper V, etc.

Dans ce qui suit, on présente une brève description de certaines solutions Cloud open source très utilisées actuellement.

– **Eucalyptus**

Eucalyptus est une plateforme open source de Cloud Computing apparu en 2007 du projet VGrADS de l'université de Californie, Santa Barbara. Il permet d'exécuter des machines virtuelles dans un IaaS virtualisé [14].

Actuellement, Eucalyptus prend en compte des IaaS munis des systèmes de virtualisation types d'hyperviseurs Xen, KVM, VMware (choix privilégie KVM).
Eucalyptus organise l'IaaS de façon hiérarchique : les machines au niveau des feuilles, les clusters (groupe de machines) au niveau intermédiaire et le Cloud (ensemble de clusters) à la racine. Son fonctionnement est organisé de la même manière en associant à chaque niveau de l'IaaS un composant précis.

Eucalyptus est composé de cinq principaux éléments (Figure A.4) :

- Cloud Controller : C'est le point d'entrée unique pour les administrateurs et les utilisateurs. Ce module est responsable de la gestion de tout le système et surveille la disponibilité des ressources sur les autres composants.
- Node Controler : Son rôle est d'héberger KVM, il sert ainsi d'hyperviseur pour les machines virtuelles déployées et appelées des instances. Ce contrôleur fonctionne sur chaque nœud et se charge de vérifier le cycle de vie des instances.
- Cluster Controller : Il sert à déployer et gérer les différents contrôleurs de nœuds. Il gère le déploiement des instances entre des nœuds, et communique des informations au contrôleur du Cloud.
- Walrus : Assure le stockage des images de machines virtuelles, les prises en fonctionnement à un instant précis (snapshots).
- Storage Controller : fonctionne avec Walrus et permet de stocker les images des machines virtuelles, les données des utilisateurs, et le stockage des fichiers et services.

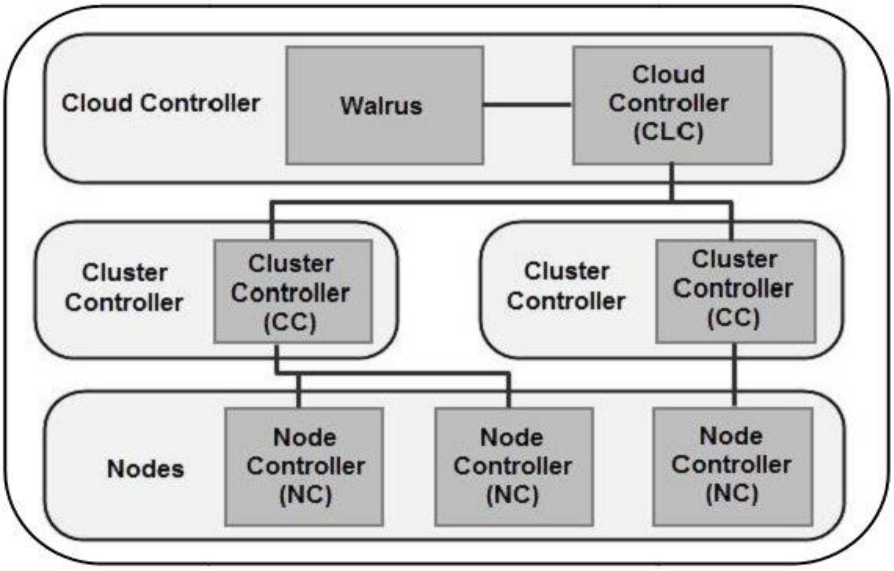


Figure A.4: Architecture Eucalyptus

– **Open Stack**

OpenStack est une offre IaaS 100% open source, créé en juillet 2010 par la NASA et l'hébergeur américain Rackspace. Ce projet est soutenu par plusieurs constructeurs tels que : AMD, Intel, Dell et Citrix [14].

OpenStack comprend le composant OpenStack Compute pour la création automatique et la gestion de grands groupes de serveurs privés virtuels ainsi que le composant OpenStack Stockage pour optimiser la gestion de stockage et répliquer le contenu sur différents serveurs et le mettre à disposition pour une utilisation massive de données.
OpenStack s'organise autour de trois composants et des API de communication :

- OpenStack Nova : Il fournit les fonctionnalités de gestion du cycle de vie des VM (viale sous composant Nova-compute), du réseau (via Nova-network) et des authentifications. Il implante également les programmes de scheduling de VM à travers son composant Nova-Scheduler. Le sous composant Queue Server implante le mécanisme de dispatching de requêtes aux autres sous composants en fonction des actions qu'elles requièrent.
- OpenStack Swift : permet de créer un service de stockage dans l'architecture Cloud. Il permet de gérer une large capacité de stockage évolutive redondante ainsi que le basculement entre les différents éléments de stockage.
- OpenStack Imaging Service : C'est un système de récupération et de recherche d'images de machines virtuelles.

– **OpenNebula**

OpenNebula a vu le jour en 2005 à l'université Complutense de Madrid dans le cadre du projet européen open source dont l'objectif est l'administration des IaaS virtualisés [14]. OpenNebula est capable de prendre en compte simultanément dans l'IaaS des hyperviseurs Xen, KVM et VMware. Il organise l'IaaS sous forme de clusters et de VLAN (réseaux virtuels). Un cluster contient un ensemble de machines physiques tandis qu'un VLAN est défini pour un ensemble de VM. Lors de la création d'une VM, le client choisit la machine et le VLAN dans lequel il souhaite l'exécuter.

Les composants d'OpenNebula peuvent être répartis sur trois couches :

- Tools : c'est l'ensemble des outils de gestion pour OpenNebula.
- Core : C'est un ensemble de composants pour contrôler les machines virtuelles, le stockage et le réseau virtuel.
- Drivers : C'est des pilotes spécifiques qui assurent l'interaction entre OpenNebula et l'infrastructure de Cloud.

Notons qu'une version commerciale d'OpenNebula (OpenNebulaPro) est disponible depuis 2010.

I.2.9 Simulateurs Cloud

Dans les offres de solutions Cloud (payants ou open source), il est impossible de procéder aux tests, expérimentations répétitives ou Benchmarking. Ceci est dû à l'environnent Cloud réel qui est partagé, dépendant et évolutif suivant la charge.

Certains propriétaires (HP, Intel et Yahoo) ont développé un projet appelé "Open Cirrus" dédié pour la simulation et les expérimentations, et qui supporte une fédération de Data Centers localisées sur 10 organisations. Cependant cette offre est coûteuse, toujours de nature partagée et difficile pour les expérimentations répétitives à cause de la variation des ressources dans le temps, en plus l'accès à Open Cirrus est limité juste aux membres de ses propriétaires.

Une autre alternative est d'utiliser des simulateurs existants de grilles de calcul, tels que GridSim, SimGrid, OptorSim, GangSim, etc. Néanmoins aucun simulateur de grille n'était capable d'isoler l'abstraction de service multi-couche (SaaS/PaaS/IaaS) et modéliser les ressources virtuelles requises par un Cloud.

Récemment, des simulateurs de Cloud ont émergé pour palier définitivement au besoin de simulation. Ces Frameworks sont extensibles, permettent la modélisation, la simulation et l'expérimentation.

Parmi ces simulateurs de Cloud, on peut citer : CloudSim, CloudAnalyst, GreenCloud, Network CloudSim, EMUSIM, MDC SIM, CDOSim, iCanCloud, VirtualCloud, TeachCloud, et autres [15].

I.2.10 Le Cloud Computing dans le contexte du Big Data

La puissance du Cloud provient du fait que les utilisateurs peuvent accéder aux ressources informatiques nécessaires avec le minimum de supports informatiques au lieu de s'investir dans le matériel ou logiciel. Une de ses caractéristiques clés est l'élasticité ou scalabilité horizontale : les utilisateurs peuvent ajouter ou retirer des ressources en temps quasi réel en fonction de la variation des exigences. Le Cloud joue un rôle important dans le grand monde des Big Data. Des améliorations spectaculaires se produisent lorsque ces infrastructures sont combinées avec les évolutions dans la gestion des données. Une infrastructure optimisée pour l'extensibilité horizontale favorise et appuie la mise en œuvre pratique des Big Data.

Deux exemples populaires confirmant les avantages du Cloud hébergeant des Big Data sont représentés par Google et Amazon.com. Les deux sociétés dépendent de leur capacité de gérer des quantités massives de données pour faire avancer leurs entreprises. Ces fournisseurs ont mis au point des infrastructures et des technologies pour soutenir des applications à une échelle massive. Considérons Gmail ou les millions de millions de messages que traite Google par jour dans le cadre de ce service. Google a été en mesure d'optimiser le système d'exploitation Linux et son environnement logiciel pour soutenir le courrier électronique de la manière la plus efficace ; Par conséquent, il peut facilement supporter des centaines de millions d'utilisateurs. Plus important encore, Google est capable de recueillir et d'exploiter les grands volumes de données sur ses utilisateurs de messagerie et de moteur de recherche pour piloter la société et améliorer ses services. De même, Amazon.com, optimise ses Data Center IaaS, pour supporter ses charges de travail afin qu'elle puisse continuer à offrir de nouveaux services et soutenir un nombre croissant de clients. Pour développer son commerce de détail, Amazon doit être en mesure de gérer les données sur ses marchandises, ses acheteurs, et son réseau de marchands partenaires. La publicité ciblée basée sur les habitudes d'achat des clients est essentielle au succès de l'entreprise. Ces entreprises offrent maintenant une gamme de services Cloud pour leurs Big Data [8].

I.2.11 Bases de données dans le Cloud Computing

Toutes les diverses offres du Cloud Computing participent fortement à la forte demande de migration des différents services des entreprises vers le Cloud. Le service des bases de données n'est pas exclu de ce courant.

Une base de données dans le Cloud est un type de service de base de données qui est développé, déployé et livré à travers une plate-forme dans le Cloud. Celle-ci permet aux organisations, aux utilisateurs finaux et leurs applications de stocker, gérer et récupérer des données à partir du Cloud généralement privé [13].

Plusieurs offres et modes de déploiement des bases de données dans le Cloud se proposent actuellement, malgré la présence de certaines contraintes qui peuvent ralentir le passage des bases de données vers le Cloud. Plusieurs recherches et efforts se font actuellement pour remédier à ces contraintes.

I.2.11.1 Modes de déploiement des bases de données dans le Cloud

Il y a principalement 3 modes de déploiement des bases de données dans les environnements Cloud qu'un client peut procurer, à savoir :

- Le client peut opter pour des bases de données dans le Cloud en payant un temps d'utilisation limité d'instances de machines virtuelles.
- Le client peut tout simplement acheter l'accès à un service de base de données maintenu et fourni par un fournisseur Cloud de base de données (Cloud Database Provider).
- Le client peut opter pour des bases de données hébergées et gérées dans une plate-forme du Cloud.

Ces bases de données peuvent être d'architectures différentes relationnelles, NoSQL ou même NewSQL.

- **Base de données sur l'instance d'une image de machine virtuelle (IaaS)**

Les plates-formes Cloud permettent aux utilisateurs d'acheter des instances de machines virtuelles pour un temps limité. Il est possible d'exécuter une base de données sur ces machines virtuelles. Les utilisateurs peuvent soit :

- Télécharger leur propre image de la machine avec une base de données installée ;
- Ou bien utiliser des images de machines prêtes à l'emploi qui incluent déjà une installation d'une base de données.

A titre d'exemple, Oracle fournit une image d'une machine toute prête (Ready-made-machine) avec une installation d'Oracle Database 11g Enterprise Edition sur Amazon EC2. Plusieurs solutions offrant des services IaaS qui peuvent être utilisés pour les Big Data comme Amazon.com, AT and T, GoGrid, Joyent, Rackspace, IBM et Verizon / Terremark.

– **Base de données en tant que service (SaaS)**

Certaines plates-formes Cloud offrent des options pour l'utilisation de la base de données en tant que service, sans avoir à lancer physiquement une instance de machine virtuelle pour la base de données. Dans cette configuration, les propriétaires d'applications n'ont pas à installer et à entretenir leur propre base de données. Le fournisseur de services de base de données se charge de l'installation et le maintien de la base de données et les propriétaires d'applications payent en fonction de leur utilisation.

Citons l'exemple d'Amazon Web Services, qui fournit trois services de base de données dans le cadre de ses offres Cloud :

- SimpleDB, une solution NoSQL de type clé-valeur ;
- DynamoDB, une solution NoSQL orientée document ;
- Amazon Aurora, MySQL-compatible RDBMS on Amazon RDS (Relational Database Service), un service de base de données relationnelle avec une interface MySQL.

– **Base de données hébergée et administrée dans le Cloud**

Il existe une autre variante ou le fournisseur de base de données dans le Cloud n'offre pas une base de données en tant que service, mais il assure seulement l'hébergement et l'administration de cette base à la place de l'utilisateur. Comme exemple, le fournisseur de service dans le Cloud Rackspace offre les services suivants :

- Hébergement et administration de MySQL sur une architecture dédiée ou dans leCloud.
- Bases de données NoSQL via Object Rocket's Managed MongoDB Service.

I.2.11.2 Caractéristiques communes aux bases de données en tant que service

Les bases de données en tant que service Cloud sont caractérisées par [15] :

– La plupart des services de base de données offrent des consoles Web utilisées pour la commander et configurer des instances de base de données.

Exemple : La console Web d'Amazon Web Services permet de lancer des instances de base de données, créer des clichés (Snapshots semblables à des sauvegardes des bases), et surveiller les statistiques de base de données.

- Les services de base de données comportent un composant de gestion de base de données, qui commande les instances de base de données sous-jacentes à l'aide d'une API de service. Cette API accessible à l'utilisateur final permet d'effectuer l'entretien et les opérations de mise à l'échelle sur les instances de la base.

  Exemple : L'API Relational Database Service d'Amazon permet la création d'un Snapshot d'une instance de base de données, la modification des ressources disponibles pour une instance donnée, la suppression d'une instance de base de données, la création d'une instance (similaire à une sauvegarde) d'une base de données et la restauration d'une base de données à partir d'un Snapshot.

- Les services de bases de données rendent la pile logicielle sous-jacente transparente pour l'utilisateur. La pile comprend généralement le système d'exploitation, la base de données et un logiciel tiers utilisé par la base de données. Le fournisseur de services est responsable de l'installation et la mise à jour de la pile logicielle sous-jacente.

  Exemple : ObjectRocket est la base de données de Rackspace MongoDB en tant que service.

- Les services de bases de données prennent en charge la mise à l'échelle ou l'extensibilité et la haute disponibilité de la base de données. La prise en charge de l'extensibilité varie d'un fournisseur à un autre. Certains offrent une mise à l'échelle automatique contrairement à d'autres qui permettent à l'utilisateur une mise à l'échelle manuelle à l'aide d'une API. Concernant la disponibilité, il s'agit généralement d'un engagement pour un certain niveau de haute disponibilité (par exemple de 99.9 % ou 99.99 %).

I.2.11.3 Modèles de données utilisés

Actuellement, il y a trois types de bases de données déployées dans le Cloud [16] :

- Les bases de données relationnelles traditionnelles dites SQL ;
- Les nouvelles bases de données s'inspirant sur le modèle relationnel dites NewSQL ;
- Les nouvelles bases de données non-relationnelles dites NoSQL (Not Only SQL).

– **Bases de données relationnelles**

Ce type de bases de données peut être utilisé dans l'environnement Cloud, en tant que machine virtuelle image ou comme un service, selon le fournisseur, mais elles ne répondent pas favorablement à l'extensibilité puisqu'elles ne sont pas nativement adaptées aux environnements distribués de grandes échelles, bien que les propriétaires du SQL tentent de relever ce défi.

La table A.1 ci-dessous, contient quelques exemples de bases de données relationnelles utilisées dans le Cloud, déployées dans des machines virtuelles ou en tant que service.

Déploiement sur machine virtuelle	Déploiement BD en tant que service
Oracle Database	Amazon Relational Database Service
IBM DB2	BitCan (MySQL, MongoDB)
Ingres (Database)	Microsoft SQL Azure (MS SQL)
PostgreSQL	Heroku PostgreSQL as a Service (Shared and Dedicated Database Options)
MySQL	Clustrix Database as a Service
NuoDB	EnterpriseDB Postgres Plus Cloud Database

Table A.1 : Exemples de bases relationnelles déployées en tant que service dans le Cloud

– **Bases de données NoSQL**

C'est une nouvelle génération de bases de données conçues nativement pour le Cloud Computing et les Big Data. Elles sont développées pour répondre favorablement au passage à l'échelle et supporter les lourdes charges de lecture / écriture.

Ces systèmes NoSQL qui feront l'objet de notre première étude, seront détaillés dans la section II.2.

– **Bases de données NewSQL** [17]

Dans la majorité des cas, la technologie NoSQL 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.).

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).

Des généralités sur cette nouvelle génération de gestion de données NewSQL, non étudiée dans ce manuscrit, sera présentée dans la section II.3.

I.2.11.4 Avantages et inconvénients des bases de données dans le Cloud

Certainement le déploiement des bases de données dans les environnements du Cloud Computing peut faire profiter davantage les utilisateurs, néanmoins il y a quelques contraintes et risques qu'il faut estimer avant de décider une telle migration :

– **Avantages** [18]

- **Réduction des coûts**

  Il est économiquement très intéressant d'échanger les lourdes dépenses liées à l'installation, la maintenance et la mise à niveau d'une infrastructure informatique sur site contre le coût d'exploitation d'un abonnement à une base de données dans le Cloud, notamment sur le court et moyen terme. Toutefois, il est important d'avoir conscience des coûts potentiels cachés associés à l'adoption de cette stratégie.

- **Elasticité**

  A mesure que l'entreprise se développe et le nombre d'utilisateurs augmente, plutôt que d'investir dans des licences logicielles et des capacités supplémentaires de serveur en interne, il existe des prestataires qui permettent d'ajuster l'abonnement mensuel selon les besoins de l'entreprise.

- **Accessibilité**

  En général, un navigateur et une bonne connexion à internet suffisent pour accéder à une base de données dans le Cloud. Ces solutions sont mises à disposition sur divers postes de bureau et appareils mobiles.

- **Sécurité**

  C'est un avantage dans le sens où la sécurité est mieux prise en charge dans les plateformes Cloud pour les entreprises qui n'ont pas les moyens et l'expertise nécessaire.

- **Résilience**

  Puisque l'infrastructure informatique et les données résident dans le centre de traitements du prestataire de services Cloud, alors en cas où les locaux de l'entreprise subissent un

sinistre quelconque, il est relativement facile de refonctionner les services de l'entreprise à partir d'un autre site équipé d'ordinateurs connectés à internet.

– **Inconvénients** [19]

- **Sécurité**

  C'est la première préoccupation pour les entreprises. L'inconvénient de la sécurité réside dans le sens de l'hébergement. La plateforme Cloud, si elle est externe (non installée sur le réseau interne ou avec une ouverture vers l'extérieur) doit être suffisamment sécurisée pour éviter le risque d'intrusion ou de vol des données par piratage. Plus généralement, une clause de confidentialité et la confiance dans son personnel sont primordiales pour que les données ne fuitent pas de manière volontaire.

- **Connexion**

  C'est le problème du goulot d'étranglement qui peut atténuer les performances. Si la connexion Internet est insuffisante (débit faible, coupures fréquentes), l'utilisateur ne pourra pas accéder à sa plateforme de travail. L'idée dans ce cas est de continuer le travail sur une application locale qui synchronise ensuite les données avec le serveur dès que l'utilisateur a de nouveau accès au réseau.

- **Interruptions de service**

  Parmi les objectifs des fournisseurs du Cloud, c'est bien planifier les interruptions de service qui sont inévitables, qu'elles soient imputables à une catastrophe naturelle, à une erreur humaine ou aux nombreuses causes intermédiaires. Un temps d'arrêt est toujours agaçant, mais une interruption de service qui s'éternise peut s'avérer désastreuse lorsqu'elle touche une application cruciale. Le client devra examiner le contrat de niveau de service (SLA) et les performances historiques de son prestataire de services très attentivement avant d'externaliser des applications cruciales de son entreprise vers une plateforme dans le Cloud public. Des outils tels que « Outage Analyzer » de Compuware et «Is It Down Right Now?» permettent de surveiller en continu les interruptions de service dans le Cloud.

- **Coût**

  Sans une étude de rentabilité, à long terme le coût peut être un inconvénient.

- **Conformité**

  Lorsque les données de l'entreprise résident dans le centre de traitements d'un prestataire de services, il peut être difficile de s'assurer de la conformité avec les réglementations gouvernementales pertinentes en matière de protection des données.

- **Performances**

  Une application hébergée dans un centre de traitements à distance et accessible via une connexion internet peut susciter des inquiétudes quant aux performances, par rapport à un logiciel s'exécutant sur une machine locale ou viable réseau local de l'entreprise. Bien évidemment, certaines tâches seront mieux adaptées que d'autres au modèle de SaaS. Les outils de gestion des performances des applications peuvent aider les clients et les fournisseurs à surveiller le fonctionnement de leurs applications.

- **Mobilité des données**

  Le marché des SaaS grouille de startups, dont certaines connaîtront inévitablement l'échec. Que deviennent-il des données si le prestataire de services met la clé sous la porte ou si le client devra changer de fournisseur de SaaS pour une raison ou une autre ? Lorsqu'un client choisi un fournisseur de SaaS, il doit s'assurer d'éviter tout emprisonnement en préparant une stratégie de sortie.

- **Intégration**

  Les entreprises qui adoptent de multiples applications de SaaS, ou qui souhaitent connecter des logiciels hébergés à des applications existantes sur site, sont confrontées au problème de l'intégration des logiciels. S'il n'est pas possible de gérer les bibliothèques, les API et structures de données pertinentes en interne, il existe une catégorie relativement nouvelle de produits d'intégration en tant que service, notamment Boomi (société détenue par Dell), CloudSwitch et Informatica.

Chap II : Limites des systèmes relationnels et mouvances NoSQL et NewSQL

Les bases de données en général et le modèle relationnel en particulier existent depuis 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, Yahoo, Facebook. En effet ce genre de sites possède plusieurs millions voire des milliards d'entrées dans leurs bases de données distribuées sur plusieurs machines, produits par des millions d'utilisateurs éparpillés partout dans le monde. Pour des raisons de fiabilité, ces bases de données sont dupliquées pour que le service ne soit pas interrompu en cas de panne. Pour répondre à ces nouveaux besoins, plusieurs réflexions de conception de nouvelles architectures ont été imposées, et qui ont émergé plusieurs modèles et solutions pour la gestion de données, principalement le NoSQL et le NewSQL.

Les différentes solutions de gestion des données conçues ces dernières années, sont caractérisées et comparés à base de différents facteurs, citons les plus importants :

- **Performance** : la performance se caractérise par la latence qui est le temps d'attente avant l'arrivée du premier octet et le débit qui est la quantité d'octets transférés.
- **Extensibilité** (élasticité ou scalabilité) : Capacité de prise en charge des accroissements de toute nature, volumétrie, nombre de clients, etc., avec des temps de réponse plus ou moins constant.
- **Disponibilité** : Capacité de fonctionnement en continu sans défaillance.

Ces critères jugés comme primordiaux, sont les indicateurs de base de réussite d'un modèle par rapport à un autre.

Avant de présenter ces nouveaux modèles, on va discuter sur les motivations et les raisons qui ont poussé les différents acteurs à chercher à adapter leurs modèles relationnels aux nouvelles tendances ou carrément les abandonner et chercher d'autres alternatives.

II.1 Les systèmes relationnels et leur limites atteintes

Le modèle relationnel a été inventé par Edgar Codd [20], un scientifique d'IBM, dans les années 1970 et a été utilisé par IBM, Oracle, Microsoft, et d'autres. Il était, et reste encore largement utilisé aujourd'hui en raison de sa simplicité et de ses solides fondations mathématiques. Dans ce modèle, les données sont stockées dans une table, l'ensemble de ces

tables constituent une base de données relationnelle ayant un schéma. Ce dernier est une représentation structurelle du contenu de la base de données, définissant les tables, les champs des tables et les relations entre les deux. Les données sont stockées sous forme de tableaux ou chaque colonne correspond à un attribut spécifique et chaque ligne correspond à un enregistrement ou un tuple. Les données de chaque table peuvent être lues, supprimées et mises à jour en utilisant le langage standard pour les bases de données relationnelles SQL (Structured Query Language). La gestion, le stockage, la mise à jour, le partage, la cohérence et la sécurité des données sont assurés par un SGBD Relationnel (RDBMS).

Face à l'évolution informatique dans la dernière décennie et notamment les grands volumes de données échangés, les SGBDR classiques ont montré de très grandes faiblesses en matière de scalabilité et en montée en charge, principalement à cause du respect des propriétés ACID, ce qui a obligé les grands acteurs du web à chercher d'autre solutions. Les systèmes NoSQL viennent, justement, pour accomplir cette mission.

II.1.1 Propriétés ACID

Les systèmes de bases de données relationnelles répondent bien au besoin transactionnel en respectant les contraintes ACID : Atomicité, Consistance ou Cohérence, Isolation et Durabilité ; l'acronyme ACID permet de garantir la fiabilité d'une transaction.

II.1.1.1 Atomicity (Atomicité)

Lorsqu'une transaction est effectuée, toutes les opérations qu'elle comporte doivent être complètement effectuées, ou pas du tout : en effet, en cas d'échec d'une seule des opérations, toutes les opérations précédentes qui font partie de la même transaction doivent être complètement annulées, peu importe le nombre d'opérations déjà réussies.

Prenons l'exemple d'une transaction comportant 100 opérations élémentaires : Si après l'exécution réussie de 80 opérations, la 81^{éme} échoue, alors la transaction entière va être annulée, impliquant l'annulation des 80 opérations déjà effectuées. L'annulation complète de la transaction est justifiée par le fait que chaque opération faite peut dépendre du contexte de l'autre, et toute rupture de ce contexte pourrait engendrer une incohérence des données.

II.1.1.2 Cohérence (Consistency)

Une base de données est dite cohérente si toutes les contraintes d'intégrité définies sont vérifiées. Le SGBD veille à ce que toutes ces contraintes soient vérifiées à chaque insertion, suppression ou modification d'une donnée. Une base de données doit être toujours dans un état cohérent, avant et après l'exécution d'une transaction. Si le contenu d'une base de données

va contenir des incohérences, après réalisation d'une certaine transaction, cela va entraîner l'échec et l'annulation de toutes les opérations de la dernière transaction. Le système revient au dernier état cohérent.

II.1.1.3 Isolation (Isolation)

La caractéristique d'isolation permet à une transaction de s'exécuter en un mode isolé. 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. C'est le système qui garantit aux autres transactions exécutées en parallèle une visibilité sur les données antérieures. Ce fonctionnement est obtenu grâce aux verrous système imposés par le SGBD. Prenons l'exemple de deux transactions concurrentes T1 et T2 : les modifications effectuées par T1 ne sont ni visibles, ni modifiables par T2 tant que T1 n'a pas été accompli et validé.

II.1.1.4 Durabilité (Durability)

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

II.1.2 Contrainte de Cohérence dans un environnement distribué

Quand les propriétés ACID sont satisfaites, la cohérence du système d'information est assurée malgré les exécutions concurrentes et les défaillances.

Historiquement, le premier modèle de cohérence des BD dans un environnement distribué a instauré le principe fondamental de la réplication et un certain nombre de techniques pour réaliser la cohérence, citons quelques-unes :

- Toute réplication apparaît en tant qu'une seule base de données à l'utilisateur.
- Si une opération de mise à jour est requise à une réplique de base de données, toutes les répliques du système doivent être mise à jour pour valider la requête.
- Une valeur d'une donnée ne peut être retournée jusqu'à ce que toutes les copies de la base de données puissent renvoyer la même valeur.

Ce modèle de cohérence est resté relativement statique jusqu'à l'explosion de la volumétrie des données, ce qui a ramené à une forte distribution et réplication sur les réseaux.

En Cloud, la réplication sur différentes zones géographiques est constamment utilisée alors pour maintenir la cohérence, le réseau sera très sollicité quoique ce soit très difficile d'assurer cette cohérence si le réseau est peu fiable. Alors conserver la cohérence d'une telle réplication

est très coûteux en termes de performances, puisque les mises à jour des données nécessitent la notification de toutes les répliques concernées (envoi de messages à toutes les répliques). S'il n'y a pas de réponses de mise à jour de toutes les répliques, à cause d'une faille réseau, toutes les répliques doivent attendre, ce qui va engendrer des transactions inefficaces et éventuellement des dégradations de performances.

La gestion de stockage dans le Cloud priorise la disponibilité et la durabilité des données qui sont les principaux objectifs des fournisseurs du Cloud, alors toute perte de données ou indisponibilité d'un service peut remettre en cause la réputation d'une entreprise. Des études indiquent que la latence est un facteur essentiel dans les interactions en ligne : une augmentation aussi faible de 100 ms réduit considérablement la probabilité qu'un client va continuer à interagir ou retourner dans l'avenir.

Malheureusement, il ya un compromis fondamental entre la cohérence, la disponibilité ou la latence, puisque la disponibilité et la latence sont deux indicateurs de performances dépendants : un système indisponible fournit essentiellement une latence extrêmement élevée. Une latence de quelques secondes est considérée comme une indisponibilité.

Alors, la solution adoptée dans les environnements distribués, est de sacrifier ou compromettre la cohérence pour atteindre les principales caractéristiques d'une plate-forme Cloud : évolutivité, disponibilité et fiabilité. Comme pour les poids lourds du net (Google, Facebook, Yahoo, Amazon, Linkedln) qui réduisent la cohérence puisqu'ils sont conçus pour une très haute disponibilité par la réplication sur un réseau WAN.

II.1.3 Limites des systèmes relationnels dans le Cloud et extension aux clusters

Le modèle relationnel est fondé sur des concepts simples qui font sa force en même temps que sa faiblesse. En effet, les systèmes de bases de données relationnelles répondent bien au besoin transactionnel grâce aux propriétés ACID (Atomicité, Cohérence, Isolation, Durabilité). De même, les extensions faites sur ce modèle (tel que les cubes, le schéma en étoile, etc.) permettent de prendre en charge le besoin d'analyse dans les grandes bases consolidées (Data Warehouse). Néanmoins, ces systèmes relationnels ne peuvent pas être déployés dans un environnement Cloud à grande échelle, en conservant les mêmes performances.

Principalement, les problèmes du modèle relationnel liés à ce passage à l'échelle sont [21] :

II.1.3.1 Application des propriétés ACID en milieu distribué

Les propriétés ACID, bien que nécessaires à la logique du relationnel, nuisent fortement aux performances, et particulièrement la propriété de cohérence. En effet, la cohérence est très

difficile à mettre en place dans le cadre de plusieurs serveurs, car pour que celle-ci soit respectée, tous les serveurs doivent être des miroirs les uns des autres, ce qui va multiplier le coût de mise à jour des données puisqu'on ne peut pas valider une transaction que si on est certain qu'elle été effectuée sur tous les serveurs alors le système fait patienter l'utilisateur durant tout ce temps (Figure A.5). La majorité des fournisseurs de services Cloud ne peuvent pas accepter une telle latence, vu que les principaux avantages du Cloud est la mise en échelle des services déployés (élasticité ou scalabilité), la disponibilité et minimiser au maximum la latence [22]. Dans un environnement Big Data, les contraintes ACID du relationnel, bien qu'elles garantissent la cohérence des données, peuvent devenir dans certains cas un facteur de blocage très coûteux. Par exemple, dans le cas du commerce électronique ou la recherche dans internet où le client d'informations s'intéresse plus à avoir une réponse instantanée sans autant exiger que ces informations soient à jour instantanément.

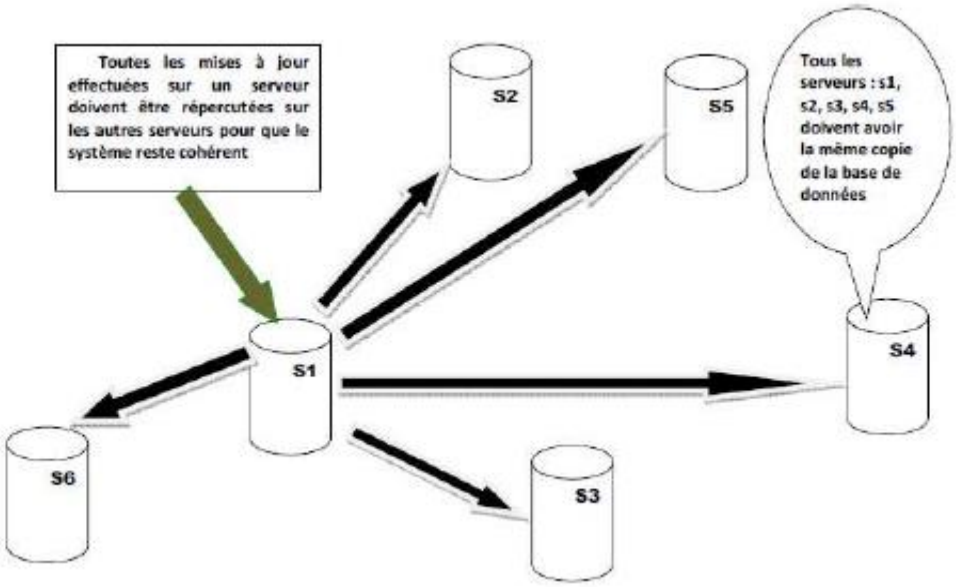


Figure A.5 : Problème lié aux propriétés ACID en milieu distribué [23]

II.1.3.2 Scalabilité limitée

Les SGBD Relationnels ne sont pas adaptés aux environnements distribués requis par les volumes gigantesques de données et par les trafics tout aussi importants comme dans le cas de traitements de données en masse chez les poids lourds du Net (Yahoo, Google, Facebook, Twitter, Amazon et autres).

Pour y remédier, certains fournisseurs de bases de données relationnelles ont élargi les possibilités de leurs systèmes en intégrant de nouvelles fonctionnalités pour supporter la gestion de leurs bases de données dans un environnement multi-nœuds. On cite à titre

d'exemple Oracle RAC (Real Application Server) [24], MySQL Cluster [25] et IBM DB2 [22]. Pour le cas le plus simple d'un cluster à deux nœuds, Oracle RAC (inclus par défaut dans la version Standard) peut assurer efficacement la répartition des charges (Load Balancing) et le basculement en cas d'échec (Fail Over). Plus on ajoute les nœuds au cluster, plus la gestion d'Oracle RAC se complique. En atteignant un certain nombre de nœuds, Oracle RAC plante ou devient inefficace. Dans un environnement Cloud où le nombre des nœuds est très grand (des centaines ou des milliers), ces solutions propriétaires de Clustering relationnels deviennent inexploitables.

II.1.3.3 Requête de jointure non optimale

Admettons une table contenant toutes les personnes ayant un compte sur Facebook, soit un milliards et demi d'utilisateurs actifs par mois, les données dans une base de données relationnelle classique sont stockées par lignes, ainsi pour effectuer une requête pour extraire tous les amis d'un usager donné, il faudra effectuer la jointure entre la table des utilisateurs et celle des amitiés, qui va engendré une énorme perte en performances en raison du temps consommé pour stocker et parcourir une telle quantité de données [26].

II.1.3.4 Gestion des objets hétérogènes

Mise à part, la contrainte de l'adaptation des systèmes relationnels avec les environnements distribués, la nature des données hétérogènes manipulées se voit comme un autre fardeau pour ces modèles. En effet, au fur et à mesure du temps, les structures de données gérées par les systèmes sont devenues de plus en plus complexes en contrepartie les moteurs de stockage évoluant peu. Parmi les principaux points faibles 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 [26].

II.1.3.5 Types de données limités

Le modèle relationnel prend en charge certains types de données qui sont assez simple : entiers, réels, chaînes de caractères, booléen ou même date. En revanche, il est très mal adapté à la gestion de données multi-valuées complexes [27]. Les seuls types étendus sont les BLOB (Binary Large OBjects) qui permettent de stocker des images ainsi que des fichiers audios ou vidéos. Ces BLOBs ne sont toutefois pas suffisants pour représenter des données complexes n'ayant pas une structure prédéfinie. Rappelons que la part de cette catégorie de données non structurées est de plus en plus dominante par rapport à l'ensemble des données produites actuellement. Les mécanismes de contrôle dans les bases de données traditionnelles sont

inexistants et le langage de requêtes (SQL) ne possède pas les opérateurs correspondant aux objets stockés soit dans ces BLOBs soit dans d'autres types [28].

II.1.3.6 Langage de manipulation

Un autre inconvénient est lié au fait que le langage SQL n'est pas un langage de programmation. Le développement d'une application autour d'une base relationnelle nécessite l'utilisation d'autres langages évolués qui peut engendrer un problème de concordance des types entre le SGBD et le langage de développement [27]. En effet ces incompatibilités de types apparaissent lors de l'utilisation de SQL dans un langage hôte comme pour la représentation des nombres. Des conversions de types sont nécessaires, et peuvent s'avérer coûteuses en temps et parfois difficiles à mettre en œuvre [28].

II.1.3.7 Pauvreté sémantique

Le modèle relationnel s'appuie sur un seul concept pour modéliser à la fois les entités et les associations entre ces entités, à savoir la table relationnelle. Donc toutes les règles de gestion conceptuelles ainsi que les différents liens sémantiques qui peuvent exister dans un système d'information vont être traduits en une table contenant un ensemble d'attributs, clé primaires et clés étrangères, ce qui crée un décalage entre la réalité et cette représentation abstraite [28].

D'autre part La pauvreté sémantique du modèle relationnel ne permet pas de prendre en compte efficacement les nouveaux besoins liés aux informations multimédia. Même, si on arrive à stocker ces données dans des tables relationnelles, le traitement du contenu de ces données de type image, audio ou vidéo et le code SQL correspondant ne sera pas forcément efficace [29].

II.2 Le NoSQL

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

II.2.1 L'émergence du NoSQL

Le volume de données de certaines entreprises est augmenté considérablement durant ces dernières années. La vulgarisation de l'informatique sur tous les plans et dans les différents domaines a eu pour conséquence une augmentation exponentielle des volumes de données qui se compte désormais en péta octets. Les grands acteurs du web tels que Facebook, Google, Yahoo, Amazon et LinkedIn ont été rapidement limités par les systèmes traditionnels pour deux raisons majeures :

- Les grandes masses de données hétérogènes.
- Les montées en charge.

N'ayant pas trouvé de solution sur le marché répondant à ces problèmes, alors, ils décidèrent de solutionner leurs problèmes en concevant de nouvelles architectures et en développant chacun à l'interne et en parallèle de leurs activités principales, leurs propres SGBD.

Alors Amazon a développé le DynamoDB pour son commerce électronique. Facebook a employé Cassandra puis HBase pour la fonction de recherche dans la boîte de réception. Le géant Google a développé Bigtable, pour assurer la gestion de son Big Data, utilisée par beaucoup de services Google, tels que la recherche, Maps et Gmail. Pour son côté, Yahoo a construit Pnuts pour stocker, les données des utilisateurs qui peuvent être lues ou écrites sur toutes les page Web, les listes de données pour les pages de shopping de Yahoo ainsi que les données pour servir ses applications de réseau social. LinkedIn a conçu Voldemort pour gérer les mises à jour en ligne à partir de diverses caractéristiques d'écritures intensives sur son site internet.

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, ont affecté aussi d'autres nombreuses entreprises de tous types d'industries confondues, c'est de ce constat qu'est né le mouvement NoSQL.

En 1998, le monde entend pour la première fois le terme NoSQL. Terme inventé et employé par Carlo Strozzi pour nommer son SGBD relationnel Open Source léger qui n'utilisait pas le langage SQL. Ironiquement, la trouvaille de M. Strozzi n'a rien à voir avec la mouvance NoSQL que l'on connaît aujourd'hui, vu que son SGBD est de type relationnel. En effet, c'est en 2009, lors d'un rassemblement de la communauté des développeurs des SGBD non-relationnels, que le terme NoSQL a été mis au goût du jour pour intituler tous les SGBD de type non-relationnel [30]. C'est à partir de cette année, que les bases de données NoSQL ont

commencé à émerger, pour répondre aux nouveaux besoins requis par les environnements Cloud et Big Data.

II.2.2 Définition et concepts de base

« **NoSQL** » est une combinaison de deux mots : No et SQL. Le nom peut sembler comme une opposition aux bases de données SQL, et pourrait être mal interprétée en pensant que cela signifie la fin du langage SQL et qu'on ne devrait donc plus l'utiliser. En fait, le « No » est un acronyme qui signifie « **N**ot **o**nly » « Non seulement », c'est donc une manière de dire qu'il y a autre chose que les bases de données relationnelles et le SQL.

L'idée et la fonction initiale du mouvement n'étant pas de remplacer les systèmes relationnels dits SQL, mais de proposer des alternatives ou compléter les fonctionnalités de ces modèles pour manipuler des grandes masses de données distribués et s'adapter aux nouvelles tendances et architectures du moment, notamment le Cloud Computing [30].

Les axes principaux du NoSQL sont une haute disponibilité et un partitionnement horizontal des données, au détriment de la cohérence, contrairement aux bases de données relationnelles basées sur les propriétés ACID.

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

Contrairement aux SGBD traditionnels et les entrepôts de données, ces nouveaux modèles de stockage de données sont distribués sur de nombreux serveurs et conçus pour s'adapter à des milliers ou des millions d'utilisateurs qui font des mises à jour ainsi que les lectures. Ils s'appuient sur les systèmes de fichiers parallèles pour améliorer les performances et remédier aux limitations des ressources en multipliant les équipements pour permettre la parallélisation de stockage et l'accès aux informations.

Une base de données pouvant se résumer topologiquement à un simple tableau associatif unidimensionnel avec des millions - voire des milliards - d'entrées. Cette conception simplicite a permis de monter en charge à des péta-octets de données et des milliers de requêtes simultanées [6, 7].

Il n'est toutefois pas simple de définir explicitement ce qu'est une base de données NoSQL étant donné qu'aucune norme n'a encore été instaurée jusqu'à présent. Ainsi, nous pouvons définir le NoSQL comme suit [31] :« NoSQL est un ensemble de concepts qui permet un traitement rapide et efficace des données en insistant sur la performance, la fiabilité et l'agilité ».

II.2.3 Intérêts

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

II.2.3.1 Scalabilité horizontale au lieu de scalabilité verticale

La « scalabilité » est le terme utilisé pour définir l'aptitude d'un système à maintenir un même niveau de performance et avec des temps de réponse plus ou moins constants, face à la montée en de charge de toute nature : volumétrie de données, nombre de clients, etc., par augmentation des ressources matérielles [32].

Il y a deux manières de rendre un système extensible : la scalabilité horizontale (externe) et la scalabilité verticale (interne), comme il est montré dans la Figure A.6.

Figure A.6 : Scalabilité horizontale et verticale

- **Scalabilité *verticale (Scale-up)*** : Les bases de données classiques adoptent encore souvent un modèle centralisé : un serveur unique, éventuellement redondé en mode actif/passif pour la disponibilité. La solution la plus courante pour supporter plus de transactions est la scalabilité verticale qui consiste à augmenter les performances matérielles en modifiant les ressources du serveur, comme par exemple le remplacement du CPU par un modèle plus puissant, augmenter la vitesse d'horloge, l'augmentation de la mémoire RAM ou même l'extension des unités d'entrées/sorties. La croissance interne consiste au fait d'évoluer une machine dans le temps ou carrément la remplacer par une machine plus puissante.

- **Scalabilité *horizontale (Scale-out)*** : Le principe de la scalabilité horizontale consiste à rajouter des serveurs supplémentaires parallèles, c'est la raison pour laquelle on dit qu'il s'agit de croissance externe. On part d'un ensemble de serveurs basiques avec lesquels on rajoute dans le réseau des nouveaux serveurs identiques, qui vont permettre de fonctionner sur plusieurs CPU, afin de répondre au passage à l'échelle pour supporter les énormes volumes de données et un nombre de requêtes très élevé et assez compliqué [31]. Le Cloud Computing est conçu pour le Scale-out, basée sur des architectures distribuées utilisant un grand nombre de serveurs, qui sont généralement moins onéreux, chacun hébergeant des copies de la base de données. Dans le même contexte, le but des systèmes NoSQL est de renforcer la scalabilité horizontale, pour répondre à des besoins spécifiques de performance et assurer une extensibilité extrême sur de grands volumes de données distribuées. Ce qui motive l'association naturelle entre le NoSQL et le Cloud, puisqu'actuellement la plupart des solutions Cloud sont basées sur le NoSQL.

II.2.3.2 Gestion de gros volume de données

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

II.2.3.3 Performance en écriture

Avoir des meilleures performances en écriture est l'intérêt principal des géants Google, Facebook (135 milliards de messages par mois) et Twitter (7 TB de données par jour). La gestion des masses de données, augmentant considérablement chaque année, exige un temps très important pour le stockage de ces volumes dans les différents serveurs (à 80MB/s, on a besoin d'une journée pour stocker 7TB). En conséquence, l'écriture se doit être distribuée sur un cluster de machines, ce qui nécessite d'autres outils et techniques comme le MapReduce, la réplication, la tolérance aux pannes et un certain niveau de cohérence [16].

II.2.3.4 Types de données flexibles

Parmi les innovations majeures caractérisant les solutions NoSQL, est le fait que celles-ci supportent de nouveaux types de données. N'étant pas enfermée dans un seul et unique modèle de données, une base de données NoSQL est beaucoup moins restreinte qu'une base SQL. Les applications NoSQL peuvent donc stocker des données sous n'importe quel format ou

structure, et changer de format en production. Les objets complexes peuvent être mappés sans trop de complications avec le modèle adopté [16].

II.2.3.5 Structure dynamique

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

II.2.3.6 Migration de données

L'absence de modèle dans un système NoSQL, joue un rôle très favorable dans le processus de migration éventuelle de données vers ces systèmes. En effet le modèle est en quelque sorte dynamique puisqu'il est créé par l'application pendant son exécution.

II.2.3.7 Acidité relative

Bien que ce ne soit pas la priorité du NoSQL, néanmoins, il existe des solutions qui permettent de conserver complètement ou partiellement les aspects des propriétés ACID. Ces compromis effectués vont être décrits dans la section qui réfère au théorème CAP et aux propriétés BASE.

II.2.3.8 Economie

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

II.2.3.9 Simplicité de développement

L'accès aux données, soit d'une manière interactive avec la base de données ou par le biais d'une application, est assez simple. Contrairement au modèle relationnel qui n'est pas très intuitif pour un développeur, puisqu'il devra être en mesure de résoudre les différents problèmes de base de données en absence du DBA [16].

II.2.4 Caractéristiques

Voici les caractéristiques principales d'une base de données NoSQL [31, 33] :

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

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

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

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

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

- Non-existence de langages de requête normalisés tels que SQL ;
- Cohérence de données abandonnée relativement au profit de la haute disponibilité ;
- Sécurité des données et manque d'autres fonctionnalités supplémentaires insuffisamment développées : Les systèmes NoSQL se base sur l'application pour la protection des données.
- Maturité et stabilité : Les bases relationnelles ont une longueur d'avance sur ce point. Les utilisateurs sont familiers avec leur fonctionnement et ont confiance en elles.
- Manque d'architectures et d'interfaces normalisées ;
- Moins de documentation et d'outils disponibles.

II.2.5 Du SQL vers le NoSQL

Le besoin fondamental auquel répond le NoSQL est la performance. Afin de résoudre les problèmes liés au « BigData », les développeurs de sociétés ont procédé à des compromis sur les propriétés ACID des SGBDR. Ces compromis sur la notion relationnelle ont permis à ces nouveaux systèmes de se libérer de leurs freins à la scalabilité horizontale. Le NoSQL renonce aux fonctionnalités classiques des SGBDR au profit de la simplicité. Les performances restent bonnes avec la montée en charge (scalabilité) en multipliant simplement le nombre de serveurs, solution raisonnable avec la baisse des coûts. Les systèmes géants comme Google, Facebook, Yahoo, Amazon sont les premiers concernés : énorme quantité de données, structuration relationnelle de moindre importance que la capacité d'accès très rapide, quitte à multiplier les serveurs.

Le NoSQL se distingue du relationnel (SQL) sur plusieurs aspects, les plus importants :

- L'absence de schéma et la structure imposée ;
- La non considération des propriétés ACID pour une transaction au profit de la performance.

Le modèle relationnel est basé sur les mathématiques des ensembles ou chaque ensemble est représenté par une table, les colonnes de la table quant à eux représentent les attributs. L'un des principes fondamentaux est la notion de relation entre tables à l'aide de cardinalités, clés primaires et clé étrangères, ceci implique au préalable une étude minutieuse sur la

modélisation du schéma de la base de données. Une fois ce modèle mis en place, il est difficile de changer la structure de celui-ci, ce qui pose des problèmes lors de sa réingénierie.

Le mouvement NoSQL est plus pragmatique, basé sur des besoins de stockage de données ainsi qu'une liaison plus forte avec les différents langages clients. La plupart des moteurs de base de données NoSQL n'utilisent pas de schémas prédéfinis à l'avance, d'où l'appellation « Schema-Less », ainsi on peut avoir dans la même table des enregistrements avec des champs différents en type ou en nombre ce qui n'oblige pas le moteur de la base de données à faire des vérifications et imposer des contraintes de schéma. Les données étant organisées du coté code client. Toutefois ce principe ne se vérifie pas entièrement en pratique, car le maintien d'une structure de données homogènes est important, pour des questions d'indexation, de recherche par critère ou de logique. La notion de « Schema-Less » en NoSQL implique aussi que la gestion des valeurs Null n'existe pas, ce qui évite la gestion supplémentaire de ces valeurs Null du relationnel. L'autre différence importante, est que dans les bases relationnelles, une transaction doit respecter les propriétés ACID afin que celle-ci soit validée. Le respect de ces propriétés dans un environnement distribué est coûteux puisque le risque de diminution des performances est proportionnel aux nombres de serveurs (nœuds). Les moteurs NoSQL font l'impasse sur ces propriétés, leurs priorités étant d'augmenter les performances par l'ajout de nœuds. Les mises à jour sont propagées éventuellement, mais les garanties sur la cohérence des lectures sont limitées. Quelques auteurs proposent l'acronyme « BASE » (voir section II.2.7) contrairement à « ACID » [34].

D'autre part, l'idée de combiner des bases de données NoSQL et des bases de données relationnelles a été évoqué dans une solution hybride [35]. D'autres systèmes ont émergé qui tentent de supporter le traitement de requêtes SQL sur des données stockées dans HDFS d'Hadoop (voir partie B section II.2.4) nommés SQL-on-Hadoop Systems [36, 37]. Ces solutions ne peuvent pas être considérées comme des bases de données car elles sont principalement conçues pour améliorer le langage de requête Hive (voir partie B section II.2.5.1). Une autre catégorie de modèles de données vise à fournir au modèle relationnel les avantages d'extensibilité horizontale et la tolérance aux pannes assurées par les solutions NoSQL, qui est le NewSQL. Ces nouveaux systèmes NewSQL ainsi que le NoSQL se présentent comme des alternatives capables de gérer d'énormes volumes de données. NewSQL est issu de trois types d'architectures : relationnelle, NoSQL et grille de données. Toutes ces solutions NewSQL prennent en charge le modèle relationnel en utilisant le SQL

comme langage de requête, même si elles sont fondées sur des hypothèses et architectures différentes des SGBDR, comme Google Spanner, VoltDB, Clustrix et NuoDB [38, 39, 40].

II.2.6 Théorème CAP et ses critiques

Le choix entre la cohérence de données ou la disponibilité de celles-ci, est un des sujets les plus sensibles entre les deux mondes que constituent le relationnel et le NoSQL. Dans un système relationnel, les différents utilisateurs voient la base de données dans un état cohérent, les données en cours de modification par un utilisateur n'étant pas visibles aux autres utilisateurs puisqu'un verrou a été posé. Le maintien de cette cohérence dans une architecture centralisée est tout à fait envisageable, cela devient cependant problématique dans le contexte d'une architecture distribuée.

Les fondateurs du NoSQL, dont les besoins en disponibilité priment sur la cohérence des données, ont décidé de privilégier celle-ci au détriment de la cohérence. L'opposition de ces deux propriétés a été présentée par Eric Brewer dans ce qu'on appelle aujourd'hui le théorème de CAP [41]. Le théorème de CAP a été énoncé pour la première fois en tant que conjecture par le chercheur Eric Brewer. En 2002, deux chercheurs au MIT, Seth Gilbert et Nancy Lynch, ont formellement démontré la vérifiabilité de la conjecture de Brewer afin d'en faire un théorème établi [16].

Ce théorème énonce qu'une base de données d'un système distribué ne peut garantir les trois contraintes suivantes en même temps (Figure A.7) [41], à savoir :

- **Cohérence (Consistency**) : Tous les nœuds (serveurs) du système contiennent exactement les mêmes données à un instant donné, pour que chaque utilisateur ait la même copie de données, quel que soit le serveur répondant à leur demande.
- **Disponibilité (Availability**) : Garantie que toute requête reçoive une réponse. Le système répondra toujours à une demande récente.
- **Résistance au partitionnement (Partition Tolerence**) : Le système doit être en mesure de répondre de manière correcte à toutes requêtes dans toutes circonstances sauf en cas d'une panne générale du réseau. Dans le cas d'un partitionnement en sous-réseaux, chacun de ces sous-réseaux doit pouvoir fonctionner de manière autonome qui veut dire que le système continue à fonctionner dans son ensemble, même si une partie du réseau n'est pas opérationnelle, les serveurs individuels échouent ou ne peuvent pas être atteints.

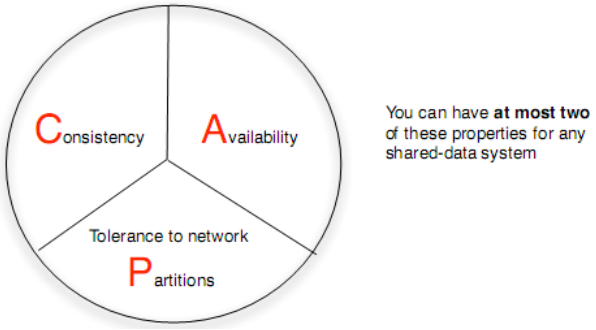


Figure A.7 : Théorème CAP de E. Brewer [41]

D'après le théorème de CAP, un système de calcul distribué ne peut garantir, à un instant donné, que deux de ces contraintes mais pas les trois, car elles se trouvent en opposition (Figure A.8).

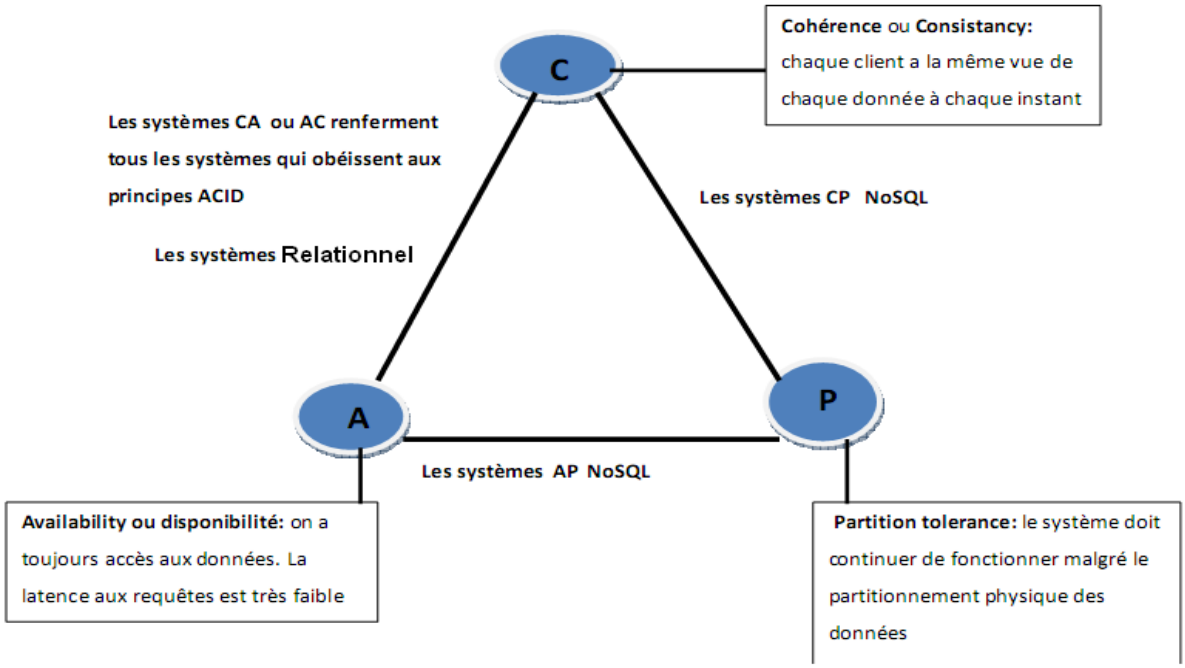


Figure A.8 : Types des systèmes suivant le théorème CAP

Par conséquent, seuls les systèmes CA, les systèmes CP et les systèmes AP sont possibles [16] :

- **CP** (cohérent et tolérant, mais pas très disponible) : Les données dans les nœuds sont cohérentes et le système possède une tolérance aux pannes, mais il peut aussi subir des problèmes de latence ou plus généralement, de disponibilité.
- **AP** (très disponible et tolérant, mais en sacrifiant la cohérence) : 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** (cohérent et hautement disponible, mais pas tolérant) : Les données dans les nœuds sont cohérentes tant que les nœuds sont en ligne. Toutes les lectures/écritures sur les nœuds, concernent les mêmes données. Mais si un problème de réseau apparait, certains nœuds seront désynchronisés au niveau des données et perdront donc de consistance.

Un système CA n'est pas vraiment cohérent car un système qui n'est pas tolérant sera, par définition, obligé d'abandonner la consistance ou la disponibilité lors d'un problème de partition. C'est pourquoi il existe une version plus moderne du théorème : « Durant un problème de partition, il faut choisir entre la disponibilité et la consistance ».

Un des premiers buts des systèmes NoSQL est de renforcer la scalabilité horizontale, il faut pour cela que le principe de tolérance au partitionnement soit respecté, ce qui exige l'abandon soit de la cohérence, soit de la haute disponibilité [16, 22]. Cela rejoint donc la version la plus récente du théorème où il faut choisir entre la disponibilité et la cohérence. Les axes principaux du NoSQL sont une haute disponibilité et un partitionnement horizontal des données, au détriment de la consistance contrairement aux systèmes classiques. Un SGBDR, pour répondre aux exigences de performance face aux grands volumes de données, doit se retourner vers du matériel de plus en plus rapide et à l'ajout de mémoire. Le NoSQL, pour sa part, pour gérer la « montée en charge » se réfère à la répartition de la charge sur les systèmes de Cloud Computing.

Contrairement aux SGBDR qui se considèrent comme des systèmes AC en assurant les propriétés de Cohérence et de Disponibilité, la plupart des bases de données NoSQL sont de type AP [41], c'est-à-dire qu'elles se concentrent plus sur la résistance au partitionnement, afin de garantir la disponibilité dans tous les temps et par conséquent abandonnent la cohérence.

A noter que le théorème de CAP a été critiqué par certains chercheurs. Nous citons par exemple la critique de D. Abadi qui souligne deux problèmes de CAP [42] :

– Suivant CAP, nous aurons soit des systèmes dits CA, CP ou AP. Il existe une asymétrie entre la valeur de la disponibilité et de la consistance : un système CP acceptera de sacrifier sa disponibilité mais seulement en cas de partition, tandis qu'un système AP abandonne définitivement sa consistance.
– Le second problème souligné par l'auteur est que, en pratique, la différence entre les systèmes CA et CP est difficile à discerner. En effet, qu'est ce que ça veut dire non tolérant à la partition ? Dans la pratique, cela signifie que, en cas de partition, le système perdra en disponibilité. Il n'existe donc en pratique que deux types de systèmes : CP/CA et AP.

L'auteur propose donc de remplacer CAP par PACELC qui pourrait être plus adapté à la durabilité [42, 43] :

*"if there is a Partition(**P**), how does the system trade off Availability and Consistency (**A** and **C**); Else (**E**) when the system is running normally in the absence of partitions, how does the system trade off Latency (**L**) and Consistency (**C**)?"*

En remplaçant C dans la définition par D (la capacité de tolérer les échecs de t nœuds sans perdre de données), et en définissant la disponibilité A par (au moins certains clients sont capables de faire des écritures), on obtient PADELD.

*"if there is a Partition (**P**), how does the system trade off Availability and Durability (**A** and **D**); Else (**E**), when the system is running normally in the absence of partitions, how does the system trade off Latency (**L**) and Durability (**D**)?"*

Notons que la disponibilité d'une base de données peut être mesurée par le nombre des 9, ces mesures cibles sont résumées dans la table A.2 :

Disponibilité en %	Arrêt annuel
90% (un neuf)	36.5 jours
99% (deux neufs)	3.65 jours
99.9% (trois neufs)	8.76 heures
99.99% (quatre neufs)	52.56 minutes
99.999% (cinq neufs)	5.26 minutes
99.9999% (six neufs)	31.5 secondes
99.99999% (sept neufs)	3.15 secondes

Table A.2 : Mesures par des 9 de disponibilité d'une base de données

II.2.7 Propriétés BASE

Il est important de rappeler que dans la plupart des cas, quand on fait référence aux principes ACID, on cherche surtout à faire en sorte de conserver l'intégrité des données. Autrement dit que ces données ne se perdent pas ou ne deviennent pas corrompues. Beaucoup de bases NoSQL peuvent fournir ce principe justement via le principe BASE [34].

Le principe BASE est donc plus facile à appliquer que l'intégralité des propriétés ACID. Prenons Amazon qui, afin de tenir la charge d'utilisateurs, a choisi d'occulter l'isolation de ses transactions. Par exemple, deux utilisateurs pourraient croire qu'ils ont acheté le dernier exemplaire d'un même livre. Amazon estime donc qu'il est plus judicieux de s'excuser envers un utilisateur que de tout ralentir et agacer une multitude d'autres utilisateurs.

Cette notion de BASE (Basically Availabe, Soft State, Eventuallty Consistent) est donc à l'opposé d'ACID. Les deux principes ne sont cependant pas mutuellement exclusifs. Ce nouveau principe est le fruit d'une réflexion menée par Eric Brewer fondée sur les limites du relationnel. Le principe BASE abandonne donc la consistance au profit de ces nouvelles propriétés :

- **Basically Available (Disponibilité basique)** : Même en cas de désynchronisation ou de panne d'un des nœuds du cluster, le système reste disponible dans le même sens que celle du le théorème CAP [32].
- **Soft-state (Cohérence légère)** : Spécifie que l'état du système risque de changer à mesure que le temps passe, et ce sans action utilisateur. C'est dû à la propriété suivante.
- **Eventual consistancy (Cohérence à terme)** : Indique que le système sera cohérent à mesure que le temps passe, à condition qu'il ne reçoive pas une nouvelle action utilisateur entre temps.

  En conclusion, comment déterminer lequel de ces principes est le plus adéquat pour un cas d'utilisation donné ?

  ACID est nécessaire si :
  - Beaucoup d'utilisateurs ou processus travaillant sur la même donnée en même temps ;
  - L'ordre des transactions est très important ;
  - L'affichage de données non actualisées n'est pas toléré ;
  - Il y a un impact significatif lorsqu'une transaction n'aboutit pas (comme par exemple dans des systèmes financiers en temps réel).

  BASE est préférable si :
  - Les utilisateurs ou processus ont surtout tendances à faire des mises à jour ou travailler sur leurs propres données ;
  - L'ordre des transactions n'est pas un problème (exemple d'Amazon plus haut) ;
  - L'utilisateur sera sur le même écran pendant un moment même s'il on lui affiche des données non actualisées ;
  - Aucun impact grave lors de l'abandon d'une transaction.

Notons enfin que les propriétés BASE conviennent très bien aux Modèles NoSQL.

II.2.8 Différents modèles NoSQL

Il existe une diversité de technologies et solutions toutes libellées en tant que NoSQL, se distinguant par leurs manières de représentation des données. Ces différents systèmes NoSQL utilisent des approches distinctes classées en quatre catégories [34, 44] :

- Orientées clé-valeur
- Orientées colonnes
- Orientées documents
- Orientées graphes

II.2.8.1 Bases de données clé-valeur (Key-value store)

Les bases de données associatives ou clé-valeur sont les formes les plus simples de bases NoSQL, qui se présentent comme de grosses tables de hachage. Il s'agit d'un système, sous forme de tableau associatif, qui stocke des valeurs indexées par des clés uniques, dont le principe est très simple : chaque valeur, nombre ou texte est associé à une clé unique, qui sera le seul moyen d'y accéder. Une entrée de tableau est un couple (Clé, Valeur) assez large en général Get(clé, valeur), Put(clé,valeur), Delete(clé). C'est uniquement par cette clé qu'il sera possible d'exécuter des requêtes sur la valeur. Ce modèle peut être comparable à un dictionnaire où chaque mot (clé) possède une ou plusieurs définitions (valeur). Du moment où le dictionnaire est ordonné (indexé), il n'est pas nécessaire de parcourir tout le dictionnaire pour retrouver un mot (Figure A.9). De même, le type NoSQL clé-valeur est indexé par clés qui pointent directement à leurs valeurs correspondantes [39].

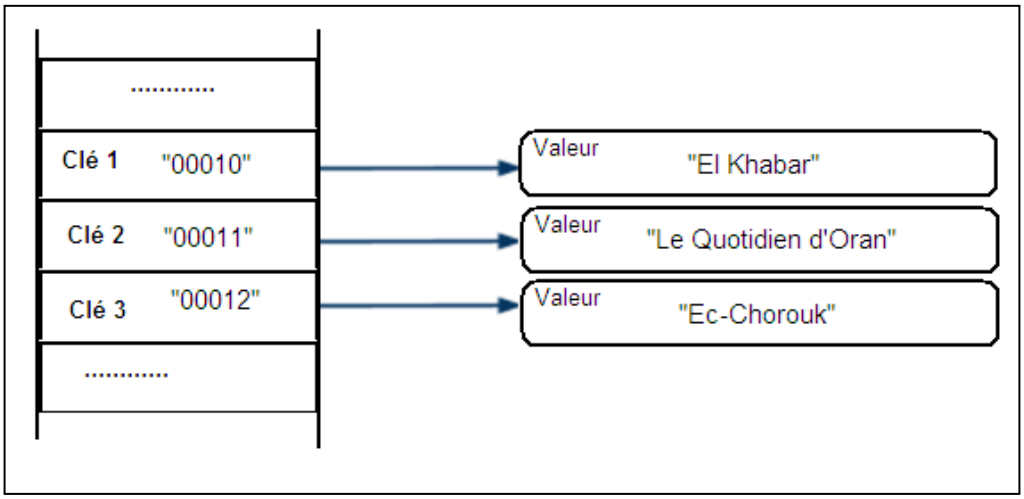


Figure A.9 : Illustration d'une base de données orientée clé-valeur

La structure de l'objet stocké est libre et donc à la charge du développeur de l'application. Les données, à la différence d'une table de hachage, sont répliquées. Un avantage considérable de

ce type est qu'il est très facilement extensible et donc facilite la scalabilité horizontale. Une simplicité du modèle est échangée par une bonne performance et surtout une scalabilité accrue. En effet dans le cas où le volume de données augmente, la charge est facilement répartissable entre les différents nœuds en redéfinissant tout simplement les intervalles des clés entre chaque nœud. Cette conception a permis de monter en charge à des péta-octets de données et des milliers de requêtes simultanées.

Ces systèmes sont donc principalement utilisés comme dépôts de données se limitant à des requêtes simples exécutées uniquement sur les clés. L'interaction avec la base de données se résume aux opérations basiques CRUD (Create, Read, Update, Delete). Les valeurs stockées sont totalement opaques pour la base et ne sont pas liées à un modèle de données prédéfini. Par conséquent il est impossible de filtrer ou demander une partie des données comme en SQL avec la clause « Where », si on ne connait pas les clés voulues.

En revanche pour une utilisation appropriée, cette architecture fournit des performances impressionnantes sur des opérations de lecture/écriture requérant de simples accès disques, et le meilleur scale-out du marché. Cela permet aussi de stocker, ensemble, des données totalement hétérogènes. On les retrouve comme système de stockage de cache ou de sessions distribuées, particulièrement là où l'intégrité des données est non significative.

Les solutions les plus connues ayant adoptées le système de couple clé-valeur sont [34, 39,45]: Redis, Memcached, Amazon DynamoDB, Riak, Voldemort Ehcache, Hazelcast (utilisé par le site LinkedIn), OrientDB, Berkeley DB, Oracle NoSQL, etc.

II.2.8.2 Bases de données orientées colonnes (Column family)

Les bases de données orientées colonnes sont grossièrement une évolution du modèle associatif. Elles constituent la poupe du mouvement NoSQL à cause de leur récurrence dans l'actualité avec des candidats comme HBase ou Cassandra [16]. Elles ont été conçues par les géants du web afin de faire face à la gestion et au traitement de gros volumes de données s'amplifiant rapidement.

De par leur structure, les bases orientées colonnes se rapprochent des bases relationnelles. Le principe d'une base de données colonnes consiste dans leur stockage par colonne et non par ligne. Contrairement d'une base de données orientée ligne classique ou les données sont stockées de façon à favoriser les lignes en regroupant toutes les colonnes d'une même ligne ensemble, les bases de données orientée colonnes quant à elles vont stocker les données de façon à ce que toutes les données d'une même colonne soient stockées ensemble. Ces bases peuvent évoluer avec le temps, que ce soit en nombre de lignes ou en nombre de colonnes. Ce

type de structure permet d'être plus évolutif et flexible ; cela vient du fait qu'on peut ajouter à tout moment une colonne. Autrement dit, et contrairement à une base de données relationnelle où les colonnes sont statiques, fixées dès la création du schéma de la table et ce nombre reste le même pour tous les enregistrements dans cette table, celles des bases orientées colonnes sont dites dynamiques et présentes donc uniquement en cas de nécessité, ce qui évite le stockage des valeurs Null [34, 39, 45]. Par exemple, l'enregistrement d'un étudiant nécessitant son nom, prénom et adresse. Dans le relationnel il faut au préalable créer le schéma (la table) qui permettra de stocker ces informations. Si un étudiant n'a pas d'adresse, la rigidité du modèle relationnel nécessite un marqueur Null, signifiant une absence de valeur qui coûtera toutefois de la place en mémoire. Dans une base NoSQL orientée colonne, la colonne adresse n'existera tout simplement pas. Les bases de données colonnes ont été conçues pour pouvoir stocker plusieurs millions de colonnes (Figure A.10).

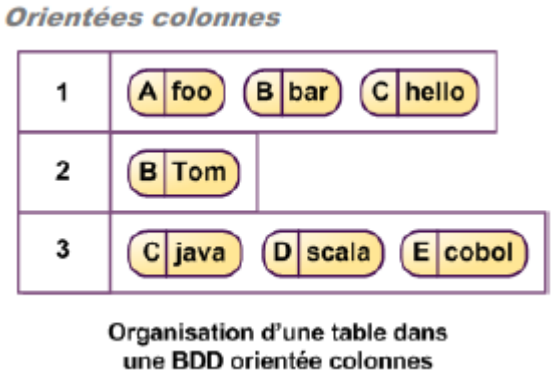


Figure A.10 : Base de données orientée colonnes [16]

Dans certaines bases orientées colonnes telles que Cassandra ou HBase, il existe quelques concepts supplémentaires tel que le regroupement logique de lignes (équivaut à une table dans le monde relationnel), et la notion de Super colonnes (nouvelle dimension supplémentaire de colonne contenant elle-même d'autres colonnes).

Les bases de données orientées colonnes comportent des avantages considérables. Les bases de données orientées colonnes sont prévues pour stocker des millions de colonnes, ce qui en fait des bases adaptées au stockage one-to-many. Leur capacité de stockage est accrue grâce à l'espace économisé sur les valeurs nulles, assurant également une rapidité remarquable d'accès aux données pour parcourir les lignes d'une table. Concrètement cela signifie que, contrairement aux bases orientées lignes, les utilisateurs qui souhaitent des informations précises n'auront pas à s'encombrer des informations inutiles des autres colonnes. En effet les bases orientées lignes se doivent de lire entièrement les lignes avant de sélectionner les colonnes désirées par la requête.

Cependant, l'utilisation de ce type de bases n'est réellement efficace que dans un contexte Big Data, pour les grands volumes de données homogènes et de même type.
Les principales solutions NoSQL orientées colonnes sont [34, 39, 45] : Cassandra (Apache), HBase (Apache), Bigtable (Google), Accumulo (Apache), Hypertable, etc.

II.2.8.3 Bases de données orientées documents (Document store)

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

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

Dans ces modèles, les clés ne sont plus associées à des valeurs sous forme de bloc binaire mais à un document de format non imposé [33, 44].
Les bases de données documentaires sont constituées de collections de documents (Figure A.11). Un document est composé d'un ensemble de champs et des valeurs associéesquipeuvent être requêtées. Bien que les documents soient structurés, ce type de base est appelée « Schema-Less » puisqu'il n'est pas nécessaire de définir au préalable les champs d'un document, ce qui permet de stocker dans la base des documents très hétérogènes. Cependant bien que ces documents ne doivent pas se conformer à un modèle, cela ne veut pas pour autant qu'il n'y ait aucune contrainte sur les données.
Le stockage structuré des documents leur confère des fonctionnalités dont ne disposent pas les bases clés-valeurs simples dont la plus évidente est la capacité à effectuer des requêtes sur le contenu des objets. La valeur, dans ce cas, est un document de type JSON, BSON ou XML. Cette caractéristique les rapproche donc des bases SQL.

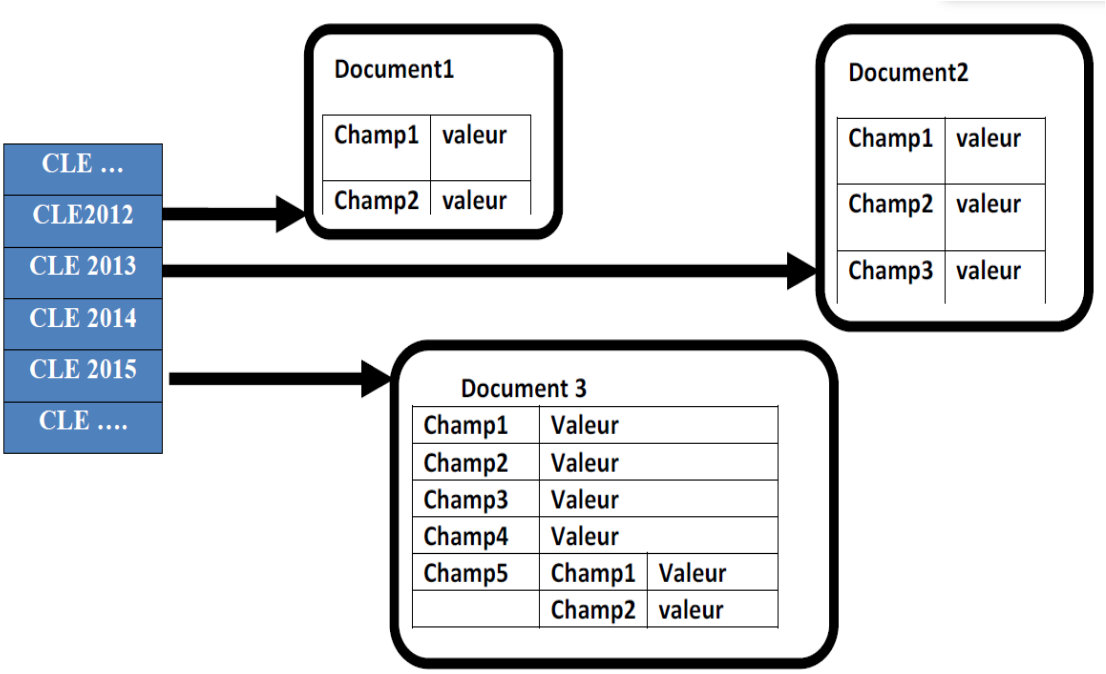


Figure A.11 : Base de données orientée documents

L'avantage considérable d'une base de données orientée document est de pouvoir récupérer un ensemble d'informations structurées hiérarchiquement depuis une clé. Une opération similaire dans le monde relationnel équivaudrait à plusieurs jointures de tables [46].

Par conséquent, ce type de bases est très convoité par les applications Web diffusant des pages entières obtenues par un ensemble de jointures. Ces applications peuvent également mettre en cache des informations sous une forme intelligible. Pour la majorité d'entre elles, la base peut être directement mise sur la mémoire RAM permettant un gain de performance considérable.

Un point négatif concernant ces bases de données document concerne la duplication des informations. Les données étant regroupées par document, il se pourrait que des problèmes d'intégrité surgissent, certaines données étant nécessairement dupliquées sur plusieurs documents. D'autre part, la très grande flexibilité de leur schéma est à priori un atout et en même temps une insuffisance, puisqu'il est facile de commettre une erreur lors de l'insertion de données. Les bases de données relationnelles offrent une plus grande garantie de cohérence des données, cardans un cas pareil, le serveur refusera d'exécuter la requête si la nouvelle donnée ne respecte pas le schéma de la base de données [46].

Les principales bases de données NoSQL orientées documents sont [34, 39, 45]: MongoDB (10gen), CouchDB, CoucHBase, Amazon DynamoDB, MarkLogic, RavenDB, Cloudant, OrientDB, GemFire, RethinkDB, Datameer, Microsoft Azure DocumentDB, ArangoDB, PouchDB, etc.

II.2.8.4 Bases de données orientées graphes (Graph store)

Les bases de données orientées graphes sont des systèmes bien différents des autres bases de données. Bien que les bases de données de type clé-valeur, colonne, ou document tirent leur principal avantage de la performance du traitement de données, les bases de données NoSQL orientées graphe n'ont pas pour but de résoudre des problèmes de performances mais plutôt de palier à des problèmes très complexes de liens de connectivité entre les données, qu'une base de données relationnelle serait incapable de le faire [33, 44]. Certains diront qu'il est possible d'exprimer de la complexité avec des bases relationnelles, et ils auront raison, à la différence près que les relations des graphes sont définies au niveau des enregistrements de la base et non au niveau du modèle de données. Le stockage de relations au niveau des données n'a de sens que si ces relations sont toutes très différentes, sinon cela revient à dupliquer toujours la même chose [16].

Les réseaux sociaux comme Facebook et Twitter, où des millions d'utilisateurs sont reliés de différentes manières, constituent un bon exemple : amis, fans, famille, etc. Dans ce contexte, le défi n'est pas le nombre d'éléments à gérer, mais le nombre de relations qu'il peut y avoir entre tous ces éléments. En effet, il y a potentiellement N^2 relations à stocker pour N éléments. L'approche par graphes devient donc inévitable pour des applications comme les réseaux sociaux [32]. Elles permettent également l'élaboration de liens entre les divers intérêts que pourraient avoir un internaute, afin de pouvoir lui proposer des produits susceptibles de l'intéresser. Ainsi, les pubs s'affichant sur Facebook sont très souvent en relation avec les recherches effectuées sur Google, et les propositions d'achats de sites de vente en ligne tels que Ebay et Amazon sont en relation avec des achats déjà effectués [39]. Chaque entité de cette base de données est appelée nœud, reliée entre elles par des arcs (les lignes) où les nœuds et les arcs ont des étiquettes. Le W3C a défini un standard appelé RDF (Resource Description Format) pour identifier les nœuds et les arcs du graphe [47].

Les bases orientées graphes ne représentent pas seulement l'aspect de scalabilité en volume (l'ordre de plusieurs milliards de nœuds et relations), mais aussi l'aspect de complexité des données en utilisant, comme leur nom l'indique, la théorie des graphes [16] qui s'appuient sur la notion de nœuds, de relations et de propriétés qui leur sont rattachées. Voici les trois éléments à retenir (Figure A.12) [46] :

- Un objet (dans le contexte de Facebook, c'est un utilisateur) sera appelé un nœud ;
- Deux objets peuvent être reliés entre eux par un arc (comme une relation d'amitié) ;
- Chaque objet peut avoir un certain nombre d'attributs (statut social, prénom, nom, etc.).

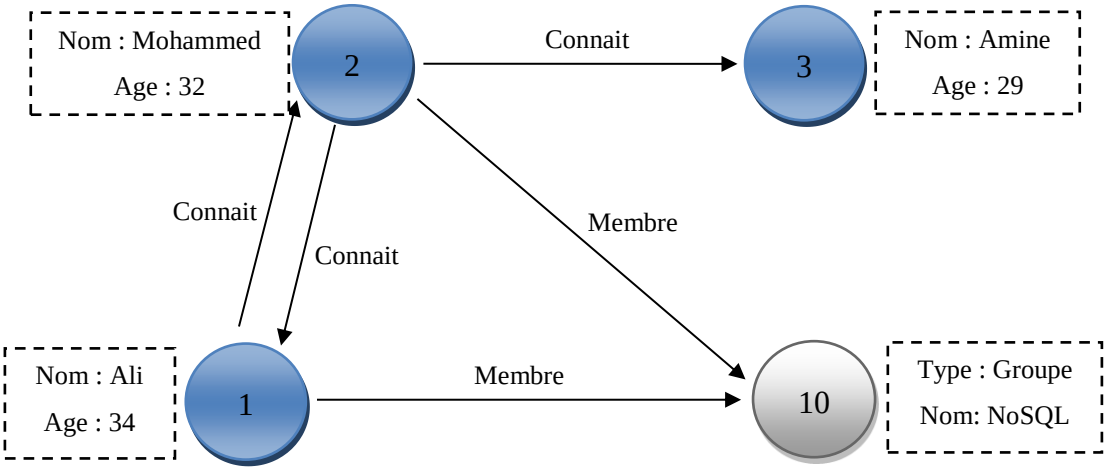


Figure A.12 : Exemple de base de données orientée Graphe

Les données sont donc stockées sur chaque nœud, lui-même organisé par des relations. A partir de là, il sera nettement plus aisé d'effectuer des opérations qui auraient été très complexes et lourdes dans un univers relationnel.

Le point positif de ce type de bases est qu'elles sont parfaitement adaptées à la gestiondes données relationnelles même dans un contexte Big Data. De plus, leurs architectures modulables peuvent être adaptées selon les besoins rencontrés. Certaines se basent sur des orientations colonne, d'autres sur l'enregistrement clé-valeur ou sur une combinaison d'entre elles. Les bases de données orientés graphes ne représenteront probablement pas la majeure partie des bases de données NoSQL.

Les solutions les plus connues ayant adoptées le NoSQL orienté graphe sont [34, 39, 45] : Neo4j, Orient DB, Titan, ArangoDB, Giraph, InfiniteGraph, Sqrrl, Sparksee, InfoGrid, HyperGraphDB, FlockDB, VelocityGraph, GlobalsDB, GraphDB, etc.

II.2.8.5 Bases de données Multi-modèles

Mise à part les quatre modèles reconnus, il y a d'autres solutions hybrides qui ont été conçues pour répondre à des besoins spécifiques comme [34, 39, 45] : Amazon DynamoDB, Microsoft Azure Cosmos DB, Datastax Enterprise, Ignite, ArangoDB, Apache Drill, Virtuoso, etc.

II.3 La contre-attaque du NewSQL

L'objectif majeur de cette nouvelle tendance est de chercher à regrouper les avantages du NoSQL et du SQL pour pouvoir exploiter les atouts du modèle relationnel à savoir ses

fondements mathématiques sur lesquels ils s'appuient, le langage de définition et manipulation de données SQL et les propriétés d'acidité.
Les bases de données NewSQL sont destinées aux entreprises qui gèrent des données sensibles et stratégiques. Elles requièrent une certaine évolutivité, mais aussi une cohérence supérieure à ce qu'apportent les bases NoSQL.

II.3.1 Définition

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 et fournir au modèle relationnel les avantages d'extensibilité horizontale et la tolérance aux pannes assurées par les solutions NoSQL. Elle profite des architectures distribuées et des évolutions sur le plan du matériel pour coller aux nouvelles tendances. Contrairement à NoSQL, le New SQL, très récent et très prometteur, permet de conserver le modèle relationnel au cœur de son système [16]. Certains auteurs le qualifient par SGBD Relationnel extensible « Scalable RDBMS ».

II.3.2 Architecture

Le NewSQL adopte une architecture distribuée, fournissant de meilleures performances par rapport aux solutions classiques de type SGBD Relationnel.
Cette architecture n'est pas totalement nouvelle, puisqu' 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 [48].

En fait, cette nouvelle architecture est née du croisement de 3 types d'architectures, relationnelle, non-relationnelle et grille de données appelée également « cache distribué », comme indiqué dans la figure A.13 [16]. 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.

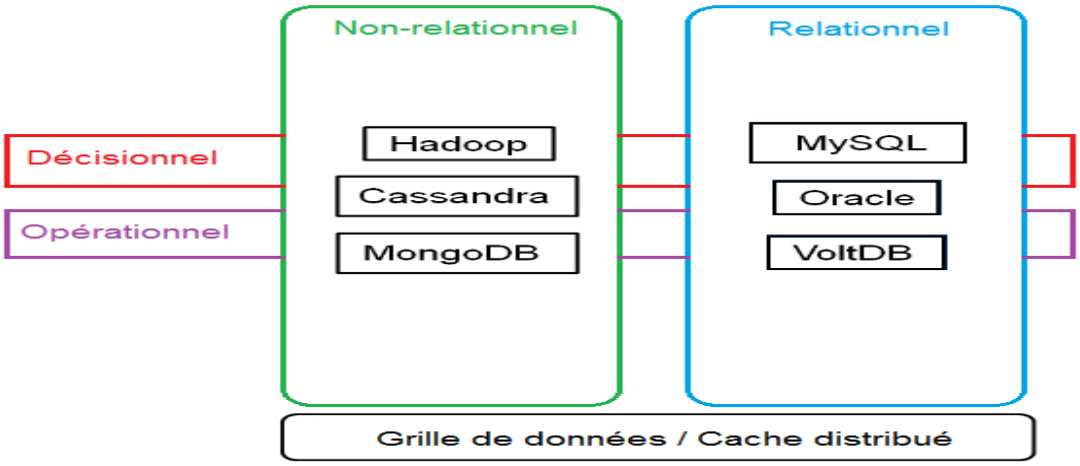


Figure A.13 : Naissance du NewSQL à partir de 3 architectures

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. Les solutions NewSQL se situent ainsi dans une position intermédiaire entre les solutions NoSQL et les grilles de données [49].

II.3.3 Caractéristiques

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 [50].

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.
- Scalabilité horizontale, architecture sans maître et capable de tourner sur un nombre important de nœuds sans affecter de goulot d'étranglement.
- Base de données en mémoire.

II.3.4 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 : VoltDB, Spanner, ClustrixDB, NuoDB, TransLattice Elastic Database, SQLFire, GridGain IMDB, Drizzle et MemSQL.

II.3.5 Avantages et inconvénients

Cette nouvelle génération vise donc d'y remédier aux différents problèmes rencontrés à la fois avec le NoSQL, mais aussi avec les bases relationnelles. L'ensemble des fonctionnalités et caractéristiques de ces modèles NewSQL, représentent des avantages, par rapport aux autres architectures :

- 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.

Cependant, ces nouvelles solutions souffrent de quelques inconvénients, citons principalement :

- Ces systèmes ne sont pas encore normalisés autant que les systèmes SQL traditionnels.
- Les architectures en mémoire peuvent être inappropriées pour des volumes dépassant quelques téraoctets et pour des machines moins puissantes.
- Offre un accès partiel aux outils riches des systèmes SQL traditionnels [51].

Chap III : Solutions NoSQL étudiées

Plusieurs architectures NoSQL ont été mises en œuvre par les grands acteurs du domaine comme Google, Yahoo, Facebook, Twitter, Amazon, etc., pour héberger dans leurs serveurs ces grands volumes de données. On dénombre actuellement dans le marché informatique plus de 305 solutions NoSQL, qu'elles soient de type :

- Base clé/valeur comme : Redis, Memcached, Voldemort, Riak, Hazelcast, Ehcache, Oracle NoSQL ;
- Base documentaire comme : MongoDB, Amazon DynamoDB, Couchbase, CouchDB, Microsoft Azure DocumentDB, Cloudant ;
- Base orientée colonne comme : Cassandra, HBase, Bigtable, Accumulo, Hypertable ;
- Base orientée graphe comme : Neo4j, Titan, Giraph, InfiniteGraph ;
- Base multi modèle comme : OrientDB, MarkLogic, ArangoDB, Virtuoso ;

La première partie de cette thèse fera l'objet d'une étude comparative entre plusieurs solutions, très populaires, de gestion de données NoSQL. Une étude comparative des différentes solutions existantes est très utile pour fournir, aux décideurs et acteurs intéressés, des critères, des indicateurs et des éléments d'informations sur leurs performances et différentes adéquations aux domaines d'utilisation, pour des éventuels choix et prise de décisions sur le modèle à adopter pour leurs projets. Ces nouveaux systèmes sont distingués à base de plusieurs critères : la performance, le modèle de données utilisé, la cohérence, les méthodes de stockage, les garanties de durabilité, la disponibilité, le support de requête, et d'autres dimensions. Ces systèmes sacrifient généralement quelques-unes de ces dimensions au profit d'autres.

Pour évaluer et comparer les solutions NoSQL disponibles, plusieurs benchmarks ont été conçus, le plus couramment utilisé est le YCSB de Yahoo. Plusieurs études comparatives dans le même contexte, utilisant YCSB, ont été développées récemment. L'étude avec laquelle on va comparer nos résultats a été menée dans une seule machine [1].

III.1 Classement de popularité des systèmes NoSQL

Des études très récentes montrent que le type NoSQL orientée documents, en tant que modèle, reste le plus utilisé, comme il est montré dans la figure A.14 [16]. La cause est probablement liée au fait qu'il soit simple à aborder (type JSON). D'un autre coté les orientées colonnes

sont au coude à coude avec les clé-valeurs. Les orientées graphes sont en dernière position puisqu'elles sont principalement réservées à des applications très spécifiques.

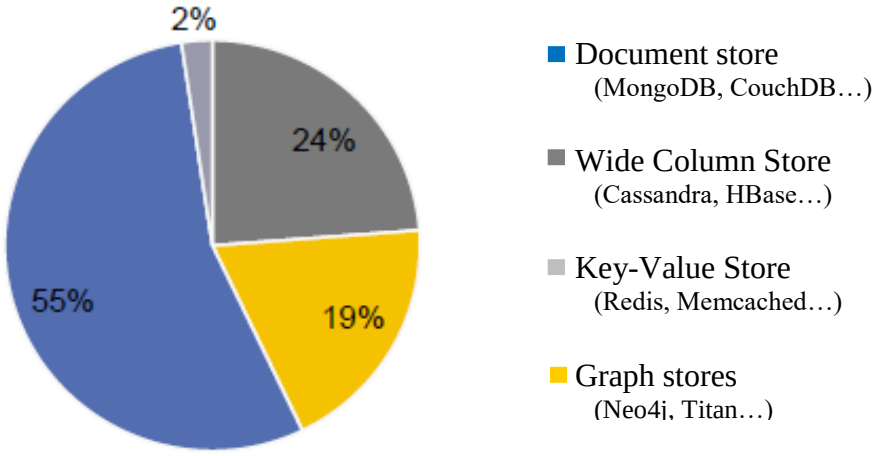


Figure A.14 : Types de NoSQL majoritairement utilisé [16]

Dans la section suivante, nous allons nous intéresser à la description proprement dite des bases NoSQL en ciblant des solutions très populaires, selon un classement de Solid IT [52]. Ce DB-Engines Ranking est une liste actualisée de 343 systèmes de gestion de bases de données classés par degré de popularité. Ils mesurent la popularité d'un système en utilisant les paramètres suivants :

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

Pour le mois de Juillet de l'année 2018, le classement des Top 50 est affiché dans le tableau A.3 suivant :

343 systems in ranking, July 2018

Rank Jul 2018	Rank Jun 2018	Rank Jul 2017	DBMS	Database Model	Score Jul 2018	Score Jun 2018	Score Jul 2017
1.	1.	1.	Oracle	Relational DBMS	1277.79	-33.47	-97.09
2.	2.	2.	MySQL	Relational DBMS	1196.07	-37.62	-153.04
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1053.41	-34.32	-172.59
4.	4.	4.	PostgreSQL	Relational DBMS	405.81	-4.86	+36.37
5.	5.	5.	MongoDB	Document store	350.33	+6.54	+17.56
6.	6.	6.	DB2	Relational DBMS	186.20	+0.56	-5.05
7.	7.	↑ 9.	Redis	Key-value store	139.91	+3.61	+18.40
8.	8.	↑ 10.	Elasticsearch	Search engine	136.22	+5.18	+20.25
9.	9.	↓ 7.	Microsoft Access	Relational DBMS	132.58	+1.59	+6.45
10.	10.	↓ 8.	Cassandra	Wide column store	121.06	+1.84	-3.07
11.	11.	11.	SQLite	Relational DBMS	115.28	+1.02	+1.41
12.	12.	12.	Teradata	Relational DBMS	78.22	+2.45	-0.14
13.	↑ 14.	↑ 16.	Splunk	Search engine	69.24	+3.46	+8.94
14.	↓ 13.	↑ 18.	MariaDB	Relational DBMS	67.51	+1.67	+13.15
15.	↑ 16.	↓ 13.	SAP Adaptive Server	Relational DBMS	62.12	+0.64	-4.79
16.	↓ 15.	↓ 14.	Solr	Search engine	61.52	-0.55	-4.51
17.	17.	↓ 15.	HBase	Wide column store	60.77	+1.07	-2.85
18.	18.	↑ 20.	Hive	Relational DBMS	57.63	+0.30	+11.42
19.	19.	↓ 17.	FileMaker	Relational DBMS	56.39	+0.21	-2.26
20.	20.	↓ 19.	SAP HANA	Relational DBMS	51.60	+2.25	+3.65
21.	21.	↑ 22.	Amazon DynamoDB	Multi-model	49.63	+3.84	+13.17
22.	22.	↓ 21.	Neo4j	Graph DBMS	41.88	-0.09	+3.36
23.	23.	↑ 24.	Memcached	Key-value store	33.88	+0.07	+5.35
24.	24.	↓ 23.	Couchbase	Document store	33.07	+0.62	+0.06
25.	↑ 26.	↑ 26.	Microsoft Azure SQL Database	Relational DBMS	26.84	+0.55	+4.55
26.	↓ 25.	↓ 25.	Informix	Relational DBMS	26.59	+0.03	-1.08
27.	27.	↑ 28.	Vertica	Relational DBMS	20.82	-0.34	-0.97
28.	28.	↑ 30.	Firebird	Relational DBMS	20.66	+0.27	+1.67
29.	29.	↓ 27.	CouchDB	Document store	19.50	-0.70	-2.65
30.	30.	↑ 41.	Microsoft Azure Cosmos DB	Multi-model	19.45	+0.25	+11.74
31.	31.	↓ 29.	Netezza	Relational DBMS	16.90	-0.20	-2.96
32.	32.	32.	Amazon Redshift	Relational DBMS	14.72	+0.35	+1.88
33.	33.	↑ 35.	Google BigQuery	Relational DBMS	13.34	+0.08	+2.04
34.	34.	↓ 31.	Impala	Relational DBMS	13.29	+0.26	-0.02
35.	35.	↑ 37.	Spark SQL	Relational DBMS	12.73	+0.81	+2.08
36.	36.	↑ 40.	InfluxDB	Time Series DBMS	11.70	+0.37	+3.59
37.	37.	↓ 33.	MarkLogic	Multi-model	11.31	+0.34	-1.12
38.	↑ 39.	↓ 34.	Greenplum	Relational DBMS	10.80	+0.19	-0.82
39.	↓ 38.	↓ 38.	dBASE	Relational DBMS	10.79	-0.01	+0.30
40.	40.	↓ 36.	Oracle Essbase	Relational DBMS	9.13	-0.06	-1.98
41.	41.	↓ 39.	Hazelcast	Key-value store	8.76	-0.33	-0.15
42.	↑ 44.	↑ 58.	Firebase Realtime Database	Document store	7.54	+0.91	+3.47
43.	↓ 42.		Datastax Enterprise	Multi-model	7.36	+0.10	
44.	↑ 48.		Microsoft Azure SQL Data Warehouse	Relational DBMS	7.09	+0.87	
45.	45.	45.	Sphinx	Search engine	6.80	+0.42	+0.37
46.	↓ 43.	↓ 44.	Ehcache	Key-value store	6.42	-0.35	-0.73
47.	↓ 46.	↓ 43.	Interbase	Relational DBMS	6.33	+0.06	-0.96
48.	↓ 47.	↓ 42.	Riak KV	Key-value store	6.28	+0.02	-1.08
49.	↑ 50.	↑ 60.	Realm	Relational DBMS	5.57	+0.23	+1.54
50.	↓ 49.	↓ 46.	OrientDB	Multi-model	5.39	+0.04	-0.18

Table A.3 : Top 50 des SGBD [52]

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

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

Système	Classement parmi les NoSQL	Classement général
MongoDB	1	5
Redis	2	7
Cassandra	3	10
HBase	4	17
Couchbase	8	24
OrientDB	17	50

Table A.4 : Classement des solutions étudiées

III.2 MongoDB

III.2.1 Description

MongoDB est une base de données orientée documents open-source, distribué sous licence AGPL (licence libre), fournissant de hautes performances, une haute disponibilité et une scalabilité automatique. MongoDB a été développé en C++ depuis 2007 par la compagnie 10gen qui travaillait sur un système de Cloud Computing à données largement réparties, semblable au service AppEngine de Google. La première version est apparue en 2009, mais c'était en 2010 que la version 1.4 a été considérée comme industriellement viable.

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

MongoDB permet de manipuler des objets structurés (documents) au format BSON (Binary JSON), un dérivé de JSON binaire plus axé sur la performance, qui a été pensé pour faciliter le scan des données. Fonctionnant comme une architecture distribuée-centralisée, il réplique les données sur plusieurs serveurs avec le principe de maître-esclave ou primaire/secondaire, permettant ainsi une plus grande tolérance aux pannes (failover)et la répartition de la charge de travail entre les nœuds (load balancing). L'architecture distribuée de MongoDB se base sur deux principes : la répartition des données (sharding) et la réplication (replica set). La répartition et la duplication de documents est faite de sorte que les documents les plus demandés soient sur le même serveur et que celui-ci soit dupliqué un nombre de fois suffisant. Par sa simplicité d'utilisation du point de vue de développement client, ainsi que ces performances remarquables, MongoDB est la base de données orientée document la plus utilisée [52].

III.2.2 Modèle de données

Le modèle de données de MongoDB est de type orienté documents. Un document, l'unité basique de MongoDB, est une structure de données composée de paires de champs et de valeur, et qui est identifié par son nom. Les valeurs des champs peuvent inclure d'autres documents, des tableaux et des tableaux de documents [53].

Les données prennent la forme de documents enregistrés dans des collections. Une collection contenant un nombre quelconque de documents. Les collections sont comparables aux tables, et les documents aux enregistrements dans les bases relationnelles. Afin d'augmenter les performances, tout en manipulant des documents, MongoDB utilise l'indexation similaire aux bases de données relationnelles [54].

Néanmoins et contrairement aux bases relationnelles, MongoDB est sans schéma prédéterminé. MongoDB ne pose aucune restriction quant aux documents contenus dans une même collection. A la différence d'une table SQL, le nombre de champs des documents d'une même collection peut varier d'un document à l'autre. Une base de données MongoDB est un conteneur de collections, tout comme une base de données SQL contenant des tables.

Par ailleurs, MongoDB ne permet pas les requêtes très complexes standardisées, mais il permet de programmer des requêtes spécifiques en JavaScript.

III.2.3 Architecture

MongoDB possèdequatre différents modes de fonctionnement [53] :

III.2.3.1 Single

Le mode Single (Figure A.15) : sert à faire fonctionner une base de données sur un seul serveur. Dans ce mode, un seul processus nommé *mongod* est utilisé pour traiter directement les données issues des requêtes du client [55].

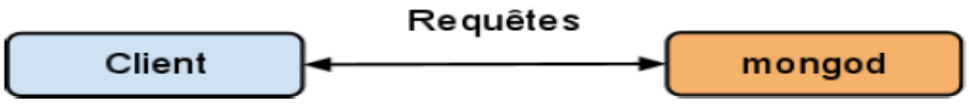


Figure A.15 : MongoDB en mode serveur seul

III.2.3.2 Replication Master / Slave

Dans ce mode Master/Slave (Figure A.16), le serveur fonctionne en tant que maître et s'occupe des demandes clients. En plus il s'occupe de répliquer les données sur le serveur esclave de façon asynchrone. L'esclave est présent pour prendre la place du maître si ce dernier tombe en

panne. Le premier avantage de cette structure est de garantir une forte cohérence des données, en centralisant les opérations clients sur le maître. Aucune opération n'est faite sur l'esclave, hormis les mises à jour envoyées par le maître vers l'esclave [55].

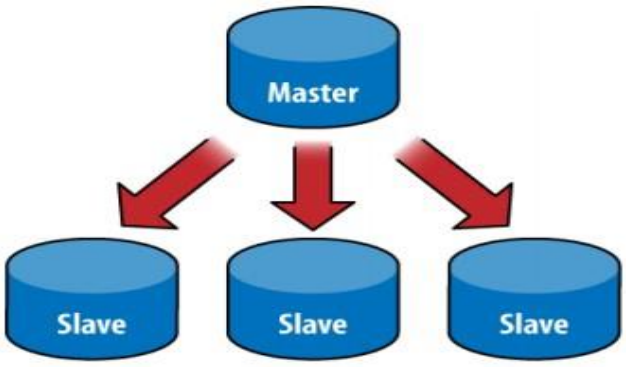


Figure A.16 : MongoDB Maître / Esclave

III.2.3.3 Replica Set

Il s'agit d'un mode de réplication maître / esclave plus avancé. Le Replica Sets fonctionne avec plusieurs nœuds possédants chacun la totalité des données (Figure A.17) : Ces différents nœuds vont alors élire un nœud primaire, qui peut s'apparenter à un maître. Pour qu'il soit élu, Il faut qu'un nœud obtienne la majorité absolue. Dans le cas où un nœud n'obtiendrait pas la majorité, le processus de vote recommencerait à zéro. De plus, une priorité peut être donnée à chaque nœud afin de lui donner plus de poids lors de l'élection. Un serveur arbitre peut-être insérer dans le système pour participer, seulement, aux élections afin de pouvoir garantir la majorité absolue. Le rôle de nœud primaire est vu comme celui du serveur maître dans le modèle maître / esclave. En cas de défaillance du nœud primaire, un nouveau nœud au sein du Replica Set est choisi pour devenir nœud primaire [55].

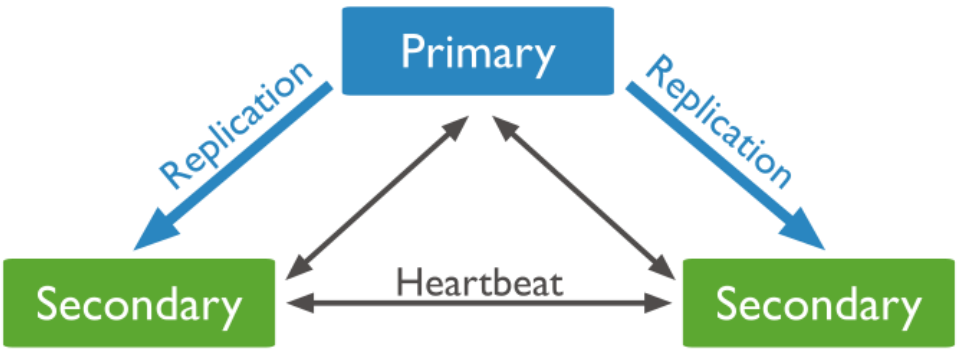


Figure A.17: MongoDB Replica Sets

III.2.3.4 Sharding

Le Sharding est une surcouche qui est basée sur du Master / Slave ou du Replica Sets (Figure A.18) : Le Sharding sert à partager les données entre plusieurs *Shards*, chaque *Shard* devant

stocker une partie des données. La définition de la manière dont les données seront découpées se fait grâce à une *Hash Shard Key* qui permet de définir sur quel critère seront partagées les données entre les *Shards* [10].

- ***Shards*** : Un ou plusieurs serveurs en mode Master / Slave ou Réplica Sets. Via son processus *mongod*, ils prennent encharge le stockage des données.
- ***Mongos*** : Des processus qui redirigent les requêtes des applications clientes vers les *shards* adéquats et regroupent les résultats avant de les transmettre à l'application cliente. Toute modification dans les serveurs de configuration est propagée immédiate mentaux processus *mongos*.
- ***Config Servers*** : Les serveurs de configurations stockent les métadonnées du système relatives à chaque *shard* et des informations sur le placement des données dans les serveurs respectifs (shards). Ce service est indispensable, il est donc nécessaire de l'assurer en lui consacrant deux, voire trois instances. L'écriture sur ces services utilise le protocole de validation en deux phases pour assurer un stockage fiable. En cas de défaillance d'un service, tous les autres serveurs passent en mode de lecture seule.

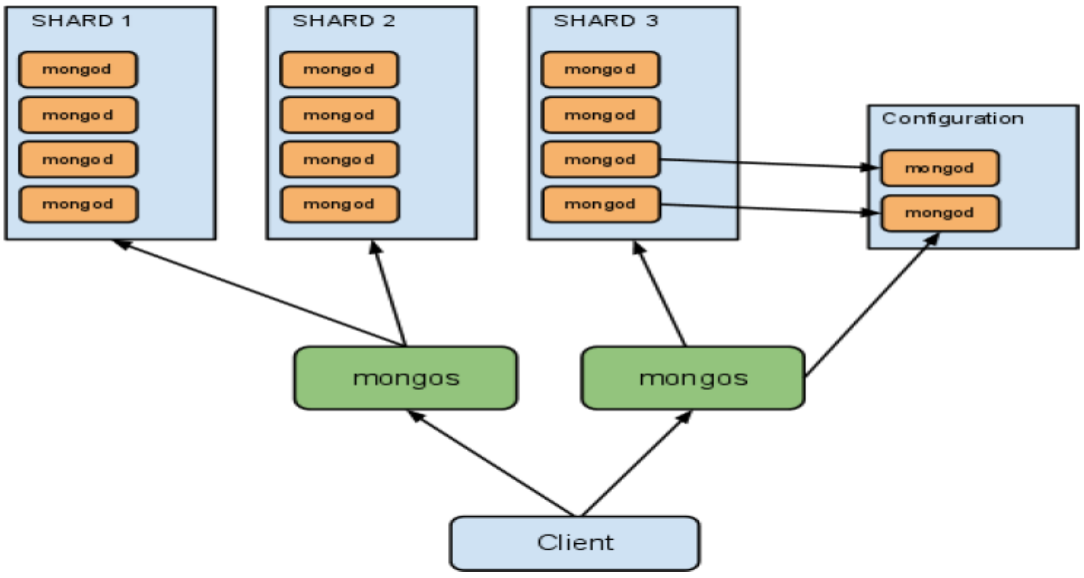


Figure A.18 : Partitionnement des donnés via Sharding

III.2.4 Manipulation des données

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

Cependant, MongoDB est livré aussi avec des liaisons pour les principaux langages deprogrammation : Perl, PHP, Python, Ruby, Scala, Go, C, C++, Dart, Erlang, Haskell, Java, JavaScript, et .NET (C# F#, PowerShell, etc). Les pilotes intégrés permettent de manipuler la base et ses données directement depuis ces langages.

MongoDB possède également un outil *mongo* en ligne de commande interactif qui permet d'accéder et manipuler directement la base de données.

Dans un exemple, d'une base de données *scolarite* contenant plusieurs collections, la commande qui permet d'afficher tous les documents de la collection *Etudiant*, est :

>use scolarite

>db.etudiant.find();

{ "_id": 2213, "Name": "Youcef", "Age" : "20"}

{ "_id": 7782, "Name": "Oussama", "Age": "19 "}

Le manuel d'administration ainsi que le site de MongoDB (*http://www.mongodb.org*) contiennent la documentation officielle en détail des mécanismes de manipulation de données par l'intermédiaire de l'outil *mongo*. En revanche, pour manipuler les bases MongoDB depuis un langage de programmation, il convient de se reporter à la documentation correspondante au pilote (driver) du langage en question.

Voici dans la table A.5, une liste non-exhaustive de commandes MongoDB :

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

Table A.5 : Commandes utiles de MongoDB

III.3 Cassandra

III.3.1 Description

Cassandra est une base de données NoSQL orientée colonne. Elle a été conçue pour pallier au problème de performances des bases de données relationnelles, ainsi qu'aux problèmes de gestion de grands volumes de données. Par ailleurs, elle pallie à la complexité de déploiement et au maintien de l'intégrité lors de déploiement en cluster et dans différents Datacenter. Cassandra est conçue pour gérer des quantités massives de données réparties sur plusieurs serveurs (Cluster), en assurant tout particulièrement une disponibilité maximale des données et en éliminant les points individuels de défaillance [46].
Initialement, elle étaitdéveloppée par Facebook, afin de répondre à des besoins concernant son service de messagerie, puis elle a été libéré en Open-source et a été adopté par d'autres grands acteurs du Web tel que Digg.com ou Twitter. Elle est aujourd'hui l'un des principaux projets de la fondation Apache, une organisation à but non lucratif développant des logiciels open-source.

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

III.3.2 Caractéristiques

Les principales caractéristiques de la base de données NoSQL Cassandra sont [46] :

III.3.2.1 Tolérance aux pannes

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

moins deux nœuds et un data center comme étant des clusters délocalisés. Cassandra permet d'assurer la réplication à travers différents data center [56].

III.3.2.2 Décentralisé

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

III.3.2.3 Modèle de données riche

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

III.3.2.4 Elastique

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

III.3.2.5 Haute disponibilité

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

III.3.3 Architecture

Cassandra est basée sur une architecture distribuée, tous les nœuds du Cluster ont le même rôle et permettent donc la lecture et l'écriture des données au sein de la base NoSQL. Le client peut alors se connecter à n'importe quel nœud pour accéder aux données. Il n'y a donc plus de SPOF (Single Point Of Failure). La communication entre les différents nœuds du Cluster se fait en « peer to peer ». Un utilisateur se connecte sur l'un des nœuds lors de l'ouverture de sa session sur le serveur, si le nœud sur lequel s'est connecté l'utilisateur, dispose de l'information que celui-ci recherche, le nœud répondra à la requête de l'utilisateur. Si toutefois ce n'est pas le cas, le nœud va communiquer avec ceux qui ont l'information pour ainsi restituer les données à l'utilisateur.

Cassandra reprend les concepts de 2 bases de données très réussies : La première est BigTable, créé par Google, pour son modèle de données orienté colonne et son mécanisme de persistance sur disque, et la seconde est Dynamo, créé par Amazon, pour son architecture distribuée sans nœud maître [28].

III.3.4 Modèle de données

Le modèle de données de Cassandra s'appuie sur un schéma dynamique, avec un modèle de données orienté colonne. Les colonnes et leurs métadonnées peuvent être ajoutées par l'application lorsque cela s'avère nécessaire.

Dans Cassandra, le Keyspace est le conteneur des données de l'application (un peu comme une base de données ou un schéma pour une base de données relationnelle) [57]. Dans ces Keyspaces se trouvent une ou plusieurs familles de colonnes (qui correspondent aux tables en base de données relationnelle). Ces familles de colonnes contiennent des colonnes ainsi qu'un ensemble de colonnes connexes qui sont identifiées par une clé de ligne. En outre, chaque ligne d'une famille de colonnes ne dispose pas nécessairement des mêmes colonnes qu'une autre ligne (Figure A.19). Enfin, Cassandra n'impose pas de relations entre les familles de colonnes au sens base de données relationnelles : il n'y a pas de clés étrangères et les jointures entre familles de colonnes ne sont pas supportées.

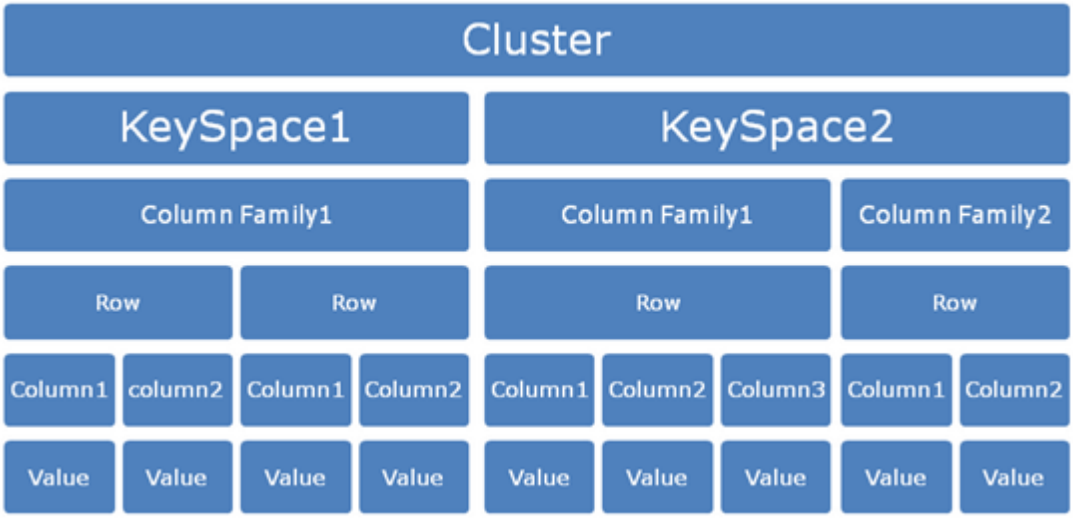


Figure A.19 : Structure d'un keyspace Cassandra

De plus, chaque ligne peut disposer d'un ensemble de colonnes différentes. Par contre, bien que les familles de colonnes puissent être flexibles, dans la pratique, il est conseillé d'y associer une sorte de schéma (à une famille de colonnes, il est préférable de n'y mettre qu'un même type de données). Pour toutes les familles de colonnes, chaque ligne est identifiée de manière unique par sa clé (un peu comme la notion de clé primaire pour les bases de données relationnelles).

En fait, dans Cassandra, une colonne est le plus petit élément de données. Elle est modélisée par un tuple qui contient le nom, la valeur et un timestamp (TableA.6). Ce timestamp est utilisé

par Cassandra pour déterminer la mise à jour la plus récente dans la colonne, et dans ce cas, c'est la plus récente qui est prioritaire lors d'une requête.

Coloumn_name
Value
Timestamp

Table A.6 : Structure d'une colonne Cassandra

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

III.3.5 Partitionnement des données dans un cluster Cassandra

Le partitionnement détermine la façon dont les données sont réparties à travers les nœuds du cluster (y compris les répliques). Fondamentalement, un partitionnement est une fonction de hachage pour calculer le jeton (c'est le hash) de la clé d'une ligne. Chaque ligne de données est identifiée de manière unique par une clé distribuée à travers le cluster par la valeur du jeton [58]. Cassandra emploie un ensemble de nœuds équivalents ou l'ajout d'un nouveau nœud est très simple puisqu'il suffit de connaître un seul nœud du cluster. Au final, les nœuds sont organisés sous la forme d'un anneau, en cas de défaillance de l'un des nœuds, il est simplement retiré de l'anneau (Figure A.20). L'anneau est divisé en plages égales au nombre de nœuds, chaque nœud étant responsable d'un ou plusieurs plages de données. Avant qu'un nœud puisse rejoindre l'anneau, on doit lui a assigné un jeton. La valeur de jeton détermine la position du nœud dans l'anneau et sa plage de données. Les données d'une table (column family) sont partitionnées sur les nœuds en se basant sur la valeur du hash de la clé de chaque ligne.

Figure A.20 : Ajout d'un nœud dans un cluster Cassandra

III.3.6 Réplication des données

La réplication est le processus permettant de stocker des copies des données sur de multiples nœuds afin de permettre leur fiabilité et la tolérance à la panne [59]. Quand un keyspace est créé dans Cassandra, il lui est affecté la stratégie de distribution des réplicas, c'est-à-dire le nombre de réplicas et la manière dont ils sont répliqués dans le cluster. La stratégie de réplication repose sur la configuration du Cluster afin de déterminer la localisation physique des nœuds ainsi que leur proximité par rapport aux autres. On fait souvent référence au facteur de réplication « RF : replication factor » pour indiquer le nombre total de réplicas par donnée dans le Cluster (Figure A.21).

- RF=1 signifie qu'il n'y a qu'une seule copie de chaque ligne ;
- RF= 2 signifie qu'il existe deux copies de chaque ligne ;
- RF=3 signifie qu'il existe trois copies de chaque ligne ;
- …..

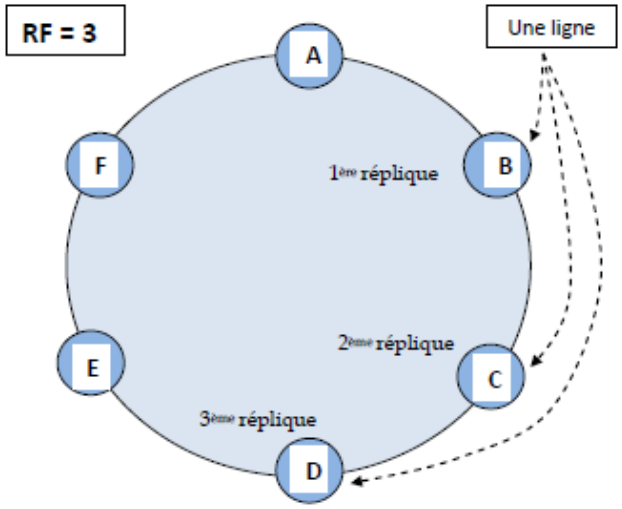


Figure A.21 : Réplication dans un Cluster Cassandra

Tous les réplicas possèdent la même importance : il n'y a pas de réplicas principaux ou maîtres du point de vue de la lecture et de l'écriture.

III.3.7 Cohérence des données

La question de cohérence dans Cassandra, s'applique à la façon dont les lignes de données sont synchronisées entre tous les réplicas. Le niveau de cohérence dans Cassandra peut être associé à n'importe quelle requête de lecture ou d'écriture. Cassandra étend le concept de cohérence éventuelle en offrant une cohérence réglable.

Les niveaux de cohérence disponibles pour une requête d'écriture sont les suivants :

Niveau	Description
ANY	Une écriture doit être effectuée au moins dans un nœud.
ONE	L'écriture doit être effectuée dans le "commit log" et "memory table" dans au moins une réplique.
QUORUM	L'écriture doit être effectuée dans le "commit log" et "memory table" dans le quorum des nœuds possédant les répliques.
LOCAL_QUORUM	L'écriture doit être effectuée dans le "commit log" et "memory table" dans le quorum dans le même Datacenter.
EACH_QUORUM	L'écriture doit être effectuée dans le "commit log" et "memory table" dans le quorum de nœuds pour chaque Datacenter.
ALL	L'écriture doit être effectuée dans le "commit log" et "memory table" dans le Cluster.

Table A.7 : Niveaux de cohérence disponible pour une opération d'écriture

En fait, pour être plus précis, un quorum est calculé de la manière ci-dessous [60] :
(Facteur de réplication/2) + 1

Ainsi, par exemple, si le facteur de réplication est de 3, alors le quorum est de 2, qui veut dire que l'on tolère qu'un seul réplica est indisponible. De même, si le facteur de réplication est de 6, alors le quorum des de 4, qui veut dire que 2 réplicas peuvent être indisponibles.
Lors d'une lecture, le niveau de consistance spécifie combien de réplicas doivent répondre avant que le résultat ne soit fourni au client. Cassandra contacte le nombre de réplicas indiqué afin d'obtenir la donnée la plus récente (en se basant sur le timestamp). Les niveaux de consistance disponibles sont les suivants :

Niveau	Description
ONE	Retourne une réponse de la réplique la plus proche.
QUORUM	Retourne l'enregistrement avec le timestamp le plus récent une fois le quorum des répliques a répondu.
LOCAL_QUORUM	Retourne l'enregistrement avec le timestamp le plus récent une fois le quorum des répliques dans le même Datacenter a répondu.

EACH_QUORUM	Retourne l'enregistrement avec le timestamp le plus récent une fois le quorum des répliques dans chaque Datacenter du cluster a répondu.
ALL	Retourne l'enregistrement avec le timestamp le plus récent une fois toutes les répliques dans le Cluster ont répondu. La lecture échoue si une réplique n'a pas pu répondre.

Table A.8 : Niveaux de cohérence disponible pour une opération de lecture

Concernant le choix du niveau de cohérence sur l'écriture et la lecture, cela dépend des besoins et des priorités. Ainsi, si la latence est une priorité, un niveau de cohérence à ONE peut être envisageable. Bien sûr, dans ce cas, il y a une forte probabilité que les données lues ne soient pas cohérentes. A l'inverse, si c'est l'écriture qui importe, alors une cohérence à un niveau ANY peut répondre au besoin. Dans ce cas, ce sont les données lues qui risquent de ne pas être cohérentes.

III.4 Redis

III.4.1 Description

Redis est une base NoSQL de type clé/valeur, acronyme de REmote DIctionary Server, écrit avec le langage de programmation C et distribué sous licence BSD. Il fait partie de la mouvance NoSQL ayant pour objectif fondamental d'avoir les performances les plus élevées possibles. Redis permet de manipuler des types de données simples : chaînes de caractères, tableaux associatifs, ensembles, ensembles ordonnés et listes. Après avoir connu une grosse croissance en 2010, Redis continue tranquillement son chemin avec l'ajout du « *sentinel failover* » qui s'occupe de monitorer, notifier et basculer automatiquement des instances en cas de problèmes. L'intégration du langage *Lua*, car tout le monde utilise du Javascript, permettra d'étendre encore les possibilités de cette base orientée clé-valeur. Actuellement elle est utilisée par des entreprises comme The Guardian, GitHub, Stack Overflow, Craigslist ou encore Blizzard Entertainment [55].

III.4.2 Stockage en mémoire vive

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

le risque de perte de données en cas d'incident. Pour y remédier, Redis offre la possibilité de sauvegarder l'état de la base dans un fichier. Cette technique ne permettant pas de garantir la conservation des opérations effectuées entre deux sauvegardes. La deuxième technique consiste à conserver une trace de toutes les opérations, et les restaurer en les ré-appliquant dans l'ordre, en cas d'incident. Voici les deux fonctions employées par Redis [55] :

- **RDB** : Consiste à copier toute la base de données se trouvant dans la RAM, dans un fichier sur le disque sous forme de backups complets périodiques. Cependant, cela n'est pas adapté pour enregistrer des transactions entre deux copies, sauf si on va procéder à un enregistrement de la base pour sauver la transaction désirée [61].
- **AOF** : fonctionne à la manière des "redo-log d'Oracle". C'est un journal dans lequel les transactions sont conservées dans des temps répétés et déterminés. Il permet à la base de garder une trace de toutes les manipulations de données, ainsi en cas de crash de la base, il permet de rejouer les modifications apportées à la base.

III.4.3 Architecture

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

III.4.3.1 Maître / Esclave

Redis se base sur une architecture maître / esclave comme beaucoup de bases NoSQL. Elle permet aussi d'évoluer dans un environnement de type Sharding, où les données sont découpées et partagées entre plusieurs serveurs. Dans un environnement maître / esclave, seul le maître s'occupera des requêtes en écriture des clients. Les esclaves quant à eux peuvent s'occuper de répondre aux clients afin d'alléger le serveur maître, des requêtes clients. Redis éprouve un inconvénient majeur : si une panne venait à survenir sur le serveur maître, aucun esclave ne viendrait le remplacer automatiquement donc il n'existe pas de mécanisme automatique de récupération. Pour y remédier à ce type de panne, l'administrateur procède manuellement pour désigner un nouveau maître parmi les esclaves qui restent toujours disponibles en lecture, afin de pouvoir continuer à répondre aux clients (Figure A.22) [55].

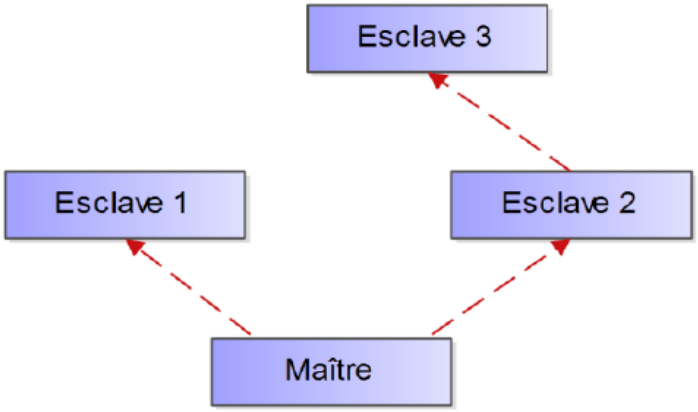


Figure A.22 : Redis Maître / Esclave

III.4.3.2 Sentinel

Redis propose un service indépendant pour palier au problème de récupération en cas de panne. Pour cela, en plus des divers serveurs s'occupant de la base de données, des serveurs Redis Sentinel ont été conçus, récemment, pour servir et gérer le cluster des serveurs (Figure A.23). Redis Sentinel propose donc trois types de services :

- **Monitoring** : Connaître l'état des différentes instances de Redis.
- **Notification** : Sentinel peut notifier à l'administrateur ou à un ordinateur le dysfonctionnement d'une instance.
- **Automatic Failover** : Dans le cas où un serveur maître échoue, Sentinel s'occupe de le remplacer en promouvant un serveur esclave [55].

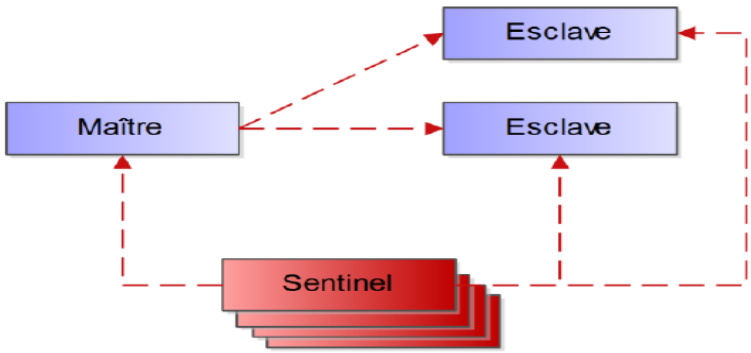


Figure A.23 : Redis Sentinel

III.4.3.3 Cluster

Pour configurer un cluster, Redis procède à la répartition des données grâce à la technique du Consistent Hashing. Cette technique permet de répartir aisément les données entre les différents nœuds et facilite le rajout de nouveaux serveurs. En outre, chaque nœud s'occupant

d'une partie des données, définie par le hash, peut avoir autant d'esclaves que nécessaire. En cas de dysfonctionnement du maître, les esclaves éliront un nouveau maître pour prendre sa place. Durant cet intermède d'indisponibilité, les autres nœuds du cluster possédant les autres données ne serviront pas les clients afin de garantir l'intégrité des données (Figure A.24) [55].

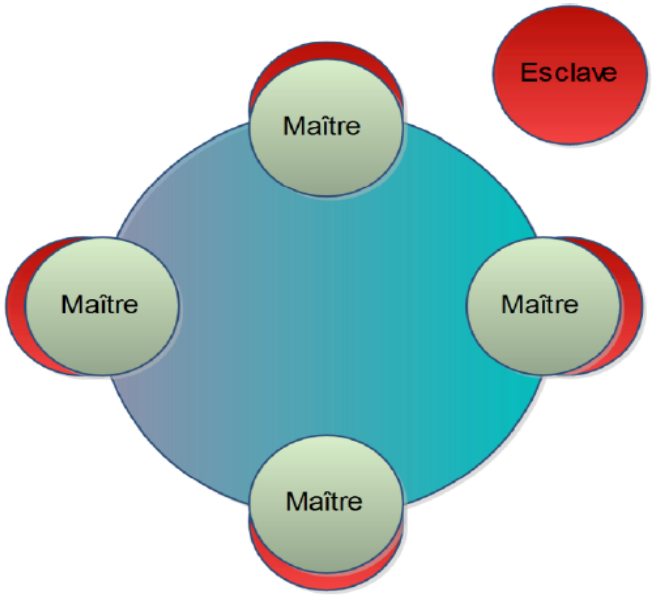


Figure A.24 : Redis Cluster

III.4.3.4 Réplication

Chaque fois qu'une transaction est effectuée sur le maître, ce dernier la reproduit directement sur les différents esclaves qui lui sont liés, ce qui permet de les maintenir à jour. A leur tour, ces derniers vont ensuite la répercuter sur les esclaves liés à eux s'il y en a [55].

III.5 HBase

III.5.1 Description

HBase est une base de données NoSQL orientée colonnes, open source, écrit en Java, inspirée des publications de Google sur BigTable [62], disposant d'un stockage structuré dans les environnements Big Data.

HBase est un sous-projet d'Hadoop sous licenceApache, développé dans le cadre du projet Apache Hadoop d'Apache Software Foundation. La société américaine Cloudera distribue une édition d'Hadoop (voir partie B - section II.2.1) et HBase dans le support Cloudera Enterprise. L'objectif majeur de ce projet est l'hébergement de très grandes tables des milliards de lignes et des millions de colonnes. HBase s'articule sur le système de gestion de fichiers distribué HDFS d'Hadoop (Hadoop Distributed File System) (voir partie B - section II.2.4),

qui est caractérisé par une haute tolérance aux pannes et un parallélisme massif. HBase permet de gérer les accès aléatoires read/write pour des applications de type temps réel en conservant un grand nombre de fichiers ouverts en même temps [62].

Facebook a annoncé en 2010 qu'il allait désormais utiliser HBase en remplacement de Cassandra, pour stocker tous les messages échangés dans le réseau social, ce qui représentait plus de 150 téraoctets de nouvelles données par mois [63, 64].

III.5.2 Modèle de données

HBase est conçu sur un modèle orienté colonnes basé sur six concepts [63, 64] (Figure A.25):

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

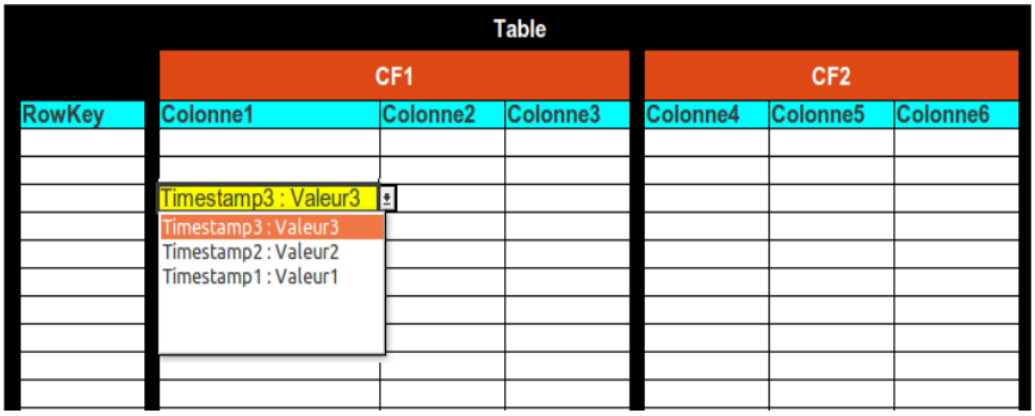


Figure A.25 : Vue synthétique du modèle de données HBase

III.5.3 Architecture

HBase est un système de gestion de base de données NoSQL distribué sur un ensemble de machines (cluster), fondée sur une architecture maître/esclave HDFS et un ensemble

d'algorithmes qui permettent une lecture/écriture aléatoire. Contrairement au SGBD orienté ligne (SGBD Relationnel), les données sont stockées sous forme de colonne.

La scalabilité, est basée sur le principe de la répartition de charge. Chaque partition est une Région qui stocke d'une manière contigüe, un ensemble de lignes triées selon la rowkey. Chaque Région est hébergée par un serveur physique [63, 64] (Figure A.26).

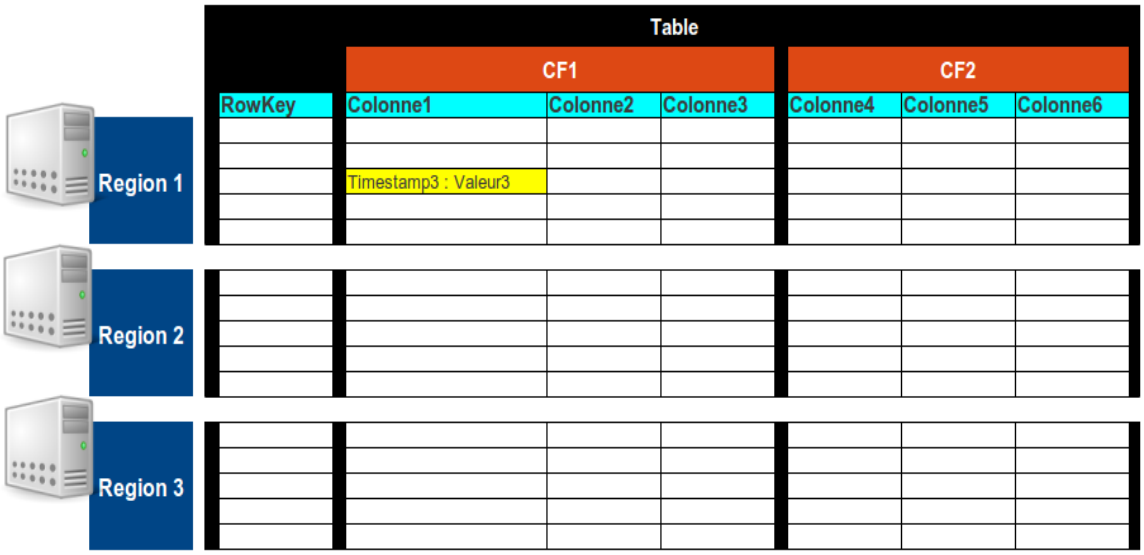


Figure A.26 : Répartition des données dans HBase

La figure ci-dessous présente le fonctionnement général d'HBase. Une base de données est un ensemble de tables ou chaque table est constituée d'un ensemble de familles de colonnes (regroupements logiques de colonnes). Sur chaque famille de colonnes il est possible d'ajouter des colonnes. Les lignes des tables sont partitionnées en plusieurs régions. Lors de la création d'une table, une seule région est créée, elle est ensuite automatiquement divisée en sous-parties lorsque sa taille atteint un certain seuil. Le client HBase envoie directement une requête de type « *get* » au Region Server qui contient l'information souhaitée.

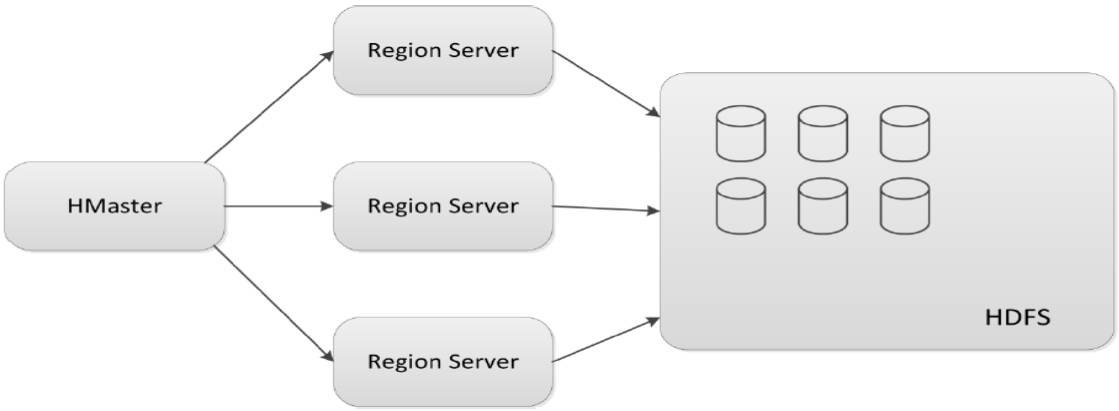


Figure A.27 : Architecture fonctionnelle d'HBase

L'architecture adoptée par HBase, s'appuyant sur un certain nombre de composants, est illustrée dans la Figure A.28 :

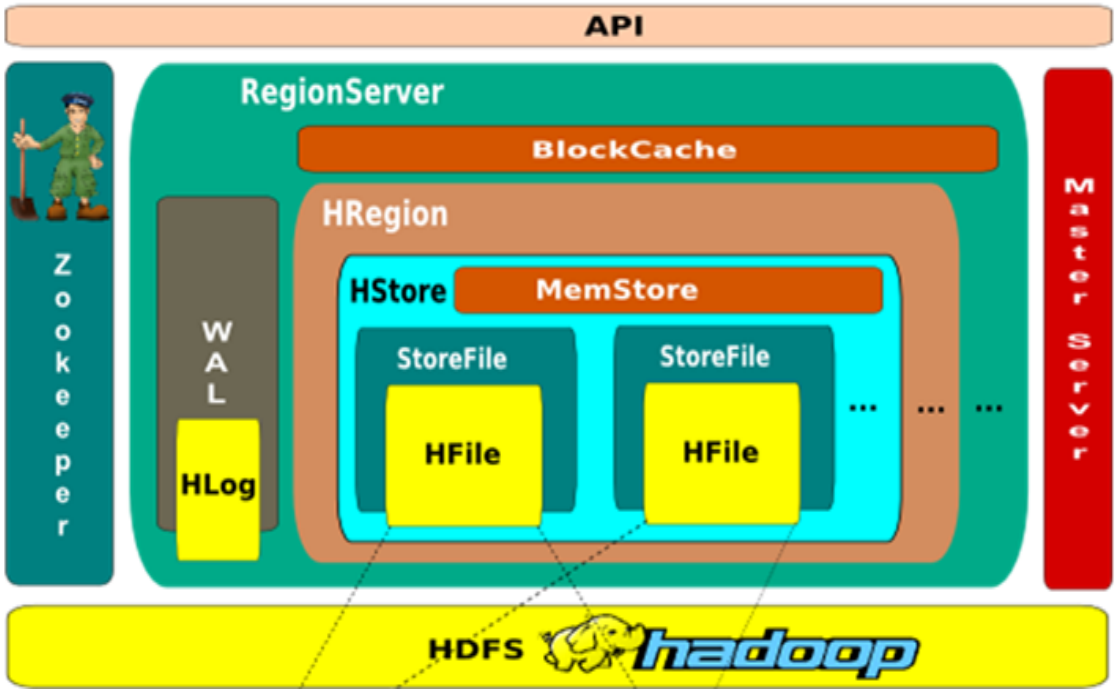


Figure A.28 : Anatomie de la base de données HBase [63]

- **Master Server** : Le composant maître appelé « HMaster », contient le schéma des tables de données. Il prend en charge la gestion, la coordination et le contrôle de toutes les instances « Region Server » du cluster.
- **RegionServer** : Les nœuds esclaves appelés « Region Server » gèrent des sous-ensembles de tables, appelés « Region ». Ces nœuds HBase sont les points d'entrée pour accéder aux données, et stockent les données des tables dans les nœuds de stockage d'HDFS « DataNode ».
- **Region** : HRegion est l'élément de base dans le processus de stockage et distribution des données dans HBase. Chaque Region gère un sous-ensemble d'une table HBase (une partition).
- **BlockCache** : Le BlockCache est activé par défaut pour toutes les tables, qui charge toute opération de lecture.
- **Write Ahead Log** : Chaque opération d'ajout ou mise à jour dans un Region Server est systématiquement écrite dans WAL en premier lieu. Avant d'être répliquer dans le MemStore. Ce mécanisme garantit la durabilité de la donnée en cas de défaillance du

serveur. Le processus WAL écrit dans le fichier Hlog qui est placé dans HDFS. Pour chaque instance Region Server il y a une instance de WAL.

- **Zookeeper :** Zookeeper est un outil permettant de coordonner des services distribués, c'est lui qui permettra de faire le lien entre le master et le système de fichier distribué HDFS. Il surveille l'état du cluster et informe régulièrement le Master Server des différents états. De plus, il stocke les informations critiques du cluster, dans la table système ROOT, qui contient la liste des tables META et leurs emplacements. La table META contient la liste des régions et leurs emplacements.

III.6 CouchBase

III.6.1 Description

Depuis sa naissance en 2011 via la fusion de Membase et CouchOne, CouchBase est devenu un acteur important du milieu NoSQL avec son produit CouchBase Server qui a fait un grand pas en avant [16]. Couchbase Server appartient à la classe des bases de données NoSQL orientée document conçu pour les applications web interactives. Elle dispose d'un modèle de données flexible, facilement extensible offrant une haute performance constante, capable de servir des données d'application avec 100% de disponibilité. C'est une base de données persistante qui exploite une couche de mise en cache RAM intégrée [65].

Les dernières fonctionnalités de CouchBase sont les suivantes [16] :

- Passage d'une orientation clé-valeur à une orientation documents : Support natif du JSON qui permet ainsi d'obtenir des documents avec des différentes structures.
- Amélioration des requêtes et indexation : Les documents peuvent être indexés via des vues. Tous les index sont automatiquement distribués à travers les nœuds d'un cluster. Les requêtes supportent dorénavant des recherches par clé, par filtre et les agrégats. Les requêtes sont également traitées pendant des changements de topologie (failover etc.)
- Réplication XDCR (cross data-centers) : Réplication des données sur plusieurs clusters répartis dans plusieurs data-centers afin d'anticiper des catastrophes naturelles ou encore pour rapprocher les données des utilisateurs finaux pour une meilleure performance.

III.6.2 Architecture

Couchbase Server comprend un seul paquet qui est installé sur tous les nœuds. En utilisant les SDKs (également connu sous le nom des bibliothèques de connectivité client à puce), on peut

écrire des applications dans différents langages (Java, node.js, .NET ou autres). Les applications se connectent à un cluster Couchbase Server pour effectuer des opérations de lecture / écriture et exécuter des requêtes avec de faibles latences et haut débit [66].

La caractéristique principale de Couchbase est qu'il est « chema-less » ou sans schéma. Il n'y a aucun schéma fixe prédéfini pour stocker les données. En outre, il n'y a aucune jointure faite sur les données (documents). Il permet le stockage distribué en utilisant des ressources informatiques, tels que le CPU et la RAM, s'étendant à travers les nœuds qui font partie du cluster Couchbase.

Les clusters de Couchbase se composent de plusieurs nœuds ou chaque cluster est une collection d'une ou plusieurs instances du serveur Couchbase qui sont configurés comme un cluster logique [67].

Contrairement à la plupart des technologies des clusters travaillant sur les relations maître/esclave, Couchbase s'appuie sur le mécanisme peer-to-peer. Autrement dit, qu'il n'y a pas de différence entre les nœuds du cluster qui possède les mêmes fonctionnalités. Il n'y a donc aucun point de défaillance unique : lorsqu'il y a une défaillance d'un nœud, un autre nœud prend ses responsabilités, fournissant ainsi une haute disponibilité.

III.6.3 Data Manager (Gestionnaire de données)

Toute opération effectuée sur la base de données Couchbase est stockée dans la mémoire, qui agit comme une couche de mise en cache. Par défaut, chaque document est stocké dans la mémoire pour chaque lecture, insertion, mis à jour, et ainsi de suite jusqu'à ce que la mémoire soit pleine. Toutefois, afin d'assurer la persistance de l'enregistrement, il y a une file d'attente du disque. Ceci purgera l'enregistrement sur le disque de manière asynchrone, sans impact sur la demande du client. Cette fonctionnalité est fournie automatiquement par le gestionnaire de données, sans aucune intervention humaine [67].

III.6.4 Data Cluster Management (Gestion du cluster)

Le gestionnaire de cluster est le responsable de l'administration des nœuds ou chaque nœud d'un cluster Couchbase inclut le composant de cluster manager et data manager. Il gère la récupération et le stockage de données. Couchbase clients utilisent la carte cluster fournie par le gestionnaire de cluster pour savoir quel nœud contient les données requises et communique ensuite avec le gestionnaire de données pour effectuer des opérations de base de données sur ce nœud [67].

III.6.5 Buckets (Seaux)

Pour stocker un document dans un cluster Couchbase, il faut d'abord créer un seau comme un espace de noms logique. Précisément, un seau est un conteneur virtuel indépendant, qui regroupe les documents logiquement dans un cluster Couchbase, ce qui équivaut à un espace de noms de base de données SGBDR. Il peut être consulté par divers clients dans une application. C'est possible également de configurer des fonctionnalités telles que la sécurité, la réplication et ainsi de suite par seau. En interne, Couchbase s'organise pour stocker des documents dans différents entrepôts pour différents seaux. Des informations telles que les statistiques d'exécution sont recueillies et rapportées par le cluster Couchbase, regroupés par type de seau [67].

Couchbase fournit deux types de seaux, qui se différencient par leur mécanisme de stockage et leurs capacités :

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

IV.6.6 Views (Les vues)

Une vue est un indice persistant de documents dans une base de données, elle permet l'indexation et l'interrogation des données ainsi que l'extraction des champs de documents JSON sans l'ID de document. Les vues sont écrites sous la forme de MapReduce. Couchbase implémente MapReduce en utilisant le langage JavaScript [67].

III.7 OrientDB

III.7.1 Description

OrientDB est une base de données NoSQL open source, écrit en Java, qui peut être déployée sur toutes les plateformes. Ce modèle est entièrement transactionnel, prenant en charge les transactions ACID garantissant le traitement fiable de toutes les transactions de la base de

données. En conséquence, tous les documents en attente, en cas d'incident, sont récupérés et engagés.

Il est possible de l'exécuter en mode distribué, en mode intégré ou en mémoire, très utile pour le développement, les essais ou petites applications autonomes [28].

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

III.7.2 Modèle orienté document

En tant que modèle orienté document, OrientDB ajoute également le concept d'un " *LINK* " comme une relation entre les documents. Lors de la récupération d'un document, tous les liens sont automatiquement générés par OrientDB. Ceci est une différence majeure par rapport à d'autres bases de données orientées document, comme MongoDB ou CouchDB, où le développeur doit gérer toutes les relations entre les documents [68].

III.7.3 Modèle orienté graphe

Cette architecture représente une structure en forme de réseau constituée de sommets reliés entre eux par des bords (arcs). Le modèle de graphe d'OrientDB est représenté par le concept d'un graphe de propriétés, définissant les éléments suivants [68] :

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

III.7.4 Modèle orienté clé / valeur

C'est le modèle le plus simpliste des trois précédents. Les données sont accessibles par une clé, où les valeurs peuvent être de types simples et complexes. OrientDB supportent les documents et les éléments graphe comme des valeurs permettant un modèle plus riche, par rapport au modèle clé / valeur classique [68].

Chap IV : Etude comparative

Les différents modèles NoSQL existants dans le marché, adoptent différentes architectures puisqu'ils étaient conçus, développés et déployés dans différents secteurs pour répondre à des besoins divergents. Le manque de standardisation est un aspect important du mouvement NoSQL et la panoplie de solutions existantes sur le marché de l'informatique, imposent aux décideurs, qui ont choisi le NoSQL, de nombreux problèmes dans le choix du modèle adéquat par rapport à leur environnement d'exploitation.

Dans ce contexte, notre travail tente de fournir quelques réponses pour choisir le système NoSQL approprié au type de données utilisées et le type de traitement exécuté sur ces données. Notre étude consiste à développer une étude comparative sur les performances de six solutions très répandues dans le marché de l'IT, à savoir MongoDB, Cassandra, Redis, HBase, CouchBase et OrientDB à l'aide de l'outil YCSB. Ce dernier étant un outil très connu pour sa puissance de test et utilisé dans plusieurs travaux d'évaluation des bases de données NoSQL. Plusieurs études comparatives, dans le même domaine, à base d'YCSB, ont été récemment réalisées. L'étude à laquelle nos résultats seront comparés, a été réalisée dans une seule machine, et est présentée dans l'article [1].

On tient à signaler que notre étude n'a pas inclus les bases de données orientées graphes, du fait que celles-ci ne doivent pas être évalués selon les mêmes scénarios utilisés dans l'analyse des autres types de bases de données NoSQL (orientées colonne, orientées document et clé/valeur), puisque l'utilisation des liens entre les enregistrements nécessitent une approche différente basée sur d'autres critères pour évaluer les performances des bases de données de graphes. Pour comparer ces modèles, d'autres outils ont été développés, tel que, XGDBench.

Après la description du benchmark utilisé et les différentes charges de travail employées, ce chapitre sera consacré, à la présentation et l'analyse des résultats obtenus des différentes expérimentations, afin d'évaluer les performances des différents modèles de bases de données par rapport à la nature des opérations effectuées sur ces bases.

Notre étude a été menée dans une machine physique avec la configuration suivante : Ubuntu 14.10, PC 64-bit, Core i5 et 6 Go de RAM. Les expériences réalisées en tant que scénarios de tests (Benchmarks) sont exposées et comparées aux résultats obtenus par [1] dans une machine virtuelle avec Ubuntu 32 bits, 2 Go de RAM installés sur un PC avec OS Windows 7 32 bits et 4 Go RAM.

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

Système	Version
YCSB	YCSB 0.8.0
MongoDB	Version 2.6.11
CouchBase	Version 2.5.2
Cassandra	Version 2.2.5
HBase	Version 0.94.8
Redis	Version 3.0.2
OrientDB	Version 2.1.3

Table A.9: Versions des systèmes et outils employés

Les processus de déploiement et configuration des différents outils, composants logiciels, les pré-requis nécessaires, le benchmark YCSB et les solutions NoSQL en question seront reportés dans la section des annexes.

IV.1 Benchmark utilisé et charges de travail

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

YCSB a été annoncé dans un article de Yahoo! dans lequel ils ont présenté des résultats de comparaison pour quatre systèmes largement utilisés : Apache HBase, Apache Cassandra, Yahoo! PNUTS et une implémentation MySQL distribué, à base de leurs caractéristiques de performance et d'élasticité [69]. Après une synthèse faite sur différents benchmarks [34], l'auteur suggère le YCSB, et affirme que probablement et actuellement, le meilleur outil de comparaison des bases NoSQL est YCSB. Le benchmark mesure les performances brutes, affiche les caractéristiques de latence au fur et à mesure qu'il y a une montée en charge d'un serveur, mesure l'extensibilité, et montre comment les systèmes comparés s'adaptent rapidement à un éventuel passage à l'échelle [34]. Il est devenu un outil de référence standard pour les systèmes NoSQL. Cet outil est multiplateforme et prend en charge la majorité des systèmes NoSQL et s'adapte facilement à ces solutions. Il est souvent utilisé pour comparer

les performances relatives des systèmes NoSQL dans le Cloud, ceci est confirmé par la multitude des études qui l'ont utilisé [1, 34, 54, 70, 71, 72, 73,74]. YCSB se compose de deux composants : un générateur de données et un ensemble de tests de performance pour évaluer les opérations d'insertion, de mise à jour et de suppression des enregistrements. Chaque test est une charge de travail, ou on peut configurer le nombre d'enregistrements à charger, le nombre d'opérations à exécuter et la proportion de lecture et d'écriture. Les opérations prises en charge comprennent : l'insertion, la mise à jour (modifier un des champs), la lecture (un champ aléatoire ou tous les champs d'un enregistrement) ainsi que le scan (lire les enregistrements dans l'ordre du démarrage à partir d'une clé d'un enregistrement sélectionné au hasard).

Chaque charge de travail (Workload) utilise des paramètres de référence différents, qui peut être modifiée et personnalisée en fonction du type de résultats attendus. Voici un exemple d'une charge de travail de mise à jour lourde : 50% de lectures - 50% de mises à jour.

```
# Yahoo! Cloud System Benchmark
# Workload A: Update heavy workload
#   Application example: Session store recording recent actions
#
#   Read/update ratio: 50/50
#   Default data size: 1 KB records (10 fields, 100 bytes each,
plus key)
#   Request distribution: zipfian
recordcount=1000
operationcount=1000
workload=com.yahoo.ycsb.workloads.CoreWorkload
readallfields=true
readproportion=0.5
updateproportion=0.5
scanproportion=0
insertproportion=0
requestdistribution=zipfian
```

Nous avons commencé par initialiser l'outil YCSB avec 6 charges de travail. Chaque test a démarré avec une base de données vide. Une fois les données chargées, les charges de travail (décrites ci-dessous) sont effectuées sur les six bases de données. Après l'exécution de chaque workload, on effectue des contrôles de santé de la base de données.

Les objectifs principaux des tests étaient les suivants :

1. Choisir des charges de travail typiques des applications modernes actuelles.
2. Utiliser des volumes de données représentatifs de données "Big Data".
3. Variez les proportions des charges de travail en lecture / écriture pour tester la performance des six solutions.

4. Conserver les mêmes noms de charges de travail avec les mêmes taux utilisés dans les travaux de Veronika et Al. [1] pour une meilleure similitude des résultats et une comparaison plus claire.

La liste des charges de travail, fournis dans la partie des annexes, inclut :

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

Cependant, afin de mieux nous concentrer sur les opérations de mise à jour, nous avons adopté deux nouvelles charges de travail supplémentaires proposées aussi par [1] pour remplacer l'exécution des charges de travail D et E contenant des opérations d'insertion. Notre principal intérêt est l'évaluation des opérations de lecture, d'écriture et de chargement des données. Néanmoins, les charges de travail D et E sont reportées à la fin et seront présentés à titre indicatif.

Les nouvelles charges de travail proposées sont les suivantes :

- **Workload G** (Update mostly 5% Read - 95% Update) : La charge de travail « mise à jour principalement » contient majoritairement des opérations de mise à jour 95% et 5% d'opérations de lecture.
- **Workload H** (Update Only 100% Update) : Le test « mise à jour seulement » est composée de 100% d'opérations de mise à jour.

YCSB fournit les résultats dans un format de texte assez structuré pour chaque charge de travail qui résume le débit global d'opérations/seconde, la latence moyenne et le temps d'exécution du test. Les temps d'exécution choisis comme indicateurs dans notre étude ont été recueillis et comparés. Notons aussi que pour minimiser l'effet de variations des performances CPU et E/S, nous avons exécuté le même test sur trois jours différents pour ne conserver que le temps moyen d'exécution obtenu.

IV.2 Résultats expérimentaux

Dans cette section, nous allons exposer les résultats de chargement de 600 000 enregistrements générés avec YCSB, dont chacun est constitué de 10 champs de 100 octets générés de manière aléatoire sur la clé d'identification de registre, qui donne à peu près 1 kb par enregistrement. Chaque enregistrement est identifié par une clé composée par la chaîne "user" suivi de plusieurs chiffres, par exemple "user379904308120", qui est la clé d'enregistrement. Les champs de l'enregistrement sont identifiés par Field0, Field1, ..., Fieldi respectivement. Les valeurs dans chaque champ sont des chaînes aléatoires de caractères ASCII, 100 octets chacun.

Nous présentons également le temps d'exécution des charges de travail obtenues lors des opérations de lecture, d'écriture et de mise à jour. Toutes les Workloads exécutent 1 000 opérations et mesurent le temps moyen d'exécution, ce qui signifie qu'il y avait à chaque fois 1 000 requêtes test interrogeant la base de données. A chaque fois, les résultats sont comparés avec [1]. Une évaluation globale de l'exécution de toutes les charges de travail (A-B-C-F-G-H) sera effectuée par la suite. Avant de conclure par une synthèse et analyse de nos résultats expérimentaux, on va comparer les bases de données, sur deux aspects distincts à savoir sur les opérations de lecture et sur les opérations d'écriture séparément.

IV.2.1 Chargement des données (LoadProcess)

Avant d'exécuter les différentes workloads, on va procéder au chargement des données en indiquant le nombre d'enregistrement à charger :

- **MongoDB**

La ligne de commande est la suivante :

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

Le résultat de chargement obtenu à partir du terminal est le suivant :

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

Le journal de chargement généré par YCSB pour MongoDB est obtenu dans un fichier texte « mongoload.txt » :

mongoload.txt

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

– **CouchBase**

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

– **Cassandra**

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

– **HBase**

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

– **Redis**

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

– **OrientDb**

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

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

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

Table A.10 : Temps de chargement

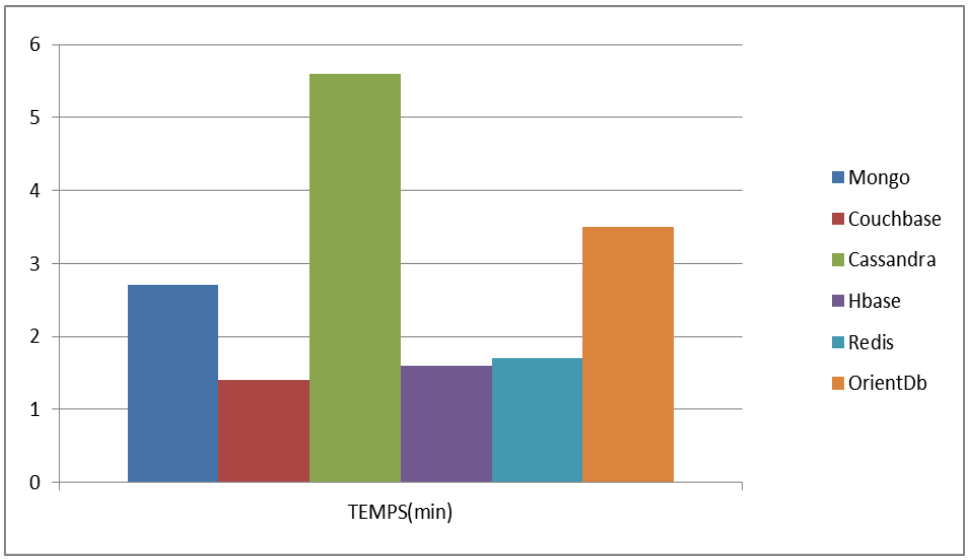


Figure A.29 : Temps de chargement

La Figure A.29 affiche les temps expirés pour l'opération de chargement de 600 mille enregistrements pour chacune des bases de données testées.

Les bases de données orientées document (CouchBase et MongoDB) ont été plus performantes en moyenne. Durant l'opération de chargement, le meilleur temps a été obtenu par CouchBase avec seulement 1 minute et 4 secondes contre 2 minutes et 7 secondes par MongoDB, ce qui signifie que la première était deux fois plus rapide que la deuxième.

HBase a également prouvé son efficacité pour les opérations de chargement initial avec seulement 1 minute et 6 secondes, contrairement à Cassandra qui était loin derrière avec 5 minutes et 6 secondes.

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

Pour le loading des 600 mille enregistrements, les deux bases de données orientées colonne ont été moins performantes, en moyenne, par rapports aux premières.

IV.2.2 Workload A (50% Read - 50% Update)

- **MongoDB**

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

mongorunwka1.txt

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

- **CouchBase**

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

- **Cassandra**

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

- **HBase**

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

- **Redis**

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

– **OrientDb**

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

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

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

Table A.10 : Temps d'exécution du Workload A

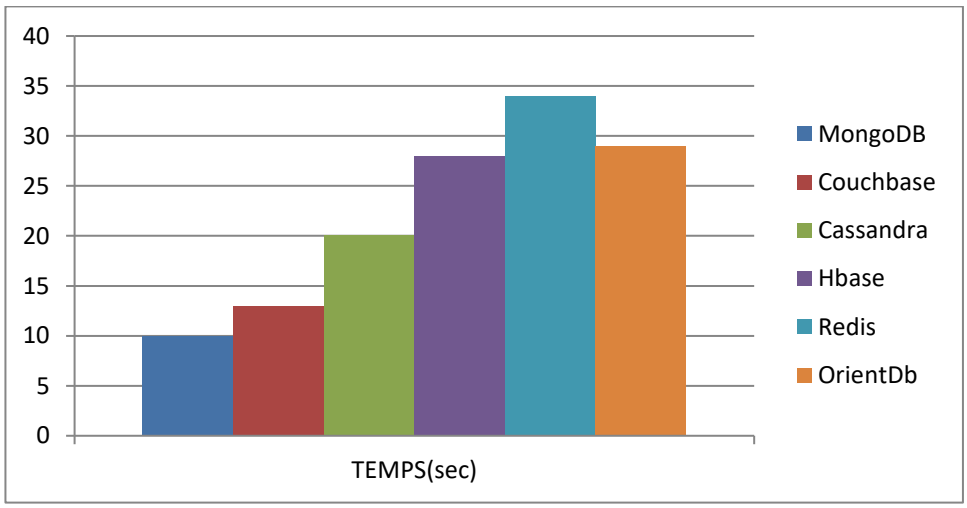


Figure A.30 : Temps d'exécution du Workload A

La Figure A.30 affiche les résultats obtenus après exécution du Workload A, composé de 50% d'opérations Read et 50% Update de 1000 opérations, effectuées sur 600000 enregistrements. Après lecture des résultats obtenus, nous remarquons que les bonnes performances sont présentées en premier lieu par la catégorie orientée document où MongoDB était la plus rapide suivie par CouchBase. En deuxième lieu on retrouve la catégorie orientée colonne, Cassandra avec 20 secondes et HBase avec 28 secondes. Les bases de données clé-valeur sont les moins performantes relativement aux précédentes.

Néanmoins, pour favoriser une certaine solution pour les opérations Read ou Update ou les deux en même temps, il faut aussi se reporter aux autres résultats et en particulier ceux des charges C (100%Read) et H (100% Update).

IV.2.3 Workload B (95% Read, 5% Update)

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

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

Table A.11 : Temps d'exécution du Workload B

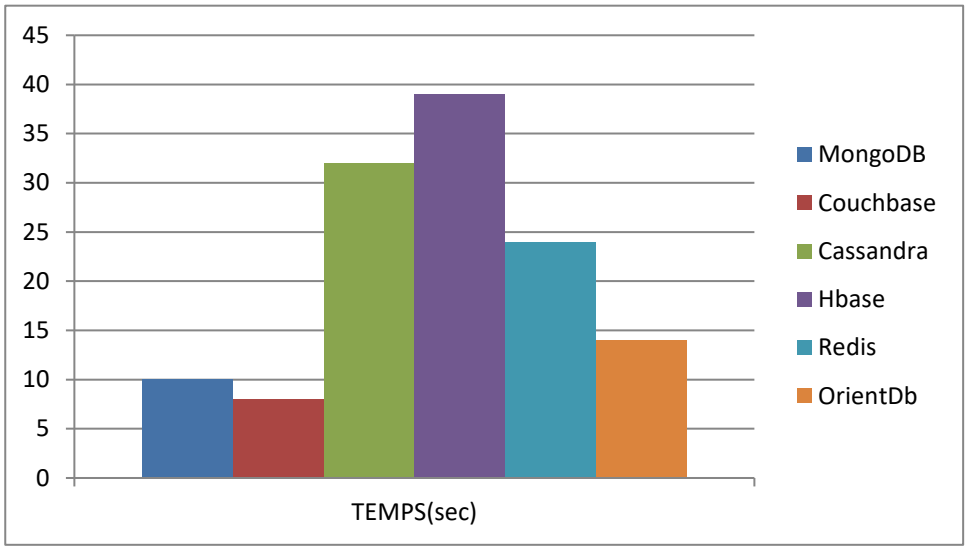


Figure A.31 : Temps d'exécution du Workload B

Les résultats de la Figure A.31 prouvent l'ascendance une autre fois des modèles orientés documents pour les charges composées principalement d'opérations de lecture. En revanche, ceux des clé-valeur ont prouvé leur performance par rapport aux modèles orientés colonnes. Rappelons aussi que CouchBase s'est imposé une autre fois sur les autres avec une durée de 8 secondes.

IV.2.4 Workload C (100% Read)

Le tableau et la figure, qui suivent, présentent le temps d'exécution moyen de la charge de travail C, composée uniquement d'opérations de lecture, pour chacune des bases :

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

Table A.12 : Temps d'exécution du Workload C

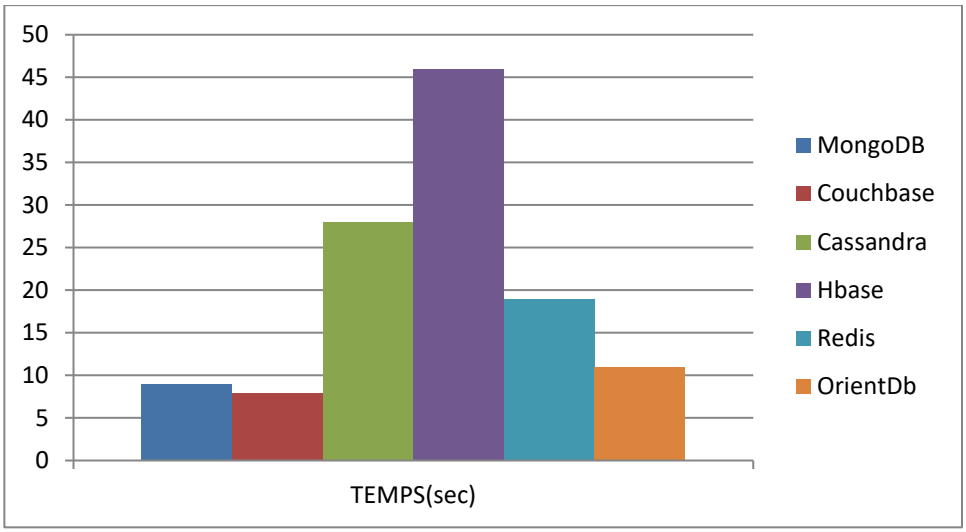


Figure A.32 : Temps d'exécution du Workload C

Pour des opérations purement lecture, les résultats obtenus confirment le classement précédent des bases lors d'exécution de la charge B. La première position pour les orientées documents, la deuxième position pour les clé-valeur et la troisième position pour les orientées colonnes.

IV.2.5 Workload F (50% Read, 50% Read-Modify-Write)

Les temps d'exécutions moyens en secondes de la charge de travail F, sont récapitulés dans le tableau et la figure ci-dessous :

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

Table A.13 : Temps d'exécution du Workload F

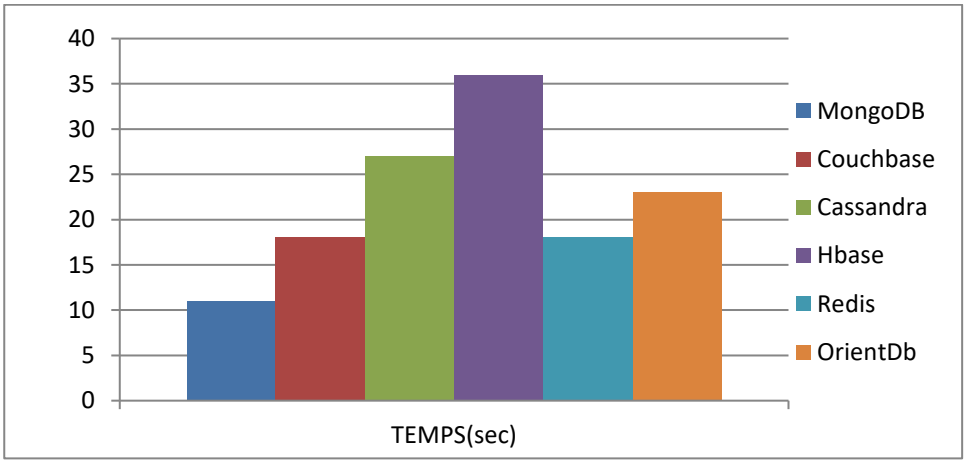


Figure A.33 : Temps d'exécution du Workload F

La Figure A.33 présente les résultats obtenus après exécution du Workload F composé de 50% d'opérations de lecture, pour la deuxième moitié : les enregistrements sont lus en premier lieu,

mis à jour ensuite et enfin sauvegardés. On peut reconfirmer une nouvelle fois la contre-performance des systèmes orientés colonnes à savoir HBase et Cassandra en raison de leurs contraintes éprouvées dans les opérations de lecture par rapport aux opérations de mise à jour. En outre, Redis et CouchBase ont obtenus desbons résultats (18 sec pour chacune). D'autre part la meilleure performance est celle de MongoDB qui est entrain de prouver son efficacité avec CouchBase pour les opérations de lecture.

IV.2.6 Workload G (5% Read, 95% Update)

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

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

Table A.14 : Temps d'exécution du Workload G

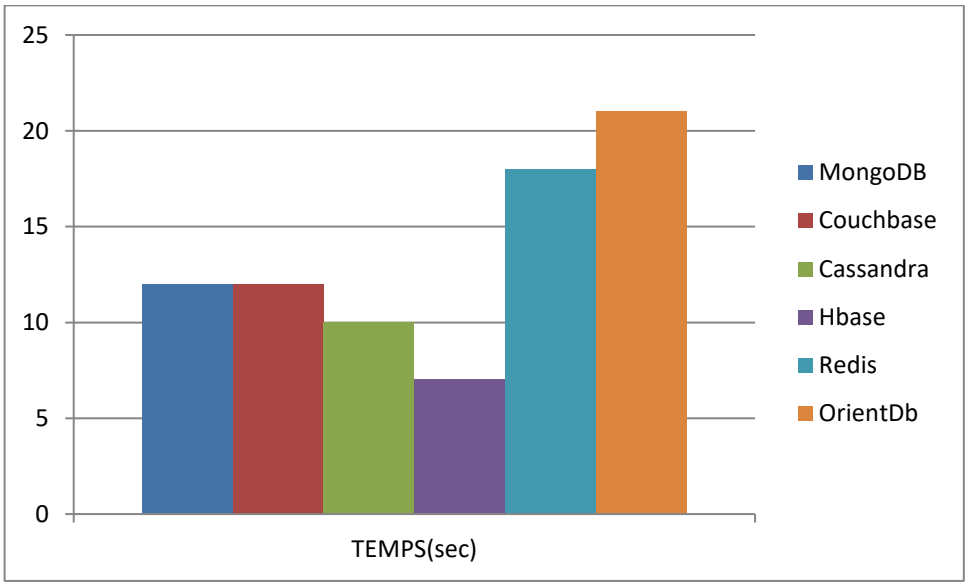


Figure A.34 : Temps d'exécution du Workload G

Les résultats du Workload G, révèlent que pour une charge composée principalement de mise à jour, les bases de données orientées colonne prennent cette fois-ci, l'ascendance sur les autres architectures, en revanche celles des clé-valeur ont été largement au-dessous.
Notons aussi que les deux systèmes orientés documents CouchBase et MongoDB ont obtenus les mêmes résultats (12 secondes chacune).

IV.2.7 Workload H (100% Update)

Les résultats d'exécution de la charge de travail H, sont récapitulés dans le tableau et la figure ci-dessous :

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

Table A.15 : Temps d'exécution du Workload H

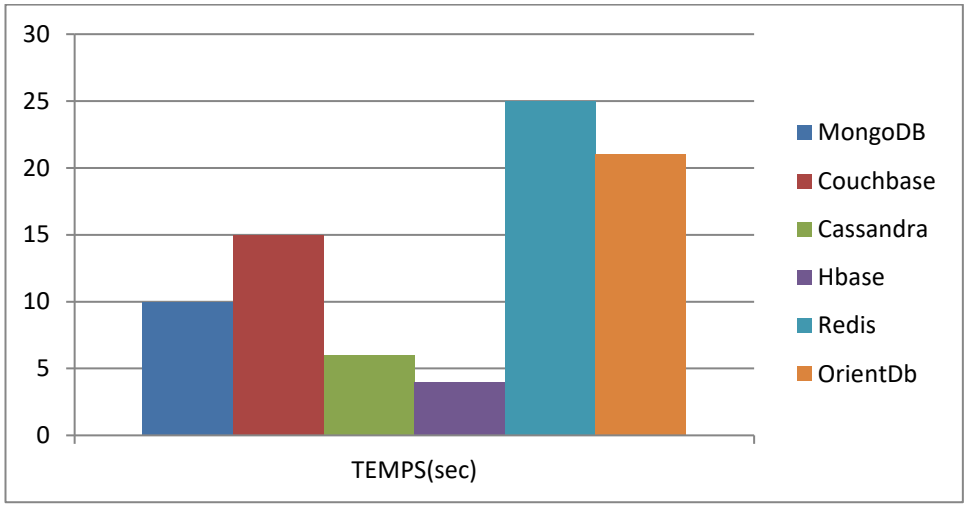


Figure A.35 : Temps d'exécution du Workload H

La Figure A.35 présente les résultats d'exécution du Workload H composé de 100% d'opérations (1000) de mise à jour.

Pour cette charge de travail purement mises à jour, les bases de données orientées colonnes HBase et Cassandra reconfirment nettement leurs performances atteintes lors d'exécution de la charge G par rapport à toutes les autres systèmes NoSQL. Les systèmes clé-valeur déçoivent une nouvelle fois pour cette catégorie de requêtes.

IV.2.8 Workload D (5% Insert, 95% Read)

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

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

Table A.16 : Temps d'exécution du Workload D

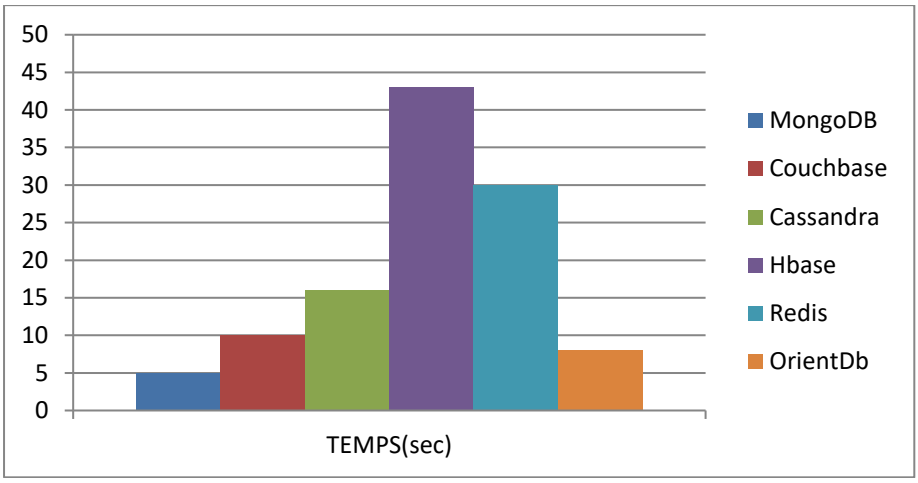


Figure A.36 : Temps d'exécution du Workload D

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

Les résultats d'exécution de cette charge de travail, confirment une nouvelle fois la performance des systèmes orientés documents, en particulier MongoDB qui était très rapide en exécutant la tâche en 5 secondes, et la contre-performance des orientés colonnes et surtout celle de HBase avec 43 secondes.

IV.2.9 Workload E (95% Scan, 5% Insert)

Les temps d'exécutions de la charge de travail E pour chacune des bases de données, sont illustrés dans le tableau et la figure ci-dessous :

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

Table A.17 : Temps d'exécution du Workload E

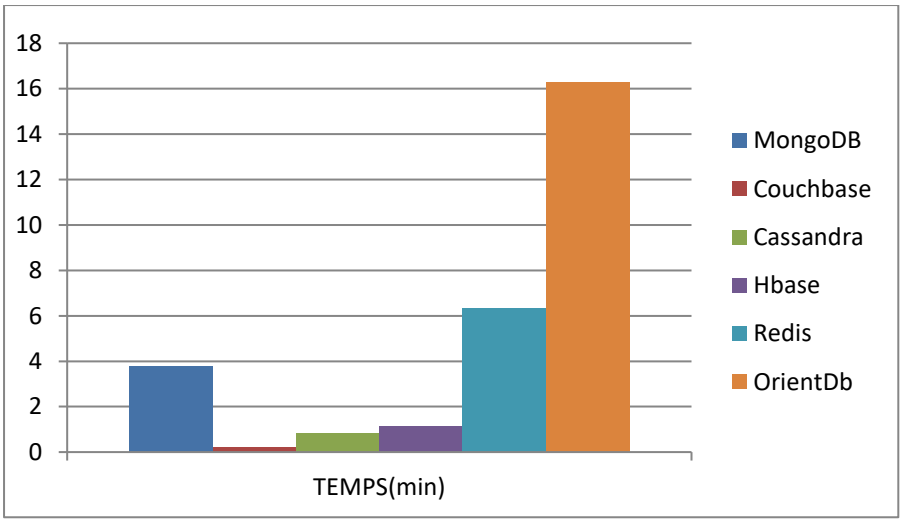


Figure A.37 : Temps d'exécution du Workload E

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

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

IV.2.10 Temps d'exécution global de l'ensemble des Workloads

Le tableau A.18 et la figure A.38 récapitule les résultats obtenus par les bases de données NoSQL pour toutes les charges de travail (A + B + C +D + E+ F + G + H)

BASE	MongoDB	Couchbase	Cassandra	HBase	Redis	OrientDb
TEMPS(min)	5	2	3	5	9	18

Table A.18 : Temps d'exécution global

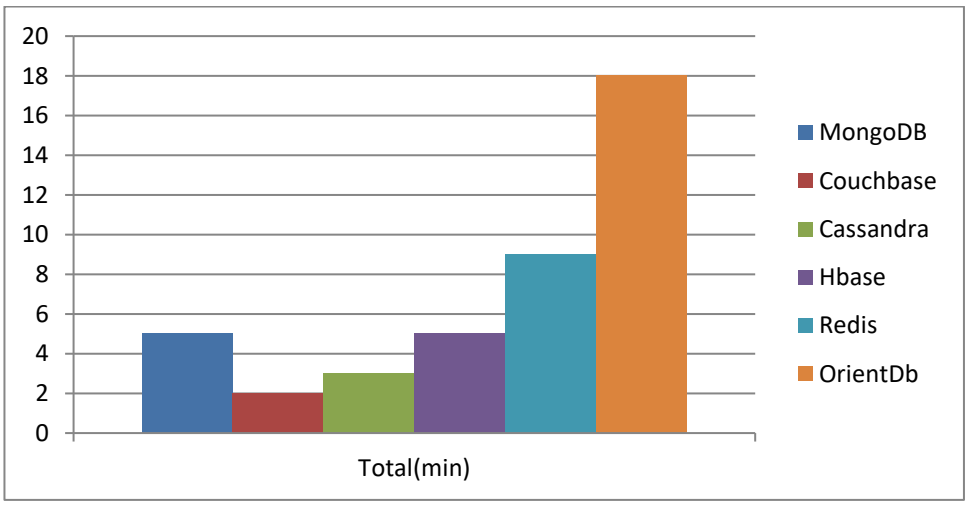


Figure A.38 : Temps d'exécution global

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

Le meilleur temps d'exécution est présenté par CouchBase avec 2 minutes seulement contre 3 minutes obtenues par Cassandra qui se place en seconde position. MongoDB et HBase se positionnent juste après avec 5 minutes. Les deux systèmes clé-valeur Redis et OrientDB se placent aux dernières positions avec 9 et 18 minutes respectivement.

IV.2.11 Evaluation globale pour les opérations de lecture et mise à jour

Les deux figures suivantes scindent entre les opérations de lecture et les opérations de mise à jour. La première figure contient en abscisses les huit charges de travail classées par ordre croissant des taux de lecture (de 0% à 100%) ainsi que la moyenne des temps d'exécutions. La deuxième figure contient en abscisses les huit charges de travail classées par ordre croissant des taux de mise à jour (de 0% à 100%) ainsi que la moyenne. Les runtimes figurent en ordonnées.

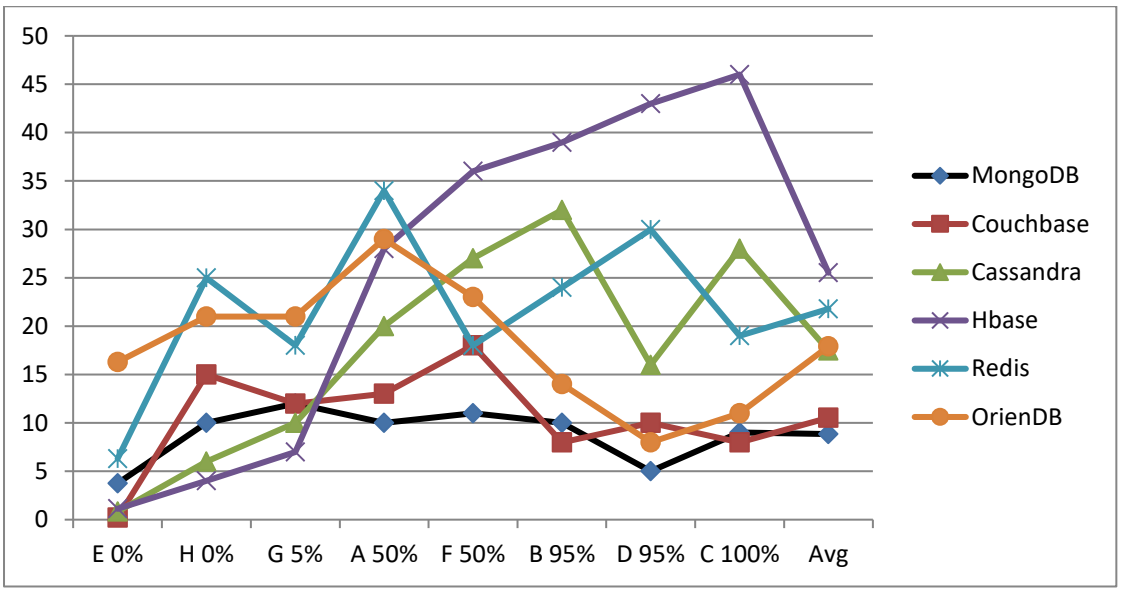


Figure A.39 : Evaluation globale pour les opérations de lecture (sec)

La Figure A.39 montre qu'en général, l'accroissement du taux d'opérations de lecture dans les Workloads est en faveur des modèles orientés documents par rapport aux autres modèles.

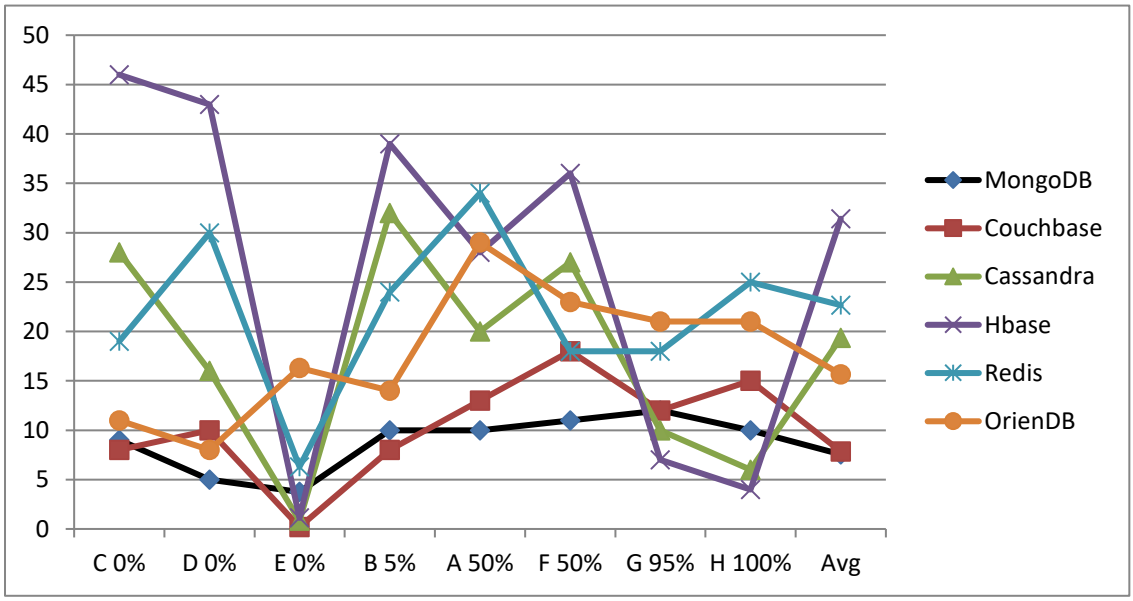


Figure A.40 : Evaluation globale pour les opérations de mise à jour (sec)

Contrairement à la Figure A.39, la Figure A.40 prouve globalement, que l'accroissement du taux d'opérations de mise à jour dans les Workloads est en faveur des modèles orientés colonnes par rapport aux autres modèles, ce qui peut être remarqué au niveau du dernier workload H, ou HBase et Cassandra étaient les plus performantes.

Après exécution des 1000 opérations sur les 600 milles enregistrements insérés, on peut remarquer qu'en moyenne, un léger avantage de systèmes orientés documents, sauf qu'on peut induire que l'augmentation du nombre d'opérations à effectuer, fixé à 1000, fera certainement la différence, en faveur des orientés colonnes.

IV.3 Synthèse des résultats

Après avoir interprété les résultats obtenus par les six systèmes, on peut conclure que les différentes optimisations et les mécanismes employés par les concepteurs des solutions NoSQL, afin d'améliorer les performances, comme une bonne exploitation des mémoires cache, ont une influence directe sur les temps d'exécutions des différents Workloads.

D'un autre côté, un environnement Hardware optimal équipé de processeurs rapides et de grandes mémoires vives avec des unités de disques modernes pour remédier à la lenteur des disques magnétiques, fournissent en évidence des performances supplémentaires aux différentes bases de données mais constitue en revanche, un support de stockage plus coûteux.

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

A travers les évaluations faites sur les performances des six bases de données NoSQL étudiées, plusieurs enseignements peuvent être tirés :

- Pour la phase de chargement des 600000 enregistrements, le meilleur résultat obtenu parmi les six bases testées c'était celui de CouchBase, ceci peut être justifié par le fait que cette dernière offre de hautes performances en raison de ses innovations qui lui permettent d'être la première à exploiter une couche de mise en cache RAM intégrée.
- HBase a prouvé sa performance dans les opérations d'insertions de nouveaux enregistrements dans la base de données. La bonne performance d'HBase dans ce premier test, est justifiée par le fait que HBase dispose d'une gestion de mémoire spécifique qui optimise son utilisation pour exécuter des opérations de chargement initial.
- CouchBase a été dans l'ensemble plus performante dans les opérations de lecture, par rapport aux autres.

- MongoDB a été aussi performante dans les opérations de lecture. Ceci est dû, principalement, à la cartographie des registres de MongoDB chargée en mémoire qui améliore ces performances en lecture, ceci est aussi confirmé par [1].
- Pour des opérations totalement ou majoritairement de lecture, HBase a été moins performante que les autres. Ceci est justifié par le fait qu'HBase assure un niveau de cohérence assez élevé par rapport aux autres, ainsi pour exécuter une opération de lecture, HBase compare toutes les copies et renvoie la copie la plus récente.
- Pour des opérations purement lecture il faut s'orienter vers les architectures orientées documents à savoir CouchBase et MongoDB.
- HBase et Cassandra ont prouvé leur performance une nouvelle fois dans les opérations d'écriture, cette fois ci dans la mise à jour des enregistrements déjà existants dans la base de données. Cela est expliqué par le fait qu'HBase et Cassandra utilisent un journal et des caches pour sauvegarder et garder la traçabilité de toutes les modifications effectuées. L'utilsation de ces mécanismes réduit considérablement le nombre d'opérations E/S effectuées sur le disque, caratérisé par une faible vitesse par rapport à la mémoire vive. La réplication des données est faite automatiquement, ce qui accélère le processus de mise à jour, ceci est confirmé aussi par [1]. Ainsi ces solutions sont optimisées spécialement pour effectuer des opérations de mise à jour.
- Le processus de mise à jour dans MongoDB, est ralentit proportionnellement par rapport aux nombres de mises à jour appliquées. Cette base de données utilise des mécanismes de verrouillage qui alourdissent le temps d'exécution de mise à jour.
- Les performances d'OrientDB et Redis se sont également dégradées avec le nombre croissant d'opérations de mise à jour.
- Pour les opérations de mis à jour lourde, il est très intéressant d'adopter des architectures orientées colonnes.
- Pour des opérations de scan, CouchBase, HBase et Cassandra ont prouvé leur performance.
- Pour les architectures clé-valeur, beaucoup d'efforts restent à fournir par les concepteurs pour améliorer leurs performances.

Dans l'ensemble, les solutions étudiées peuvent être classées en deux catégories, celles optimisées pour les lectures et celles optimisées pour les mises à jour. Ainsi, MongoDB, Redis, CouchBase et OrientDB sont optimisées pour effectuer des opérations de lecture, tandis que Cassandra et HBase ont de meilleures performances dans les opérations de mise à jour.

Notons aussi que quelques bases de données ont été moins performantes car elles requièrent davantage de capacités de système par rapport à l'environnement utilisé dans l'évaluation.
Enfin et après lecture de tous les résultats obtenus, il s’avère que les critères de choix de la solution adéquate dépendent, des besoins applicatifs, de la nature d’opérations exécutées sur les données manipulées et selon le contexte d’utilisation de ces dernières.

Conclusion

L'étude développée dans cette premiére partie de la thése, consiste à comparer les performances de 6 modèles NoSQL, très utilisés actuellement, à savoir MongoDB et CouchBase en tant qu'orienté documents, Cassandra et HBase en tant qu'orienté colonnes, Redis et OrientDB en tant que clé/valeur.

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

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

Après un rapprochement des résultats avec ceux obtenus par [1], on peut constater que nos résultats ont été globalement meilleurs, certainement à cause des différentes configurations de l'environnement Hard employées dans les deux études.

En outre, la lecture et l'analyse des résultats expérimentaux, peut affirmer qu'il y a des bases de données très performantes pour des Workloads particuliers, contrairement à d'autres qui étaient meilleures dans d'autres charges de travail ; alors il y a des solutions qui sont optimisées pour les lectures, celles pour les mises à jour et d'autres pour les opérations de scan, comme a été affirmé par [1].

Plusieurs travaux de recherche se focalisent actuellement sur le sujet qui se voit très prometteur et est considéré comme un chantier ouvert pour les chercheurs.

Une autre étude comparative a été menée entre plusieurs bases de données NoSQL dans un environnement distribué (1, 2, 4, 8, 16 et 32 nœuds) dans [70]. En outre, un autre système de comparaison NoSQL a été développé sur 12 serveurs dans [71]. Une évaluation de la dégradation des performances dans les bases de données NoSQL générées par la virtualisation est proposée dans [72]. Il existe aussi d'autres travaux comparant les performances des systèmes SQL avec NoSQL comme ceux dans [73, 74].

D'autres comparaisons entre les différentes familles de solutions populaires NoSQL, clé valeur, orientée document, orientée colonne et orientée graphe comme Elastic search, Memcached, Amazon DynamoDB, CouchDB, Accumulo et autres, dans un environnement Cloud, sont toujours très sollicitées et recommandées pour apporter l'assistance et l'aide nécessaire aux intéressés de Big Data et de Cloud Computing pour des éventuelles prises de décision sur les solutions convenables.

Partie B

Hadoop, HDFS & MapReduce

Partie B : Hadoop, HDFS & Mapreduce

Introduction

Historiquement, le premier modèle de mise en commun des données est « le Mainframe », qui était un ordinateur puissant centralisant les données et les traitements d'un système d'information. C'est le niveau le plus simple où le client ne dispose pas d'espace de stockage et se cantonne à un simple terminal informatique, éventuellement passif.
Le deuxième modèle est « l'Ordinateur Personnel » ou les utilisateurs peuvent travailler sur leur propre ordinateur comprenant des équipements de stockage. Ils peuvent accéder à des ordinateurs centraux, mais ne peuvent pas partagées leurs propres données entre eux.
Le deuxième modèle de mise en commun des données est « Clients Serveurs » : L'apparition des réseaux locaux a conduit à la constitution des groupes de travail. Le réseau transporte les données qui sont rendues accessibles aux groupes d'utilisateurs et qui peuvent être partagées par tous les membres du groupe.
Le dernier modèle est « l'Architecture Orientée service » qui consiste à étendre le travail en groupe sur des réseaux locaux à un travail coopératif par le biais de réseaux couvrant une surface plus grande. Ces réseaux sont connectés entre eux et relient des ordinateurs de différentes natures.

Les difficultés rencontrées en matière de performance par les modes d'accès traditionnels aux données, proviennent de l'accroissement de divers facteurs à savoir : Volume de données, évolution des disques, variété de systèmes d'exploitation et le nombre de nœuds de traitements.

L'industrie informatique a abondamment recours au parallélisme dès qu'une limite apparaît, alors la solution était la multiplication des systèmes hôtes afin de ne pas dépasser les limites d'un ordinateur. Les solutions Cloud Computing sont des architectures distribuées qui s'articulent sur plusieurs milliers de nœuds. « Storage-as-a-Service » englobe toutes les infrastructures matérielles équipées de logiciels mises à la disposition des clients du Cloud, c'est une fédération logique de grandes quantités d'équipements répartis sur plusieurs sites. Ce concept inclue un ou des espaces de stockage et un mode d'accès [5, 6, 7].

Les complexités de mise en œuvre des Clouds sont causées par la dispersion géographique en raison de l'externalisation des données hébergées dans des sites situés un peu partout dans le monde. Les architectures Clouds, permettent d'échanger des grands flux de données et métadonnées sur des réseaux distants. Ainsi la localisation des données et métadonnées restent délicates à traiter [75].

Si la difficulté de localisation des données est prise en compte dès le concept des Clouds, la localisation des métadonnées est plus difficile à décider. A ce sujet, La contrainte technique principale à l'extensibilité des Clouds demeure la gestion des métadonnées. La centralisation de ce service dans un seul serveur, comme c'est le cas dans HDFS et la plupart des architectures, constituera un goulot d'étranglement dans le futur qui peut affecter les performances du processus.

La mise en œuvre de la séparation du traitement des métadonnées et du transport des données n'est pas aisée, la recherche dans cet axe est toujours d'actualité. Les différentes approches d'amélioration de performance et d'extensibilité dans la matière seront très considérées dans les prochaines échéances.

Principalement, il existe deux approches de localisation des métadonnées centralisées ou distribuées. Le modèle P2P applique un parallélisme très fort, à tous les niveaux de données et de métadonnées. Le modèle Hadoop opte pour un service de métadonnées séparé avec un parallélisme massif pour le transport les données, et une centralisation des métadonnées afin de garantir la fiabilité du service et l'intégrité des métadonnées.

Le modèle présenté dans ce document apporte des propositions de remaniement sur le modèle HDFS de Hadoop, actuellement le plus répandu dans le marché informatique, en employant un parallélisme modéré des métadonnées. L'optique de cette solution hybride est d'accentuer la performance et l'extensibilité du modèle Hadoop.

Chap I : Systèmes de Gestion de Fichiers

Suite aux évolutions technologiques qui ont affecté l'industrie informatique, différents modèles de stockage et d'accès aux données sont émergés au cours de l'histoire informatique. Ces architectures sont toujours employées, dans les différents domaines informatiques et à des niveaux d'utilisations différents.

Les différentes manières d'accès aux données, pour une application, depuis le mécanisme de base jusqu'aux architectures mettant en relation plusieurs ordinateurs, qui ont précédé les modèles de stockage et d'accès aux données et des traitements spécifiques aux Clouds, sont [76] :

- Systèmes de fichiers locaux
- Systèmes de fichiers distribués
- Systèmes de fichiers partagés
- Systèmes de fichiers parallèles

Un système de gestion de fichiers (SGF) est une couche logicielle en charge de la gestion de fichiers qui s'exécute sur l'ordinateur connecté directement au support physique des fichiers. Le gestionnaire de fichiers établit la correspondance entre emplacement physique sur disque et adresse logique au sein du fichier. Il permet de traiter, de stocker et de partager des quantités importantes de données dans des fichiers sur des mémoires. Le SGF offre à l'utilisateur une vue abstraite sur les données et permet de les localiser à partir d'un chemin d'accès.

I.1 Système de fichiers local (SFL)

C'est la configuration la plus simple, ou le système employé est abrégé en un seul ordinateur auxquels sont rattachés les disques locaux pour héberger les données. Le système de fichiers local gère les données du système hôte unique disposant d'un ou plusieurs disques. C'est le mode d'accès aux données dans le cas où les utilisateurs travaillent sur un serveur accédant directement à son propre espace de stockage, sans aucun partage de données avec les utilisateurs travaillant sur d'autres systèmes. Les données structurées sont accédées par les moteurs de bases de données qui utilisent des mécanismes spécifiques comme le hachage, fichiers indexés ou encore B-arbre. Les données non structurées sont fournies aux applications sous la forme de fichiers.

Ce mode d'accès est connu par Direct Attached Storage (DAS), comme il est montré dans la Figure B.1 pour les exemples : FAT et NTFS pour Windows et EXT2, EXT3, EXT4 et ReiserFS pour Linux.

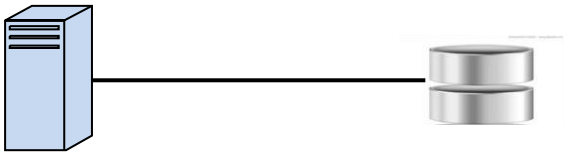


Figure B.1: Direct Attached Storage – DAS

Les différentes politiques employées par un système de fichiers local pour améliorer la performance, ciblent principalement les disques magnétiques.

I.2 Systèmes de fichiers distribués (SFD)

Le système de fichiers distribués désigne un service de données Client/Serveur via un réseau local, où la donnée est attachée au serveur et est partagée par l'ensemble des clients. C'est le mode le plus répandu de service de données. Le SFD transporte l'information (données et métadonnées) via le réseau entre un serveur et des clients. Depuis un système client, les fichiers apparaissent de la même manière que des fichiers locaux, ainsi l'existence du réseau local entre le client et le serveur est masquée. Le serveur offre aux clients une interface d'accès aux données en mode fichiers, si les données sont structurées, l'appellation plus précises est « BD distribuées ».

Ce mode d'accès est connu par Network Attached Storage, NAS (Figure B.2). Le serveur est désigné par le terme tête NAS ; Ce type d'architecture rajoute à la gestion des fichiers locaux, des couches logicielles de circulation des données et métadonnées sur le réseau, par exemples : CIFS pour Windows, NFS pour Linux/Unix. Les SFD sont implantés au sein du système d'exploitation du client par un aiguillage qui dirige les demandes des applications utilisant des fichiers distants, présents sur le serveur, vers les SFD, et vers les SFL les fichiers locaux, présents sur le client.

Les SFD ajoutent à la difficulté d'obtenir de bonnes performances depuis les fichiers locaux, la prise en compte du réseau entre le fichier sur le serveur et le programme applicatif sur les clients.

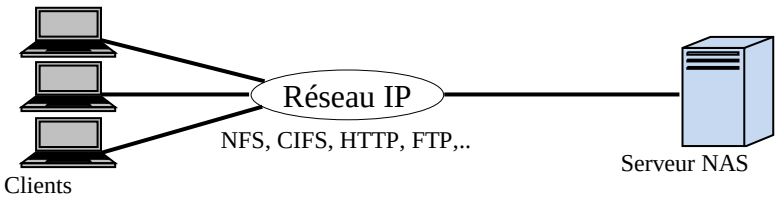


Figure B.2: Network Attached Storage – NAS

I.3 Systèmes de fichiers partagés (SFP)

Les Systèmes de fichiers partagés permettent de partager les données résidant sur les disques en commun. La fonction principale de cette architecture est de mettre en commun l'accès au stockage. Cette infrastructure est connue par Storage Area Network, SAN (Figure B.3). Des applications, situées sur des serveurs distincts, peuvent accéder simultanément à un même fichier via le SAN sans risque de corruption de données car la gestion d'accès concurrents est prise en charge par le logiciel de partage.

SAN est un réseau à haut débit, dédié au stockage reliant des ordinateurs et des équipements de stockage à travers des commutateurs, il est constitué de serveurs, d'éléments réseaux tels que des switchs ou des routeurs et des baies de disques qui vont fournir de l'espace de stockage. Une baie de disques ou disk array contient des disques qui sont pilotés par un ou des contrôleurs suivant la disponibilité des données que l'on souhaite (Figure B.3). Ces disques sont regroupés en volume via un système de RAID (Redundant Array of Independent or Inexpensive Disks, regroupement redondant de disques indépendants).

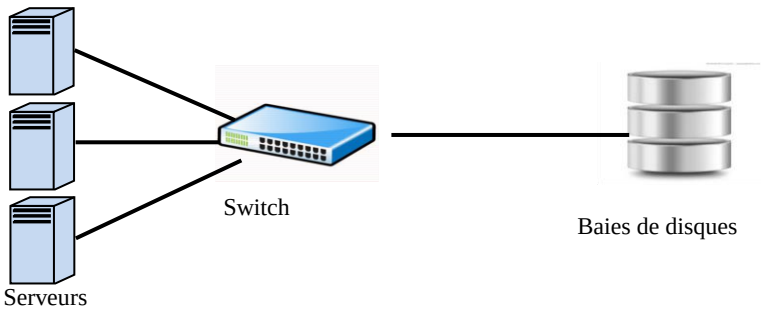


Figure B.3: Storage Area Network – SAN

I.4 Systèmes de fichiers parallèles (SFP)

L'évolution des modes d'accès précédents n'a pas empêché quelques contraintes qui atténuent les performances de ces architectures. La puissance du serveur unique est une barrière infranchissable et les goulots d'étranglements subsistent, malgré le parallélisme interne employé avec l'augmentation des CPU et de cœurs par CPU, qui fait accroître la performance du serveur. Les cartes réseau constituent pour elles aussi un autre goulot d'étranglement. Technologie des disques est une limite commune à tous les types d'accès. Le nombre limité des systèmes clients qui peut être supporté et le nombre limité des fichiers traités fixe un seuil à l'extensibilité. Ces obstacles à la performance ont conduit vers les systèmes de fichiers parallèles [78]. L'industrie informatique a abondamment recours au parallélisme dès qu'une limite apparaît. Ainsi la solution commune pour remédier aux limitations de chaque type

d'équipement est la multiplication des équipements, permettant la parallélisation des accès aux informations. Le parallélisme n'est pas une fin en soi mais une stratégie pour améliorer les performances.

Les SFP s'articulent sur un découplage du traitement des métadonnées de celui des données. Le ou les serveurs des métadonnées sont appelés MetaData Servers (MDS). Les chemins entre les clients et le stockage peuvent être multipliés à grande échelle. Le parallélisme permet à de multiples clients de communiquer simultanément avec plusieurs serveurs de transport de données, ça permet aussi à un seul client de traiter plusieurs flux de données en parallèle. Le client s'adresse d'abord au MDS pour connaître l'emplacement des données, le MDS pointe ensuite le client vers le stockage. Le goulot d'étranglement n'est plus le serveur unique, remplacé par un MDS et des serveurs de transport, mais le client lui-même. Cette infrastructure est connue par Parallel File System (Figure B.4).

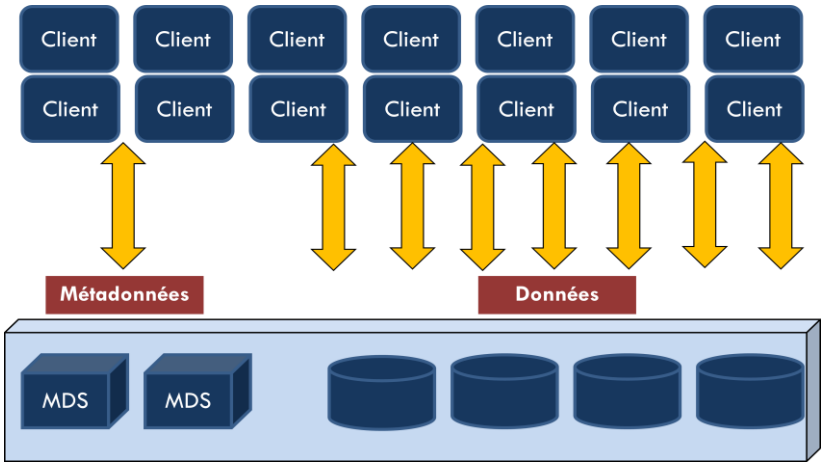


Figure B.4: Parallel File System – PFS

Les avantages apportés par les SFP sont principalement :

- L'extensibilité en termes de nombre clients.
- L'extensibilité en termes de capacité de stockage.
- Le débit de transfert de données.
- L'unification de l'espace de nommage.

D'autre part, on peut identifier les insuffisances ou limitations suivantes :

- Parallélisme du traitement des métadonnées est peu développé.
- Le nombre d'entités de données prises en charge est limité par la puissance du traitement des métadonnées.

- L'intégrité des données n'est pas une priorité dans les SFP.
- Les types de systèmes d'exploitation des clients sont relativement limités.
- L'empilement des protocoles et couches logicielles est une preuve de complexité.
- Grand débit atteint l'ordre de la centaine de Go/s mais l'ordre de grandeur du To/s n'est pas encore accessible.
- Multitude de clients peut atteindre une centaine de milliers mais ne couvre pas les besoins d'accès par le grand public sur le net, qui dépasse l'ordre des millions.

Chap II : Clouds de stockage et l'écosystème Hadoop

Dans le chapitre suivant, nous allons commencer par présenter le nouveau modèle de référence « Storage-as-a-Service » ainsi que les principales implantations dans les infrastructures Cloud Computing pour le stockage massif de données. Par la suite, nous allons étaler sur la solution la plus répandue dans le domaine, à savoir Hadoop. La dernière section est consacrée à l'aspect métadonnées, des systèmes de gestion de donnéesdans les grandes échelles, qui nécessite une bonne prise en charge vu leur influence sur les performances de ces systèmes. Notre approche proposée basée sur Hadoop, va se focaliser sur cet aspect.

II.1 Clouds de stockage

L'objectif fondateur du principe des Clouds de stockage est d'offrir le stockage en tant que service « Storage-as-a-Service ». Il qualifie le service Cloud qui englobe toutes les infrastructures matérielles équipées de logiciels les transformant en Cloud de stockage. Le Cloud de stockage est une notion incluant un ou des espaces de stockage et un mode d'accès, il se présente comme un nouveau modèle de référence d'accès aux données qui est à la fois une rupture et un prolongement des modes d'accès antérieurs. Ce nouveau modèle constitue l'architecture qui répond aux accroissements massifs des données que ce soit en volume, en nombre ou en type.

Le stockage en ligne sur le Cloud est désormais un comportement habituel et prend peu à peu la place des disques durs et autres supports physiques. Voici le top 15 des services disponibles sur le net : Dropbox, Microsoft OneDrive, Google Drive, Mega, pCloud, Tresorit, Box, Knowhow, Mediafire, Apple iCloud, Mozy, Spideroak, Amazon Cloud Drive, Egnyte, OpenDrive.

II.1.1 Relation Client-Serveur : Concepts et objectifs différents

La relation Client-Serveur entre les architectures précédentes et l'architecture Cloud est totalement renversé :

- Dans l'architecture Client-Serveur classique, les systèmes clients sont des machines de traitements applicatifs qui vont chercher les données qu'ils ont besoin pour travailler, auprès du service de données. Une fois les données modifiées, elles retournent sur le service de données pour y être stockées. Le traitement est réalisé sur le client, le serveur de transport ne faisant qu'approvisionner en données l'ensemble des clients.

- Une architecture Client-Serveur dans un environnement Cloud, comprend un très grand nombre de serveurs auxquels sont rattachés des espaces de stockage hébergeant des données qui sont utilisées par des applications dont les besoins en débit sont connus. Du fait, de leur très grand nombre, les serveurs de données ne sont pas complètement chargés et disposent de ressources en temps de calcul dont le Cloud va en profiter.Afin de ne pas réaliser un aller-retour de données de type Serveur-Données/ Serveur-Traitement/ Serveur-Données, les données ne sont plus transmises vers le serveur de traitement, mais ce sont les demandes de traitement qui sont envoyées aux serveurs de données, alors le principe sera d'approvisionner les serveurs de données en calcul plutôt que les serveurs de calcul en données.

II.1.2 Mise en œuvre des Clouds de stockage

Trois modèles de répartition des tâches de mise en œuvre des Clouds de stockage sont déployés dans les différentes solutions existantes :

- Une instance logique de données correspond à l'hébergement de la donnée sur un support physique, sans virtualisation d'aucune sorte. Les traitements et la prise en charge des données à travers la pile logicielle proposée sont du ressort des utilisateurs (IaaS : Amazon/EC2)
- Virtualisation totale du traitement, la gestion du stockage reste totalement liée à la localisation physique des données, c'est le cadre applicatif qui apporte élasticité et disponibilité aux applications sans intervention de leur part (SaaS : Google Apps, Salesforce.com)
- Entre ces 2 extrêmes, se situe le troisième modèle, où les applications sont exécutées par le fournisseur (PaaS : Microsoft Azure, Google App engine)

II.1.3 Implantation des Clouds de stockage

Le cloud de stockage est l'assemblage d'un ensemble de serveurs et de supports de stockage pour offrir l'apparence d'un seul élément (serveur + stockage). Il y a plusieurs niveaux d'implantations logicielles et matérielles possibles [76].

II.1.3.1 Niveaux d'implantation

Dans la partie des Clouds de stockage destiné aux applications, il y trois niveaux d'implantation logicielle. Ces niveaux d'implantation se différencient selon leur position par

rapport au système d'exploitation (OS) des serveurs ou par rapport à leurs emplacements serveur ou client :

1. Au niveau applicatif

- Sur le serveur
- Sur les machines clients
- Sur les deux

2. Selon leur position par rapport à l'OS des serveurs

- Mise en œuvre au sein des OS
- Partiellement à ce niveau
- Totalement en dehors (au niveau applicatif)

Le mode de mise en œuvre en partie au niveau de l'OS est le plus largement répandu, principalement en raison du grand succès d'Hadoop. La partie de cette implantation au sein de l'OS est le système de gestion de fichiers.

II.1.3.2 Architectures

Il existe deux principales architectures d'implantation des Clouds au niveau des systèmes de gestion de fichiers suivant la localisation des métadonnées qui peuvent être centralisées ou distribuées :

- Architecture symétrique (P2P) : où l'ensemble des métadonnées est totalement distribué. Elles sont situées sur plusieurs serveurs, en général les mêmes que ceux stockant les données.
- Architecture centralisée : Une architecture à base d'un service de métadonnées centralisé où les métadonnées résident dans des serveurs différents que ceux hébergeant les données (comme celle d'Hadoop). Un serveur de métadonnées étant un point de passage obligatoire.

1. Modèle P2P

Une architecture P2P traite les données et les métadonnées de la même manière en appliquant un parallélisme massif à tous les niveaux. La distribution des objets est réalisée par un algorithme propre à chaque solution ; c'est dans cet algorithme que réside les avantages associés à chaque solution. La distribution des objets, la facilité d'accès à ces objets et l'équilibrage de charge résultent de cet algorithme. Il faut bien placer les objets pour équilibrer

les espaces de stockage et de manière à les retrouver le plus rapidement possible en ciblant bien le serveur de stockage.

Les deux avantages de cette architecture distribuée étant l'extensibilité et l'absence de point de défaillance unique. D'une part, l'absence du serveur centralisé fait de ce modèle très tolérant aux fautes ; la réplication de données et de métadonnées permet de résister à de multiples défaillances de couples (serveur + stockage). D'autre part, l'élasticité parait infinie, puisque chaque serveur de stockage étant un point d'entrée permettant d'héberger un objet. La seule barrière de l'extensibilité reste les limites d'efficacité de l'algorithme de placement d'un objet. Cette architecture symétrique est préférable à une architecture centralisée dans un Cloud où les données n'ont pas besoin d'être traitées en temps réel ou de manière très performante.

2. Modèle HDFS

Contrairement à P2P, l'architecture adoptée par Hadoop traite les données et les métadonnées de manière complètement différente, à base d'un service de métadonnées centralisé, comme dans les PFS. C'est un modèle qui se considère, incontestablement, comme la mise en œuvre la plus largement répandue des Clouds [76, 77].

Le parallélisme massif est employé uniquement pour le transport les données afin d'éviter les goulots d'étranglements. En ce qui concerne les métadonnées, HDFS réserve un traitement particulier et sûr aux métadonnées avec une centralisation du traitement et du stockage des métadonnées dans des serveurs particuliers afin de garantir la fiabilité du service et l'intégrité des métadonnées imposée par l'influence de ces informations sur le système. C'est dans cette optique qu'HDFS d'Hadoop a adopté un découplage du service des métadonnées au service de stockage en centralisant le serveur des métadonnées et en l'isolant des serveurs de stockage. C'est une implantation très réussie, utilisée par les grands acteurs d'internet.

Hadoop ainsi que son système de gestion de fichiers HDFS vont faire l'objet des sections suivantes.

II.2 Hadoop

II.2.1 Présentation générale

Apache Hadoop (High-Availability Distributed Object-Oriented Platform) est un framework Javalibre,crée par Doug Cutting, destiné à faciliter la création d'applications distribuées et extensibles [78]. Il est dédié aux traitements de données volumineuses pouvant aller jusqu'à plusieurs pétaoctets. C'est la mise en œuvre la plus connue du modèle de programmation

MapReduce [22, 79]. Il a été initié en grande partie par Google et Yahoo et a été programmé par Apache Software Foundation.

En 2004, Google a publié un article présentant son algorithme MapReduce, basé sur des opérations analytiques à grande échelle sur un grand cluster de serveurs, ainsi que son système de fichier en cluster, GoogleFS. *Doug Cutting* qui travaillait à cette époque sur le développement d'Apache Lucene cherchant une solution pour accroître la taille de l'index de son moteur web Nutch, décida alors de reprendre les mêmes concepts pour développer sa propre version des outils en version Open Source, c'est ce qui deviendra le projet Hadoop. Il s'inspira du doudou de son fils de 3 ans, un éléphant jaune, pour le logo ainsi que pour le nom de ce nouveau framework Java [77].

Certains classent ce type de logiciels parmi les solutions NoSQL qui cherchent à surmonter les limites des bases de données relationnelles au profit de la simplicité, la scalabilité et la performance [34].

En fait, Hadoop est un ensemble de logiciels et d'outils qui permettent de créer des applications distribuées. C'est une plate-forme logicielle open-source, écrite en Java et fondée sur le modèle **MapReduce** [78] et le système de fichiers distribués **HDFS** [77], qui permet de prendre en charge les applications distribuées en analysant de très grands ensembles de données. Ce sont les deux composantes principales formant l'écosystème Hadoop, écosystème fortement convoité et qui se trouve au centre de l'univers du Big Data.Plusieurs travaux utilisent le framework Hadoop [80].

Hadoop a deux couches principales à savoir la couche de stockage HDFS (Hadoop Distributed File System) et la couche de traitement / calcul (Moteur MapReduce). Plus d'autres utilitaires et modules, tels que les bases de données (HBase et Cassandra), les outils de requêtage (Hive et Pig) et la coordination (ZooKeeper) [81].

Hadoop propose son système de fichier distribué **HDFS** pour assurer le stockage et l'intégrité des données en dupliquant plusieurs copies d'un même bloc à travers des dizaines, des centaines, voire des milliers de machines différentes, ce qui amène un cluster Hadoop à être configuré sans requérir un système RAID. D'autre part, Hadoop fournit son système d'analyse de données **MapReduce** pour effectuer des traitements de gros volumes de données grâce à une répartition efficace des tâches sur les différents nœuds de calcul. Hadoop est utilisé particulièrement dans l'indexation et le tri de grands ensembles de données, le data mining, l'analyse de logs et le traitement d'images.

Hadoop n'a pas été conçu pour traiter de grandes quantités de données structurées à grande vitesse. Cette mission reste largement l'apanage des grands systèmes de Data Warehouse et

de Datamart reposant sur des SGBD traditionnelles et faisant usage de SQL comme langage de requête. La spécialité d'Hadoop, ce serait plutôt le traitement à très grande échelle de grands volumes de données non structurées tels que des documents textuels, des images, des fichiers audio… même s'il est aussi possible de traiter des données semi-structurées ou structurées avec Hadoop [82].

Hadoop fait partie des projets de la fondation de logiciel Apache depuis 2009. Il est destiné à faciliter le développement d'applications distribuées et scalables, permettant la gestion de milliers de nœuds ainsi que des pétaoctets de données. En 2011, Yahoo! crée Horton works, sa filiale dédiée à Hadoop, fidèle à la distribution Apache et donc 100% open source. De la même manière, Cloudera, créé au début de l'année 2009, se place comme l'un des plus gros contributeurs au projet Hadoop, open source en grande partie sauf pour les outils d'administration. A côté de ces deux solutions, on retrouve MapR (2009) avec le noyau Hadoop mais repackagé et enrichi de solutions propriétaires. Cloudera, Horton works et MapR, sont actuellement les trois principales distributions Hadoop disponibles sur le marché.

Exemple d'usage

Ce qui fait la spécificité de Hadoop est qu'il est conçu pour traiter un énorme volume de données en un temps record. A titre d'exemple, les laboratoires de Yahoo! ont trié l'équivalent de 500 GB de données en 59 secondes sur un Cluster de 1400 nœuds (Avril 2009) [82].

Sa vocation première est donc d'implémenter des traitements batchs performants, particulièrement lorsqu'ils impliquent un volume de données très important. En dehors de Yahoo!, citons les cas d'utilisation de deux sociétés prestigieuses :

- La plateforme musicale Last.fm met en œuvre Hadoop pour générer les statistiques hebdomadaires (Tops artistes et Top titres) ou mesurer les tendances musicales.
- Facebook l'utilise pour la production de rapports à usage interne, comme la performance des campagnes publicitaires opérées par la plateforme sociale, et les statistiques diverses (croissance du nombre des utilisateurs, consultation des pages, temps moyen de consultation du site, etc.).

II.2.2 Hadoop et l'infrastructure de stockage de données

Les éditeurs traditionnels dans le domainede stockage (Terradata, Oracle, …) ont clairement évolué vers la convergence et l'intégration d'Hadoop dans leurs piles logicielles et matérielles. Hadoop est capable de stocker et d'interroger de très grands volumes de données structurées et non structurées (HDFS propose des snapshots et autres mécanismes intelligents de réplication semblable au mécanisme des baies de stockage évoluées).

Grâce à Hadoop, L'entreprise aujourd'hui peut utiliser une grappe de serveurs comme moyen de stockage de données au lieu d'acquérir des baies de stockage extrêmement coûteuses. L'environnement Hadoop permet de séparer la couche hardware du traitement des données. Il se charge de stocker et de garantir l'intégrité des données.

Hadoop s'est imposé comme un nouveau paradigme en matière de stockage de grandes masses de données grâce à ces caractéristiques etaux avantages fournis, notamment :

- Capacité : la faculté d'avoir un stockage élastique permet d'accroître aisément la capacité de stockage en rajoutant des nœuds. Cette notion reste fondamentale à la mise en place d'infrastructure Big Data.
- Latence : l'infrastructure permet la gestion de flux en temps réels, ainsi qu'un support d'E/S de disques importants.
- Coût : au-delà des coûtsinhérents du stockage physique, les coûts des opérations humaines sont aussi réduits.
- Virtualisation : ne plus s'appuyer directement sur une infrastructure physique coûteuse (baie de stockage) mais choisir la virtualisation de ses clusters, permet à Hadoop d'apporter les bénéfices suivants :
    - Facilité de paramétrages des clusters ;
    - Rapidité de déploiement ;
    - Haute disponibilité ;
    - Utilisation optimale des ressources.

II.2.3 MapReduce

L'architecture distribuée implantée pour le stockage et le traitement des données Big Data présente un obstacle qui ralenti considérablement les performances souhaitées par les utilisateurs.

Pour répondre à ces nouvelles exigences de performances et de volumes, les concepteurs informatiques ont dû recourir au parallélisme des traitements par une fragmentation (décomposition) des données et leurs traitements simultanés (en parallèle). Plusieurs outils et techniques ont été développés et employés dans les environnements distribués pour prendre en en charge le parallélisme massif des traitements requis des données hébergées. Ce traitement de données de façon distribuée soulève certaines questions : Comment distribuer le

travail entre les serveurs ? Comment synchroniser les différents résultats ? Comment gérer une panne d'une unité de traitement ?

Comme solution proposée, on retrouve, Principalement « MapReduce » [8, 78].

II.2.3.1 Présentation

« MapReduce » a émergé comme modèle de programmation et une implémentation associée pour le traitement et la génération de grands volumes de données avec un algorithme distribué parallèle sur un cluster. MapReduce est à l'origine une technique de programmation connue de longue date en programmation fonctionnelles'inspirant des langages fonctionnels et plus précisément du langage Lisp. MapReduce est un paradigme de programmation, dans lequel sont effectués des traitements parallèles et distribués de données potentiellement très volumineuses, supérieures en taille à 1 téraoctet.

Ce Framework popularisé par Google en 2004, est actuellement intégré dans plusieurs solutions Big Data.

MapReduce est implémenté au cœur du projet Hadoop. Il existe plusieurs versions du MapReduce comme la version 0.x et 1.x (architecture purement maître-esclave) ou la version 2.x YARN (architecture maître-esclave à deux niveaux).

MapReduce est le second composant majeur d'Hadoop, après HDFS, conçu pour gérer la répartition et l'exécution des requêtes sur les données stockées sur le Cluster.Le Framework MapReduce est conçu pour traiter des problèmes parallèlisables à très grande échelle en s'appuyant sur un très grand nombre de nœuds. L'objectif de MapReduce et de son mécanisme avancé de distribution de tâches est de tirer parti de la proximité des données aux traitements sur le même nœud de façon à minimiser l'impact des transferts de données entre les nœuds du Cluster sur la performance [82].

Les principes de MapReduce sont le parallélisme des traitements et leur distribution sur les serveurs de données. Il est conçu pour la scalabilité et la tolérance aux pannes. Ce modèle est en train de connaitre un vif succès auprès des sociétés possédant d'importants Datacenters telles qu'Amazon ou Facebook [22].

II.2.3.2 Principe

Le principe de MapReduce est simple comme ci-illustré dans la figure B.5: il s'agit de découper une tâche manipulant un gros volume de données en plusieurs tâches traitant chacune un sous-ensemble de ces données.

L'exécution des jobs se fait à l'aide d'un JobTracker et de TaskTrackers : lors de son exécution le job est soumis au JobTracker qui s'occupe de le distribuer au TaskTracker qui tourne sur

chaque noeud. Le JobTracker choisit toujours les TaskTrackers qui sont les plus proches de l'emplacement de la donnée à traiter [83].

MapReduce est composé de deux fonctions Map() et Reduce() qui opèrent sur des couples (clés, valeurs) en entrée et en sortie :

- **Map** : Transforme les données d'entrée en une série de couples clef /valeur et distribue le traitement sur les différents serveurs pour aboutir à un premier niveau de résultats de synthèse intermédiaires. Le parallélisme est la caractéristique de cette première phase : le nœud analyse le problème, le découpe en sous-problèmesou fragments et les délègue à d'autres nœuds (qui peuvent en faire de même récursivement) pourfaire exécuter l'opération *MAP* à chaque machine du cluster sur un fragment distinct. Chaque nœud traite un sous-ensemble des données [84].

  La fonction *Map* s'écrit de la manière suivante :

  Map (clé1, valeur1) → Liste (clé2, valeur2).

- **Reduce** : Applique un traitement à toutes les valeurs de chacune des clés distinctes produites par l'opération *MAP*. Reduce consolide sur un nombre plus réduit de serveurs, jusqu'à un seul point pour simplifier, l'ensemble des résultats des différentes tâches de Map, et réalise une synthèse des résultats intermédiaires pour répondre à la requête du client : les nœuds les plus bas font remonter leurs résultats au nœud parent qui les avait sollicités. Celui-ci calcule un résultat partiel à l'aide de la fonction *Reduce* (réduction) qui associe toutes les valeurs correspondantes à la même clé à une unique paire (clé, valeur). Puis il remonte l'information à son tour [84].

  La fonction *Reduce* s'écrit de la manière suivante :

  Reduce (clé2, Liste(valeur2)) → Liste(valeur2).

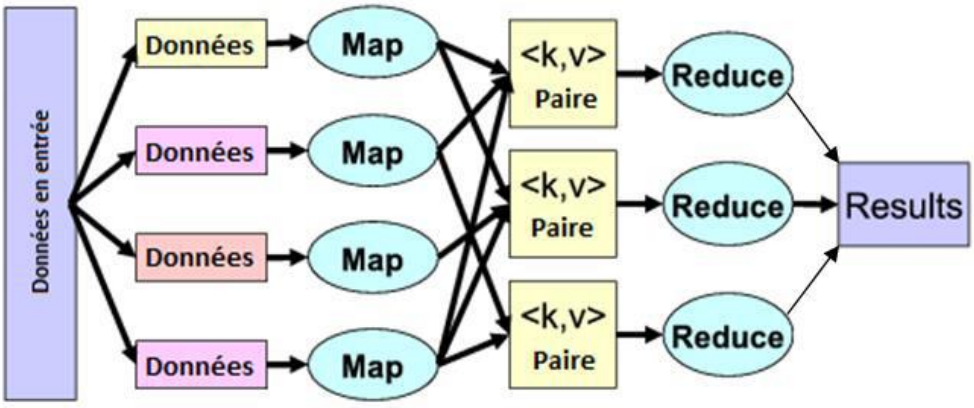


Figure B.5 : Principe Map-Reduce

II.2.3.3Architecture fonctionnelle

L'architecture maître-esclave de MapReduce s'articule sur les 2composants (Figure B.6) :

- Le maître MapReduce : le JobTracker ;
- Les esclaves MapReduce : les TaskTracker.

JobTracker [85]

- Gère l'ensemble des ressources du système ;
- Reçoit les jobs des clients ;
- Ordonnance les différentes tâches des jobs soumis ;
- Assigne les tâches aux TaskTrackers ;
- Réaffecte les tâches défaillantes ;
- Maintien des informations sur l'état d'avancement des jobs.

TaskTracker [85]

- Exécute les tâches données par le JobTracker ;
- Exécution des tâches dans une autre JVM (Child) ;
- A une capacité en termes de nombres de tâches qu'il peut exécuter ;
- Heartbeat avec le JobTracker : Le processus Heartbeat se charge de passer un message-aller vers le JobTracker indiquant son état.

L'exécution d'un job MapReduce, concerne quatre entités indépendantes [86] :

- Le client, qui soumet le job MapReduce ;
- Le JobTracker, qui coordonne l'exécution et le partage du job en sous tâches. Le JobTracker est une application Java dont la classe principale est JobTracker ;
- Les TaskTrackers, qui gèrent les tâches que le JobTracker lui a confiées. Les Tasktrackers sont des applications Java dont la classe principale est TaskTracker;
- Le système de fichiers distribué, qui est utilisée pour le partage de fichiers de travail entre les autres entités (TaskTrackers).

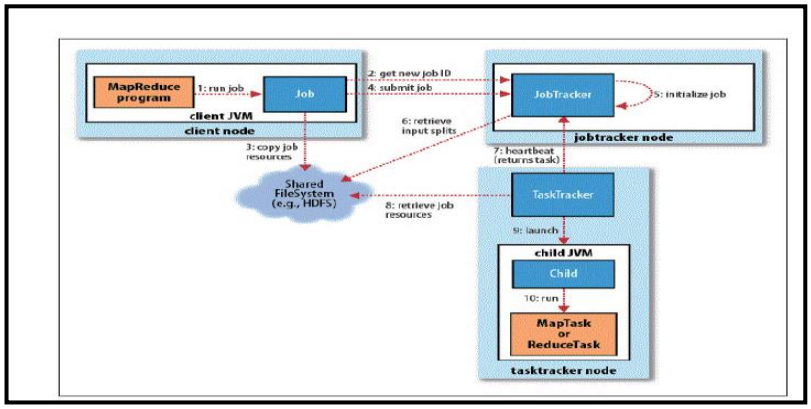


Figure B.6: Structure fonctionnelle de MapReduce1 [86]

II.2.3.4 Hadoop MapReduce 2.x: YARN

YARN (Yet-Another-Resource-Negotiator) est un nouveau composant intégré dans l'architecture maître-esclave par rapport aux architectures précédentes des versions 0.x et 1.x pour qui permet de fournir une nouvelle architecture à deux niveaux (Figure B.7) [86].

- Le YARN Ressource Manager, coordonne l'attribution des ressources de calcul sur le cluster ;
- LesYARN NodeManagers, lancent et contrôlent les conteneurs de calcul sur les machines de la grappe ;
- Le MapReduce application master, coordonne les tâches en cours d'exécution de MapReduce. Ce dernier avec les tâches MapReduce sont exécutés dans des conteneurs programmés par le YARN RessourcesManager et gérés par les Nodes Managers;
- Le système de fichiers distribué HDFS, est utilisé pour le partage de fichiers de travail entre les autres entités.

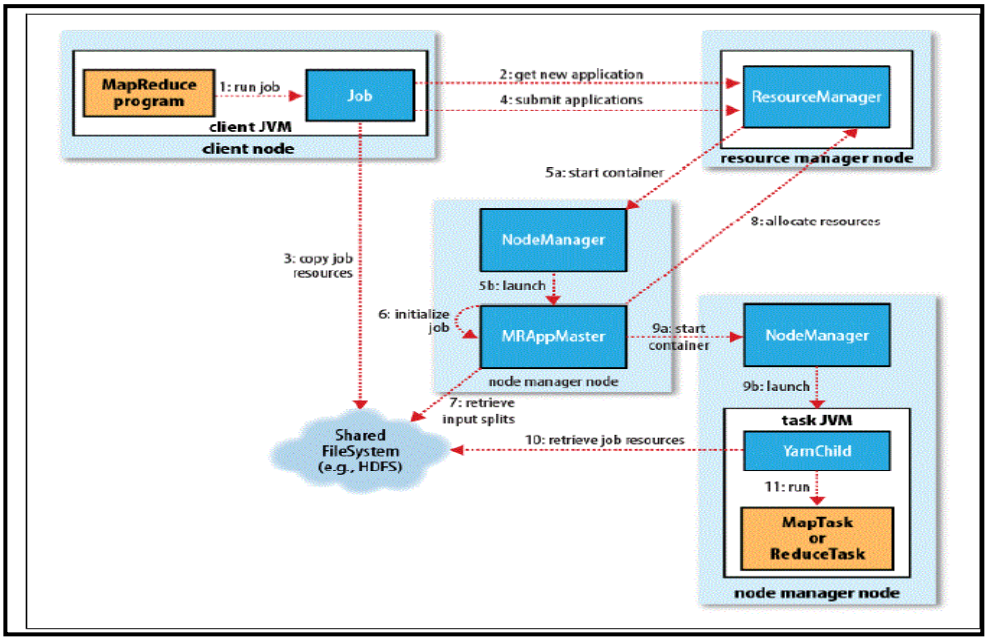


Figure B.7: Structure fonctionnelle de MapReduce2 [86]

II.2.4 Hadoop Distributed File System

La partie gestion des données de cette implantation est Hadoop Distributed File System. HDFS est un système de fichiers distribué, extensible et portable inspiré du système GFS (Google File System) développé par Google. Il a été conçu pour stocker de très gros volumes de données sur un grand nombre de machines équipées de disques durs banalisés. HDFS fournit un haut débit d'accès aux données et est adapté pour les applications qui nécessitent de grands volumes de données [85].

HDFS va prendre les données en entrée, qu'elles soient structurées ou non, et va ensuite les partitionner en plusieurs blocs de données. Afin de garantir une disponibilité des données en cas de panne d'un nœud, le système fera un réplica des données. Par défaut les données sont répliquées sur trois nœuds différents, deux sur le même support et un sur un support différent. Les différents nœuds de données peuvent communiquer entre eux pour rééquilibrer les données [46, 84].

HDFS est fait pour stocker de gros fichiers de l'ordre du GB voire du TB. HDFS est fait pour une très grande scalabilité et tourne déjà en production sur des très gros Clusters (plus de 1000 machines) dans des sociétés comme Facebook ou Yahoo [83]. HDFS est conçu pour être utilisé avec Hadoop dont la philosophie d'implémentation repose sur le principe qu'il est plus efficace de déplacer les unités de calcul que les données. HDFS fournit donc une interface pour pouvoir déplacer le traitement au plus près des données associées.

II.2.4.1 Caractéristiques

Parmi les particularités d'Hadoop Distributed File System (HDFS) en tant que système de gestion de fichiers moderne, c'est le fait qu'il assure un stockage à bas coût (excluant le SAN/NAS qui sont couteux en matériel). Il assure aussi une élasticité illimitéeen stockage (scale-out) grâce à son architecture distribuée. Il permet l'abstraction de l'architecture physique de stockage, afin de manipuler un système de fichiers distribué comme s'il s'agissait d'un disque dur unique. Le goulot d'étranglement d'accès aux données est dégagé par un parallélisme massif, étendu jusqu'à plusieurs milliers de chemins qui conduisent à des nœuds de stockage (serveur + stockage). Ces voies de transport de données se multiplient par la présence de plusieurs unités de disques sur chaque serveur. HDFS emploie un grand nombre de serveurs gérant les données et métadonnées pour un grand nombre de clients.

HDFS se démarque d'un système de fichiers classique pour les principales raisons suivantes [87] :

- HDFS n'est pas fusionné au noyau du système d'exploitation. Il assure une portabilité et peut être déployé sur différents systèmes d'exploitation. Un des inconvénients est de devoir solliciter une application externe pour monter une unité de disque HDFS.
- HDFS est un système distribué sur un système classique ou la taille du disque est généralement considérée comme la limite globale de stockage, contrairement à un système distribué comme HDFS, ou chaque nœud d'un cluster correspond à un sous-ensemble du volume global de données du cluster. Pour augmenter ce volume global, il

suffira d'ajouter de nouveaux nœuds. On retrouvera également dans HDFS, un service central appelé NameNode qui aura la tâche de gérer les métadonnées.

- HDFS utilise des tailles de blocs largement supérieures à ceux des systèmes classiques ou la taille est fixée par défaut à 64 Mo. Il est toutefois possible de monter à 128 Mo, 256 Mo, 512 M, voire 1 Go. L'intérêt est de fournir des tailles plus grandes permettant de réduire le temps d'accès à un bloc. Notez que si la taille du fichier est inférieure à la taille d'un bloc, le fichier n'occupera pas la taille totale de ce bloc.
- HDFS fournit un système de réplication des blocs dont le nombre de réplications est configurable. Pendant la phase d'écriture, chaque bloc correspondant au fichier est répliqué sur plusieurs nœuds. Pour la lecture, si un bloc est indisponible sur un nœud, des copies de ce bloc seront disponibles sur d'autres nœuds comme ci-illustré dans la Figure B.8.

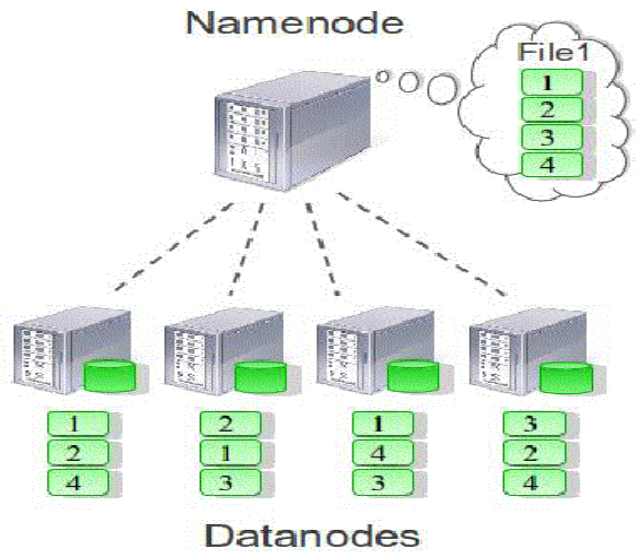


Figure B.8: Hadoop Distributed File System (HDFS) [85]

II.2.4.2 Architecture

HDFS adopte une architecture de type maître/esclave qui s'articule sur cinq éléments (Figure B.9) :

- JobTracker : un aiguilleur de tâches centralisé ;
- TaskTrackers : des gestionnaires de traitements locaux aux serveurs ;
- DataNode : les nœuds de stockage ;
- NameNode : un serveur central de métadonnées ;
- Secondary NameNode : un serveur de secours.

Les applications soumettent leurs travaux au JobTracker qui distribue le travail aux TaskTrackers disponibles, de façon à ce que les traitements soient proches des données le plus possible. Si une tâche n'a pas échangé d'informations avec l'extérieur de son serveur d'exécution, elle peut être redémarrée par le JobTracker sans problèmes, sur un autre serveur ; ce qui explique la fiabilité du processus. La même tâche peut être exécutée simultanément par plusieurs serveurs, ainsi le traitement global est accéléré [76, 77, 78].

Figure B.9: Architecture d'un Cluster Hadoop [76]

– **Nœud principal ou NameNode**

Le « NameNode » est la pièce centrale dans HDFS, il n'y a qu'un « NameNode » par cluster HDFS. Il maintient une arborescence de tous les fichiers du système et gère l'espace de nommage. Il centralise la localisation des blocs de données répartis sur le système. Sans le « NameNode », les données peuvent être considérées comme perdues car il s'occupe de reconstituer un fichier à partir des différents blocs répartis dans les différents « DataNode ». Le NameNode dispose d'un fichier de configuration dans lequel il entre l'adresse de chaque machine sur laquelle tourne un DataNode, et se connecte ensuite à chacune de ces machines via SSH au démarrage de HDFS. Par ailleurs, le NameNode est l'arbitre et le dépositaire de toutes les métadonnées d'HDFS. HDFS est conçu de telle sorte qu'aucune donnée de l'utilisateur ne transite par le NameNode [83].

Essentiellement, les tâches assumées par le NameNode sont [85] :

- Responsable de la distribution et de la réplication des blocs ;
- Serveur d'informations HDFS pour le client HDFS ;
- Stocke et gère les métadonnées ;

- Liste des blocs pour chaque fichier (dans le cas d'une lecture) ;
- Liste des DataNodes pour chaque bloc (dans le cas d'une écriture) ;
- Journalise les métadonnées et les transactions sur un support persistant ;
- Lectures/écritures ;
- Créations/suppressions ;
- Démarre à partir d'une image de HDFS (fsimage).

– **Gestionnaire de tâches ou JobTracker**

Il s'occupe de la coordination des tâches sur les différents clusters. Il attribue les fonctions de MapReduce aux différents « TaskTrackers ». Le « JobTracker » est un « Daemon » cohabitant avec le « NameNode » et ne possède donc qu'une instance par cluster.

– **Moniteur de tâches ou TaskTracker**

Il permet l'exécution des ordres MapReduce, ainsi que la lecture des blocs de données en accédant aux différents « DataNodes ». Par ailleurs, le « TaskTacker » notifie de façon périodique au « JobTracker » le niveau de progression des tâches qu'il exécute, ou alors d'éventuelles erreurs pour que celui-ci puisse reprogrammer et assigner une nouvelle tâche. Un « TaskTracker » est un « Deamon » cohabitant avec un « DataNode », il y a un donc un « TaskTracker » par « DataNode ».

– **Nœud de données ou DataNode**

Le nœud de stockage (DataNode), est chargé principalement de [85] :

- Stocker des blocs de données dans le système de fichier local ;
- Communiquer périodiquement au « Name Node » une liste des blocs qu'il gère ;
- Heartbeat avec le NameNode : Heartbeat est système sous Linux permettant la mise en clusters de plusieurs serveurs pour assurer une tolérance de panne. Le processus Heartbeat se charge de passer un message-aller vers le NameNode indiquant : son identité, sa capacité totale, son espace utilisé, son espace restant.

Un HDFS contient plusieurs nœuds de données ainsi que des réplications d'entre eux. Ce sont les nœuds esclaves [46].

– **Nœud secondaire ou Secondary NameNode**

Dans une architecture Hadoop, le NameNode est un point unique de défaillance (SPOF- Single Point Of Failure). Si ce service est arrêté, il n'y a pas moyen de pouvoir extraire les blocs d'un fichier donné. N'étant initialement pas présent dans l'architecture Hadoop, un NameNode secondaire appelé Secondary NameNodea été ajouté dans l'architecture Hadooppar la suiteafin de répondre au problème du SPOF. Le « Secondary NameNode » va donc

périodiquement faire une copie des données du « NameNode » afin de pouvoir prendre la relève en cas de panne de ce dernier (Figure B.10). Son rôle consiste, principalement, à :

- Télécharger régulièrement les logs sur le NameNode ;
- Crée une nouvelle image en fusionnant les logs avec l'image HDFS ;
- Renvoie la nouvelle image au NameNode.

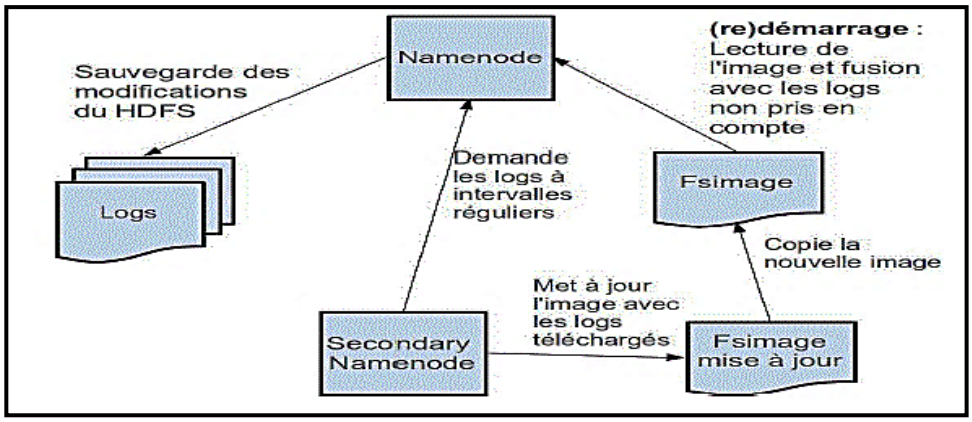


Figure B.10 : Fonctionnement du Secondary NameNode [85]

HDFS est conçu pour tourner sur des machines simples sous GNU/linux, et est programmé en Java. Toute machine qui dispose de Java peut donc faire tourner un NameNode ou un DataNode. Les nœuds communiquent entre eux via SSH. Il faut donc entrer la clé publique de chaque DataNode dans le fichier authorized_keys du NameNode, afin qu'il puisse se connecter aux autres nœuds via SSH sans avoir besoin de taper un mot de passe à chaque accés [83].

II.2.4.3 MapReduce et HDFS

Dans un écosystème d'Hadoop multi nœuds, on peut avoir sur une même machine physique l'exécution du JobTracker et du NameNode, aussi l'exécution sur une même machine physique d'un TaskTracker avec un DataNode (Figure B.11).

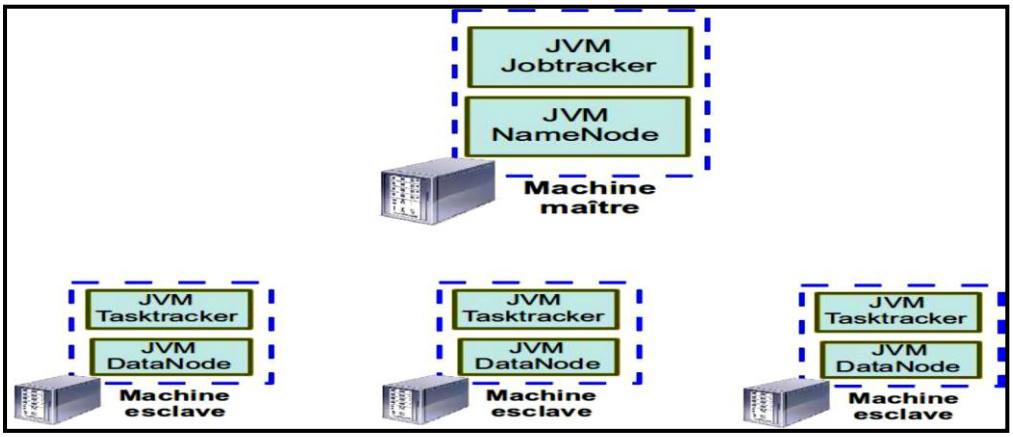


Figure B.11 : Architecture maître-esclave du MapReduce [85]

II.2.4.4 Lecture d'un fichier HDFS

La Figure B.12 illustre le processus de lecture dans un système de fichier HDFS :

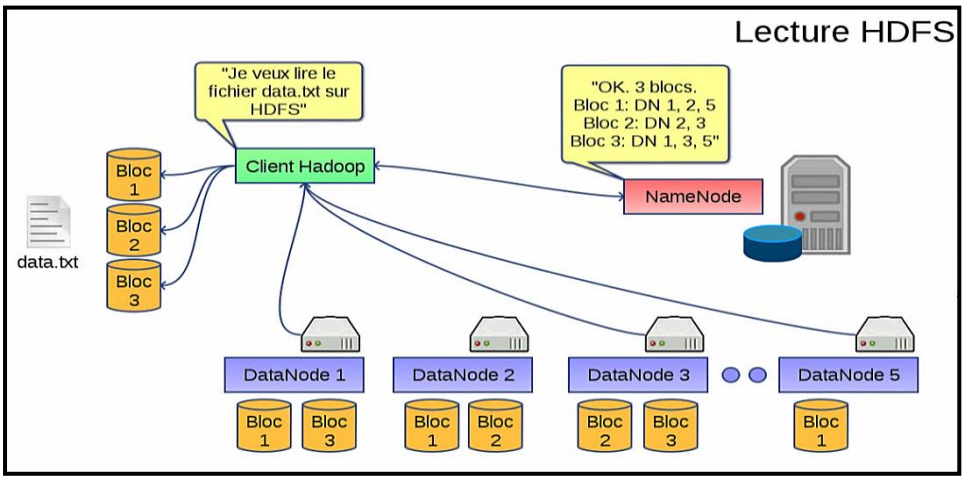


Figure B.12 : Lecture d'un fichier HDFS [86]

- Etape 1 : Le client indique au NameNode qu'il souhaite lire le fichier data.txt.
- Etape 2 : Le NameNode va lui transmettre la taille de fichier (nombre de blocs) ainsi que les différents DataNodes hébergeant les n blocs.
- Etape 3 : Le client tente de récupérerles blocs du fichier à un des DataNodes.
- Etape 4 : En cas d'erreur/non réponse d'un des DataNodes, il passe au suivant dans la liste fournie par le NameNode.

II.2.4.5 Ecriture dans un fichier HDFS

La Figure B.13 illustre le processus d'écriture dans un système de fichier HDFS :

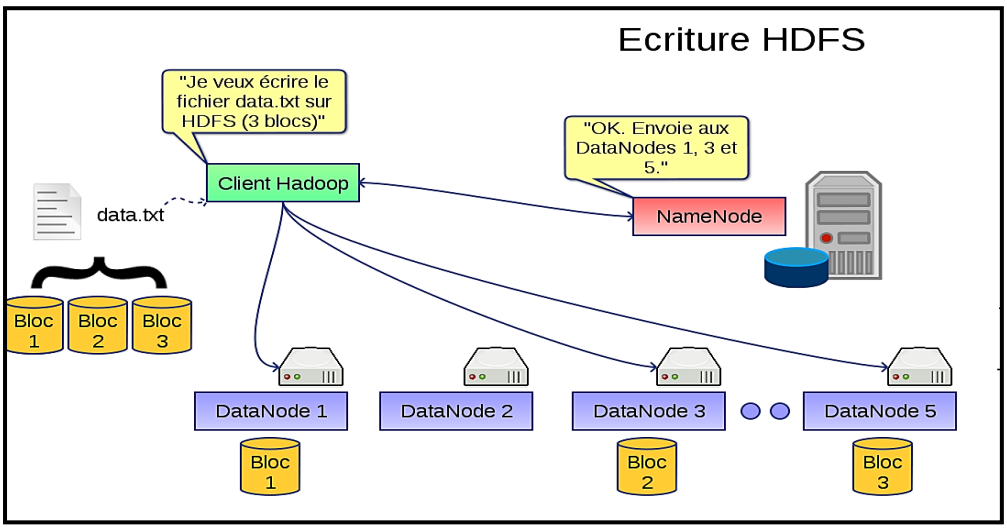


Figure B.13 : Ecriture dans fichier HDFS [86]

- Etape 1 : Admettons qu'on souhaite stocker le fichier data.txt sur HDFS : On va utiliser la commande principale de gestion d'Hadoop : *Hadoop*, avec l'option *fs*.

- Etape 2 : Le programme va diviser le fichier en blocs de 64 KB (ou autre, selon la configuration) – supposons qu'on ait ici 3 blocs.
- Etape 3 : Le NameNode lui indique les DataNodes disponibles.
- Etape 4 : Le client sollicite directement le DataNode concerné et lui demande de stocker le bloc.
- Etape 5 : les DataNodes s'occuperont – en informant le NameNode – de répliquer les données entre eux pour éviter toute perte de données.
- Etape 6 : Le cycle se répète pour le bloc suivant.

Notons que la stratégie de placement de bloc est décrite ci-dessous [86] :

- Une réplication sur un nœud aléatoire dans le même rack ;
- Deuxième réplication sur un rack distant ;
- Troisième réplication sur le même rack distant ;
- Réplication additionnelle placée aléatoirement ;

Le client lit la plus proche réplique.

II.2.4.6 HDFS et tolérance aux fautes

Pour éviter un crash du DataNode, HDFS doit multiplier :

- Les Heartbeats (détection par le NameNode) ;
- Le nombre de répliques.

Dans le cas d'un crash du NameNode, HDFS procède au :

- Sauvegarde des logs de transaction sur un support stable ;
- Redémarrage sur la dernière image du HDFS.

L'architecture d'Hadoop s'articule sur le NameNode qui représente un point unique de défaillance. L'arrêt du NameNode entrainera automatiquement l'arrêt d'HDFS [85], au moins pour lui permettre de redémarrer sur la dernière image fsimage qui a été créé par le Secondary NameNode (Figure B.14).

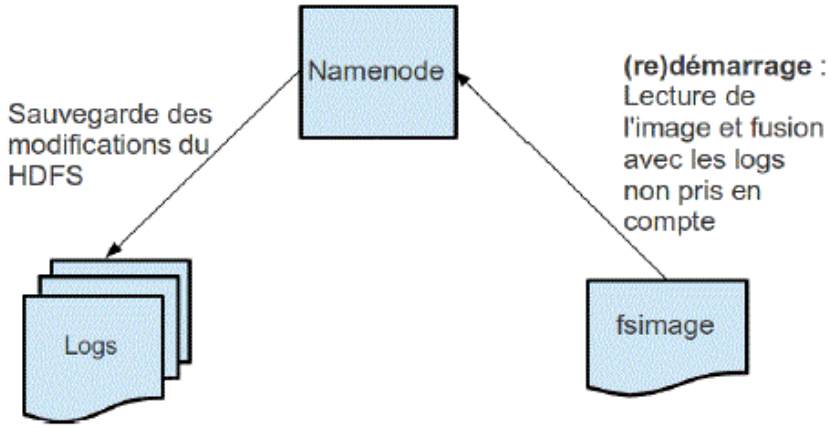


Figure B.14: Mécanisme de reprise d'activité du NameNode [85]

II.2.4.7 Service de métadonnées séparé

HDFS adopte un service de métadonnées séparé avec isolation et centralisation du service sur un serveur intitulé NameNode, hébergeant toutes les métadonnées. La centralisation est motivée principalement par la garantie de l'intégrité des métadonnées et la fiabilité du service qui exige une gestion rigoureuse et prise en charge sûre des métadonnées. L'isolation du service des métadonnées conduit naturellement à la séparation des flux de données et de métadonnées. Le client récupère auprès Name node le layout des blocs composant le fichier accédé, à savoir la liste des Data nodes. Puis les données circulent parallèlement entre clients et Data nodes. Le NameNode reste à l'écart du circuit des données [88].

II.2.5 Ecosystème d'Hadoop

Aujourd'hui il est difficile de se retrouver dans la " jungle " d'Hadoop du fait qu'il regroupe un ensemble de technologies jeunes en plein essor très poussés par une multitude de travaux de différents acteurs (spécialistes du web, start-up…) qui veulent prendre le train Big Data en marche.

Mise à part les différents éléments composants le noyau Hadoop (HDFS, MapReduce, YARN, HBase, Cassandra), plusieurs outils sont développés et intégrés constituant l'écosystème Hadoop [89].

II.2.5.1 Outils de Requêtage et de scripting des données

- **Hive (Facebook)** : Hive est à l'origine un projet de Facebook qui permet de faire le lien entre le monde SQL et Hadoop. Il permet l'exécution de requêtes SQL sur un cluster Hadoop en vue d'analyser et d'agréger les données. Le langage SQL est nommé HiveQL. C'est un langage de visualisation uniquement, c'est pourquoi seules les instructions de

type "Select" sont supportées pour la manipulation des données. Dans certains cas, les développeurs doivent faire le Mapping entre les structures de données et Hive.

- **Pig (Yahoo) :** il apporte un modèle de développement de plus haut niveau, et donc beaucoup plus expressif et simple à appréhender, afin de démocratiser l'écriture de traitements MapReduce. Pig se rapproche plus d'un ETL où on part d'un ou plusieurs flux de données que l'on transforme étape par étape jusqu'à atteindre le résultat souhaité. Les différentes étapes de la transformation sont exprimées dans un langage procédural (Pig Latin). Pig est à l'origine un projet de Yahoo qui permet le requêtage des données Hadoop à partir d'un langage de script.

  Contrairement à Hive, Pig est basé sur un langage de haut niveau Pig Latin, qui permet de créer des programmes de type MapReduce. Il ne dispose pas d'interface web. Pig se base sur HDFS et MapReduce pour exécuter le script en en arrière-plan.

II.2.5.2 Outil d'intégration SGBDRelationnel

Sqoop permet le transfert des données entre un cluster Hadoop et des bases de données relationnelles. C'est un produit développé par Cloudera qui permet d'importer/exporter des données depuis/vers Hadoop et Hive. Sqoop utilise MapReduce et des drivers JDBC, pour la manipulation des données.

II.2.5.3 Outils de gestion et de supervision du cluster Hadoop

- **Apache ZooKeeper** : ZooKeeper est un service de coordination des services d'un cluster Hadoop. En particulier, le rôle de ZooKeeper est de fournir, aux composants Hadoop, les fonctionnalités de distribution. Pour y faire, il centralise les éléments de configuration du cluster Hadoop, propose des services de clustérisation et gère la synchronisation des différents éléments (événements) [89].

  ZooKeeper est un élément indispensable au bon fonctionnement d'Hbase.

- **Apache Ambari (HortonWorks)** : Ambari est un projet d'incubation Apache initié par HortonWorks et destiné à la supervision et à l'administration de clusters Hadoop. C'est un outil web qui propose un tableau de bord qui permet de visualiser rapidement l'état d'un cluster [89].

  Ambari dispose d'un tableau de bord dont le rôle est de fournir une représentation de :

  - L'état des services ;
  - La configuration du cluster et des services ;
  - L'exécution des jobs ;

- Métriques de chaque machine et du cluster.

II.2.5.4 Outil d'ordonnancement et de coordination

Apache Oozie est une solution de workflow développée par Yahoo au sens d'ordonnanceur (scheduler) d'exploitation, utilisée pour gérer et coordonner les tâches de traitement de données à destination d'Hadoop.Oozie s'intègre parfaitement avec l'écosystème Hadoop puisqu'il supporte les types de jobs suivants :

- MapReduce ;
- Pig ;
- Hive ;
- Sqoop ;
- Jobs spécifiques du système tels que des programmes java et des scripts Shell.

II.2.5.5 Outil de collecte et d'agrégation de fichiers logs

Apache Flume, est une solution de collecte et d'agrégation de fichiers logs, destinés à être stockés et traités par Hadoop. Il a été conçu pour s'interfacer directement avec HDFS au travers d'une API native. Flume est à l'origine un projet Cloudera, reversé depuis à la fondation Apache [89].

La figure suivante illustre la distribution Cloudera de l'écosystème Hadoop (CDH) :

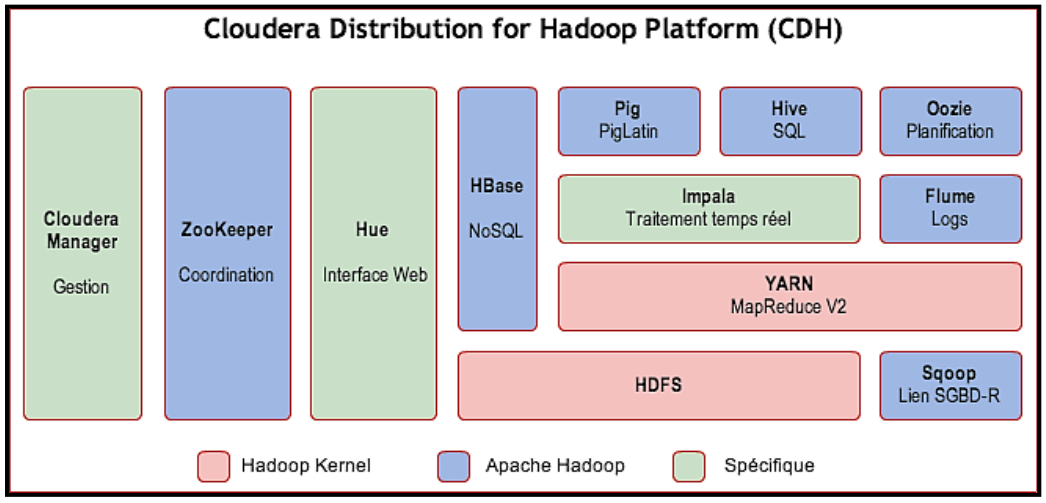


Figure B.15: Ecosystème d'Hadoop [89]

II.3 Traitement des métadonnées

Un système de gestion des données, pour fonctionner et assurer ses objectifs doit pouvoir retrouver les métadonnées.

II.3.1 Classification des métadonnées

Dans les environnements distribués comme ceux des clouds, les métadonnées peuvent être classées en quatre groupes principaux :

II.3.1.1 Métadonnées propres aux données

Les métadonnées intrinsèques sont propres aux données, indépendamment de l'outil qui les manipule : La taille, la date de création, la date de dernier accès, la date de dernière modification, l'espace alloué, le type de données (image, texte, vidéo). Ces informations sont parfois qualifiées d'attributs de fichier [76].

II.3.1.2 Métadonnées propres au système

Les métadonnées induites par le système de gestion de données qui les manipule, dans un environnement multi-utilisateur : Le propriétaire des données, liste des utilisateurs, les droits d'accès des autres utilisateurs, le comportement en cas d'accès simultanés, la méthode de récupération après incident, etc. Ces métadonnées sont propres à un système d'exploitation et au produit de manipulation des données, et ne sont donc pas identiques en nature d'une machine à une autre.

II.3.1.3 Métadonnées propres à l'application

Les métadonnées utilisées par une application particulière permettant de décrire le contenu des données en fonction des besoins de l'application, de les référencer par un identifiant propre à l'application. Ce groupe de métadonnées est bien entendu propre à l'application.

II.3.1.4 Les informations sur le stockage

La description de la localisation des données sur le support de stockage fait couramment partie des métadonnées. Pour les clouds, la liste des serveurs et leurs adresses réseau fait partie de la notion de layout. De même pour la quantité d'espace de stockage libre restant sur chaque serveur de stockage de données, qui est nécessaire pour les fonctionnalités d'équilibrage de charge, peut être considérée comme une métadonnée, sans pourtant être spécifiquement liée aux données.

II.3.2 Séparation des voies de données et de métadonnées

Un système distribué peut être modélisé en combinant les deux différents composants :

- L'état du système (System state) représenté par les méta-données, cruciales pour le bon le fonctionnement du système.

- L'état de l'application (Application state) représenté par les applications spécifiques ou les données stockées par le système.

Plusieurs travaux approuvent que le découplage des deux états se considère comme la cause principale de la grande scalabilité des systèmes modernes sur Internet [90].

Alors, le principe des Clouds requiert l'évitement du goulot d'étranglement que constitue le traitement des métadonnées, afin de satisfaire les objectifs de performance et d'extensibilité. Autant la circulation des données peut être accéléré par la parallélisation des flux de données, autant le traitement des métadonnées est séquentiel afin de garantir la cohérence et l'intégrité de ces métadonnées. Le principe est donc le découplage du traitement des données à celui de métadonnées. Les données et les métadonnées empruntent des chemins complètement séparés. Le serveur ayant en charge le traitement des demandes d'accès, les contrôles d'accès, le déclenchement des transferts de données, les informations de contrôle, le traitement du protocole de service de données et toutes les métadonnées circulent dans une voie en dehors du chemin de données, alors que les données sont transportées dans une autre voie.

Les applications clients accèdent directement aux données dans les serveurs de stockage. Les chemins entre les clients et les serveurs de stockage peuvent être multipliés à grande échelle. Le parallélisme des Clouds de stockage est donc un parallélisme de transfert de données, alors que le traitement des métadonnées reste en grande partie séquentiel. Les architectures les plus répandues dans les environnements Clouds et Big Data comme celle d'Hadoop, adoptent un service centralisé des métadonnées pris en charge par un serveur.

II.3.3 Influence du traitement des métadonnées sur la performance

Le volume des métadonnées représente en moyenne 1 à 2 % de la totalité de l'espace de stockage des données. Un volume non négligeable, qui nécessite des réflexions pour l'accélération de l'accès aux supports physiques de métadonnées. De plus les échanges de métadonnées dans le fonctionnement des Clouds de stockage sont très fréquents. Le client sollicite le serveur des métadonnées pour recevoir les layouts, les indicateurs de remplissage des serveurs de stockage, les indications sur l'état des connexions réseau et les métadonnées propres aux données. Rappelons que l'augmentation du parallélisme dans tous les traitements informatiques engendre directement une augmentation du nombre de métadonnées accédées simultanément. La disposition du fichier sur de multiples espaces de stockage provoque une multiplicité de métadonnées pour ce seul fichier. Les mouvements des métadonnées recouvrent potentiellement plusieurs échanges. Pour chaque opération, la nature de cette

dernière détermine le nombre de requêtes réseau nécessaires et la durée de verrouillage de la métadonnée modifiée. En conséquence, la performance dans le traitement des métadonnées se voit très influente dans la performance dans le système global. Ce qui hisse l'objectif de performance dans le traitement des métadonnées comme problématique de premier ordre pour les chercheurs dans le domaine [76].

Chap III : Approche hybride

HDFS est basée sur une architecture simple et sûre, mais qui peut atténuer relativement la performance et l'extensibilité des clouds de stockage et des Big Data. Afin de préserver cette cohérence des métadonnées qualifiée comme primordiale dans un système de gestion de données, sans trop compromettre la performance et l'extensibilité, nous proposons une solution mixte entre centralisation et distribution du service des métadonnées. Cette manière qui consiste à décomposer les métadonnées pour améliorer les performances est également utilisée par l'architecture Lambda [91] qui divise le système en trois couches.

III.1 Parallélisme modéré des métadonnées

La complexité vient du fait que les utilisateurs sont également géographiquement distribués : une centralisation excessive des métadonnées risque d'engorger l'accès à l'infrastructure réseau et au serveur de métadonnées. Il est peut-être intéressant de scinder les métadonnées en sous-ensembles qui sont gérés différemment aussi bien pour la localisation physique que pour le nombre d'instances. Les métadonnées nécessaires à la navigation des utilisateurs, portant des informations sur les données proprement dites (II.3.1.1) étant employées plus fréquemment que celles propres à l'accès comme les droits d'accès et la localité de données (II.3.1.2), (II.3.1.3) et (II.3.1.4). La description des données (taille, date de création, date de dernier accès, date de dernière modification) peut être rapprochée des utilisateurs et répliquée suivant les besoins comme dans [92], tout en centralisant les métadonnées les plus sensibles.
L'accélération du traitement des métadonnées se fait potentiellement par délégation d'une partie du service de métadonnées aux serveurs de stockage.
Cela servira aussi à l'adaptation de la présentation des données aux multiples interfaces immanentes des Clouds en prenant en compte la résidence des métadonnées permettant la navigation vis-à-vis des utilisateurs.

III.2 Description de l'approche

L'architecture proposée est une variante de HDFS, en déployant les mêmes composants d'Hadoop à savoir :

- Job tracker : un aiguilleur de tâches centralisé
- Task trackers : des gestionnaires de traitements locaux aux serveurs
- Data node : les nœuds de stockage de données et métadonnées associées
- Name node : un serveur central de métadonnées communes

- Secondary Name node : un serveur de secours de métadonnées communes

Ces mêmes unités sont reconduites mais avec un remaniement interne et de nouvelles tâches seront affectés au Job tracker suivant les principes suivants :

III.2.1 Métadonnées communes

Les métadonnées communes les plus sensibles du système seront concentrées sur un Name node, une machine super puissante, dopée d'une mémoire vive importante et d'une bonne mémoire cache pour avoir de meilleures performances en débit et en transactions par seconde, équipée de disques SSD (Solide State Disk) pour remédier à la lenteur des disques magnétiques.

III.2.2 Métadonnées spécifiques

Les métadonnées nécessaires à la navigation des utilisateurs qui renseignent sur les données proprement dites, peuvent être rapprochées des utilisateurs sur les Data node hébergeant les données associées.

III.2.3 Fonctions reconduites de l'aiguilleur des tâches

Les fonctions de base de l'aiguilleur des tâches conservées :

- Réception des travaux soumis par les applications clientes
- Demande de la localisation de la donnée auprès du Name node
- Localisation des nœuds Task trackers
- Affectation des tâches aux Task trackers : les nœuds hébergeant les données, si le travail ne peut pas être accompli sur le nœud où se trouvent les données, la priorité est donnée à des nœuds dans le même rack pour réduire le trafic réseau)

III.2.4 Nouvelles fonctions de l'aiguilleur des tâches

De nouvelles fonctions viennent s'ajouter aux fonctions précédentes du Job tracker, comme il est décrit dans l'algorithme (ModPara) :

Pour les métadonnées communes (Liste des utilisateurs, les droits d'accès, le comportement en cas d'accès simultanés, le layout, les informations de saturation des serveurs de stockage, les indications sur l'état des connexions réseaux) nécessaires pour répondre à l'application cliente le Job tracker les solliciteront auprès du Name node, contrairement aux métadonnées sur les données proprement dites (taille, date de création, date de dernier accès, date de dernière modification), requises pour la même application cliente seront recherchées auprès des Data node par le biais des Task trackers associés. Si le traitement va être accompli au

niveau du même Task tracker associé au Data node hébergeant la donnée, les métadonnées récupérées par le Job tracker du Name node seront envoyés au Task tracker pour s'ajouter aux autres métadonnées déjà présentes dans ce nœud. Si le traitement va se faire dans un autre nœud, le Job tracker va consolider les métadonnées recherchées auprès du Name node avec celles récupérées du nœud défaillant, au niveau de son mémoire cache, pour les transmettre en même temps avec les données correspondantes au nouveau nœud le plus proche du nœud défaillant pour ne pas éloigner le nœud qui va traiter les données aux nœuds qui héberge ces mêmes données.

III.3 Algorithme : ModPara

Soient mdc : métadonnées communes

```
      mds : métadonnées spécifiques
      a,b : variables
1: Begin
2: for each work(clients)
3: do search(name_node)
      4: if   etat(name_node)=failure
            5: then a ← secondary_name_node.mdc
            6: else a ← name_node.mdc
      7:   endif
      8:   job_tracker ← a
      9:   search(task_tracker)
      10: search(data_node)
      11: b ← data_node.mds
      12: if   etat(task_tracker)=failure
            13: then job_tracker ← a+b
            14: search(task_tracker_near)
            15: processing(task_tracker_near)
            16: job_tracker ← task_tracker_near.results
            17: data_node ← job_tracker.results
            18: else task_tracker ← a+b
            19: processing(task_tracker)
            20: data_node ← task_tracker.results
      21: endif
22: enddo
23: client ← results
24: endfor
25: end.
```

Explication :

2 : Pour toute tâche émise par une application cliente

3 : Rechercher les métadonnées communes du NameNode

4 : Si le serveur central de métadonnées est défaillant

5,6 : Les métadonnées communes sont récupérées du serveur central de métadonnées ou du serveur secondaire de métadonnées et affectées à « a »

9,10 : Localisation du gestionnaire de traitement local et le nœud de stockage hébergeant les données sollicitées

11 : Les métadonnées spécifiques sont affectées à « b »

12 : Si le gestionnaire de traitement local est défaillant

13 : Les métadonnées communes et spécifiques sont cumulées et affectées à l'aiguilleur de tâches

14 : Chercher le gestionnaire de traitement local le plus proche des données dans le réseau

15 : Le traitement est fait par le gestionnaire de traitement local le plus proche de celui qui est défaillant

16 : Les résultats sont retournés à l'aiguilleur de tâches

18,19 : Sinon le traitement est fait par le gestionnaire de traitement associé au nœud de stockage contenant ces données

17,20 : Dans les deux cas, les résultats sont conservés dans le nœud de stockage correspondant.

III.4 Architecture

III.4.1 Architecture du modèle proposé

La structure de la solution proposée basée sur le modèle HDFS est schématisée dans la Figure B.16.

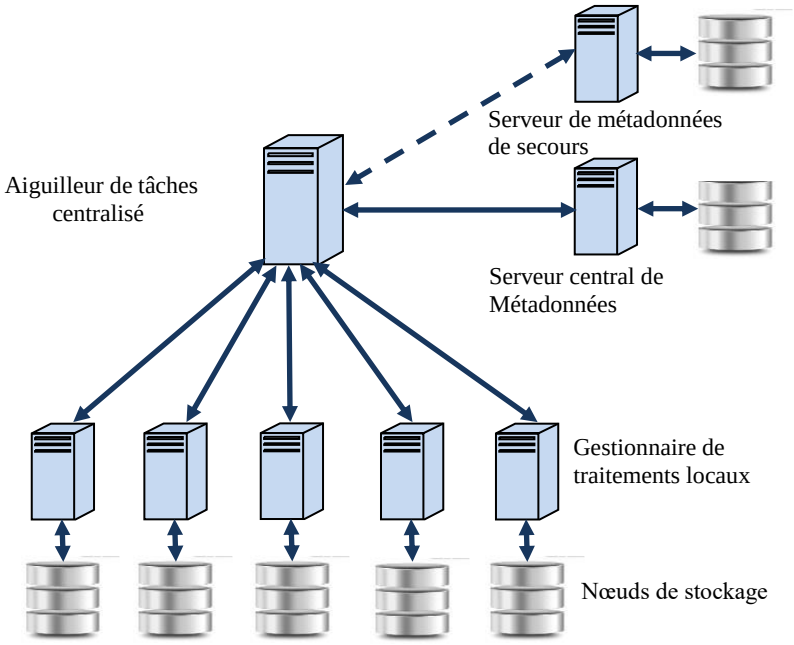


Figure B.16: Structure du modèle proposé

III.4.2 Architecture fonctionnelle

Une fois une nouvelle tâche est affectée au job_tracker, les métadonnées communes sont recherchées auprès du name_node. Si le name_node est défaillant alors les métadonnées sont récupérées du secondary_name_node. Le job_tracker localise le task_tracker et le data_node hébergeant les données demandées et les métadonnées spécifiques sont récupérées aussi. Si le task_tracker est défaillant, alors les métadonnées communes et spécifiques sont cumulées et affectées au job_tracker, qui va solliciter un autre task_tracker dans le voisinage réseau des données. Alors, une fois le traitement fait par ce nouveau task_tracker, les résultats sont retournés au job_tracker. Dans le cas contraire, le traitement est assuré par le task_tracker associé au data_node ou les données sont localisées. Dans les deux cas, les résultats sont retournés à l'application cliente et conservés dans le nœud de stockage correspondant.

La latence peut être atténuée et le débit peut être amélioré par le parallélisme employé au niveau du service des métadonnées spécifiques tout en conservant la cohérence des métadonnées les plus sensibles du système, en les isolants dans un serveur central. Par conséquent, l'extensibilité sera multipliée en raison de la distribution de métadonnées sur les différents serveurs en remplacement du serveur central.

L'architecture fonctionnelle du modèle proposé est illustrée dans la Figure B.17 :

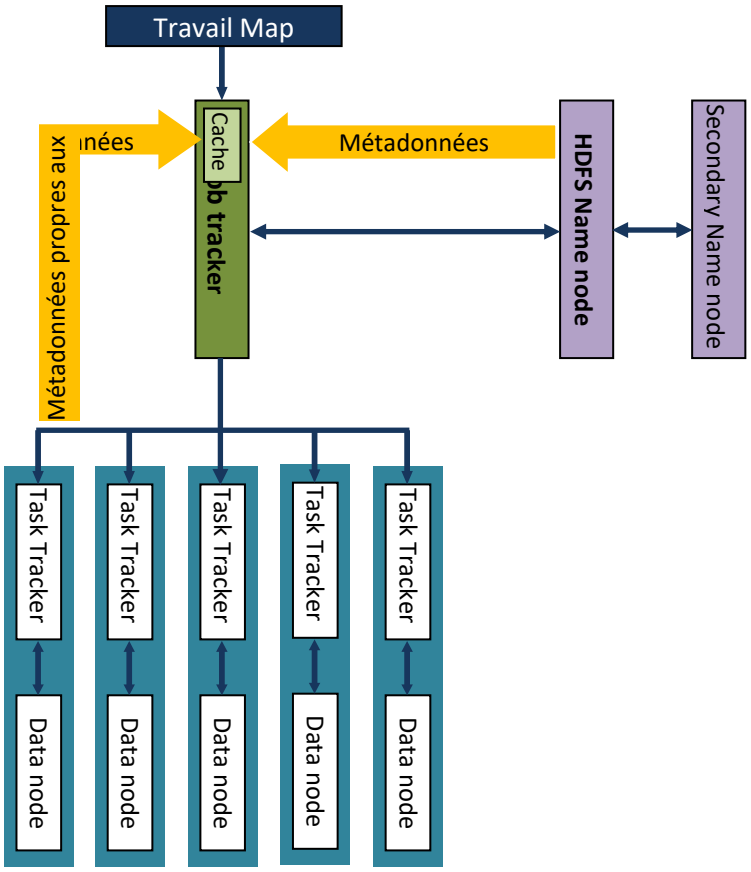


Figure B.17: Architecture fonctionnelle du modèle proposé

III.5 Travaux connexes

D'autres modèles et architectures de stockage et d'analyse de données dans les environnements à grande échelle peuvent exister. Des approches qui n'emploient pas forcément un parallélisme des traitements et de données comme dans les Clouds, mais peuvent utiliser un ou des serveurs uniques très puissants. D'autres se basent sur des bases de données relationnelles [76]. Les études montrent que malgré que certains projets visent d'ajouter les caractéristiques de cloud dans les systèmes de gestion de bases de données relationnelles comme (Elas-TraS, DBonS3, Project Deuteronomy, Relational Cloud, epiC), ces derniers ne sont pas susceptibles d'être déployés dans le cloud, au moins dans un proche avenir [22]. D'autres architectures distribuées émergent connues par bases de données parallèles sans partage (Shared-Nothing Parallel Databases) dans lesquelles chaque nœud est indépendant et autonome. Aucune concurrence, ni de conflit au sein du système, aucun partage de mémoire ou de disque entre les nœuds n'est appliqué. Plusieurs solutions se trouvent actuellement dans le marché : Teradata, Netezza, Green- prune, DATAllegro, Vertica et Aster Data utilisent une architecture Shared-Nothing dans leurs produits SGBD analytiques au moins dans la couche

de stockage et récemment IBM DB2 et Oracle ont ajouté aussi Shared-nothing dans leurs produits analytiques [22].

Autres travaux de recherches dans le même contexte de stockage de données dans les grands environnements traitent l'aspect diversifié des types de données structurés, semi-structurées et non structurées. Plusieurs approches ont abouti à la popularité des bases de données NoSQL [81]. NoSQL désigne une catégorie de systèmes de gestion de base de données (SGBD) qui n'est plus fondée sur l'architecture classique des bases relationnelles. L'unité logique n'y est plus la table, et les données ne sont en général pas manipulées avec SQL. À l'origine servant à manipuler des bases de données géantes pour des sites web de très grande audience tels que Google, Amazon.com, Facebook ou eBay. Le NoSQL renonce aux fonctionnalités classiques des SGBD relationnels (Propriétés transactionnelles ACID) au profit de la simplicité. Les performances restent bonnes avec la montée en charge (scalabilité) en multipliant simplement le nombre de serveurs. Les systèmes géants sont les premiers concernés : énorme quantité de données, structuration relationnelle faible (ou de moindre importance que la capacité d'accès très rapide, quitte à multiplier les serveurs). Cette conception a permis de monter en charge à des pétaoctets de données et des milliers de requêtes simultanées. Plusieurs solutions à base de cette architecture, existent dans le marché, citons Bigtable (Google), Pnuts (Yahoo) et Dynamo (Amazon) et leurs homologues en Open-Source Cassandra (Facebook, Twitter), Hbase (Facebook) et Voldemort (Linkedln) [6,7]. La majorité de ces produits s'appuient sur Hadoop et ses composants et sont conçus sur des architectures parallèles.

Certainement, les systèmes qui emploient le parallélisme massif sont les plus adéquats aux environnements Clouds et BigData. On peut distinguer trois manières de parallélisations :

- Distribution des métadonnées : Plusieurs implémentations ont été proposées comme Scality, Lustre, Ceph et Panasas [76].
- Centralisation des métadonnées : L'implantation de référence du Cloud et Big Data Hadoop [77,78]. Plusieurs acteurs industriels suggèrent une intégration ou une offre Hadoop comme MapR, Cloudera, Hortonworks [76].
- Implantation hybride : Centralisation des métadonnées intrasites et P2P intersites. XtreemFS est un exemple typique de cette catégorie [76].

Notre approche vient se placer entre la deuxième et la troisième manière de parallélisation. C'est le parallélisme massif des données d'une part, et la centralisation en partie des métadonnées et la décentralisation de la deuxième partie des métadonnées. C'est une continuité dans les travaux effectués sur Hadoop dans laquelle nous apportons des

modifications sur son système de gestion de fichiers HDFS, à savoir une redistribution des métadonnées.

Conclusion

Les Clouds de stockage constituent l'architecture qui répond aux accroissements massifs des données que ce soit en volume, en nombre ou en type. Les quantités d'informations et les volumétries à traiter induisent un parallélisme massif et le découplage du traitement des données à celui de métadonnées. Dans cette deuxiéme partie du manuscrit, nous avons rappelé les différents systèmes de gestion de fichiers employés dans les différentes architectures mettant en relation un ou plusieurs ordinateurs, le principe des métadonnées a été aussi présenté, ainsi que les diverses implantations des clouds de stockage et principalement celle d'Hadoop ont été évoqué.

L'architecture proposée dans cette partie B (Figure B.17), se place entre :

- Une architecture qui se base sur un parallélisme massif de données et métadonnées, comme dans P2P, d'un côté.
- Une architecture appuyée par un parallélisme massif de données et une centralisation de métadonnées, comme dans HDFS, de l'autre côté.

Toutefois toutes les approches d'amélioration des systèmes de gestion de données dans les environnements des big data et des clouds de stockage seront très appréciées dans le futur.

Conclusion générale

Conclusion

Avec la domination d'Internet dans la majorité des secteurs, industriels, medias, communication, et même familial, les technologies de Big Data et Cloud Computing sont aujourd'hui en plein essor. Dans les prochaines années, nous estimons que ces technologies seront de plus en plus utilisées pour répondre à de nouvelles problématiques pour la gestion de données. Très logiquement, les systèmes NoSQL répondent favorablement aux changements d'échelles et se considèrent, actuellement, comme les solutions idéales pour gérer ces grands volumes de données qui caractérisent le Big Data.

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. Divers modèles NoSQL basées sur des architectures différentes sont conçus, développés et déployés dans les différents secteurs. L'absence de normalisation est un aspect marquant de cette mouvance NoSQL et la panoplie de solutions existantes dans le marché, mettent les différents décideurs devant un embarras dans le choix du modèle approprié par rapport à leur environnement d'exploitation.

Dans cette optique qui vise de justifier et motiver un tel basculement du SQL vers le NoSQL, notre thèse est parvenue dans sa premiére partie, à apporter des éléments de réponse dans les éventuels choix du système NoSQL adéquat au type de données manipulées et au type de traitement exécuté sur ces données. L'étude consistait à l'évaluation de performances de six solutions NoSQL : MongoDB, CouchBase, Cassandra, HBase, Redis et OrientDB.

En conclusion de la premiére partie de la thése, on peut retenir que le choix d'utilisation d'un SGBD dépend d'un ensemble de paramètres en relation avec l'environnement dans lequel les données sont exploitées. En effet le type de données et le type des traitements effectués sur ces données sont des indices importants pour définir la solution à adopter ainsi que les besoins spécifiques qui peuvent varier d'une entreprise à une autre. La fréquence estimée de lecture, d'écriture, de mise à jour ainsi que la taille des données sont les facteurs essentiels déterminants dans le choix d'une alternative parmi d'autres. Actuellement la tendance vers une favorisation d'une solution NoSQL précise est loin d'être indiscutable à cause du nombre important des systèmes existants. Plusieurs solutions open-source et payantes sont présentées aux différents acteurs concernés. En effet, Il n'y a pas de solution parfaite et il n'y a pas de mauvaises solutions de gestion de données NoSQL. 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.
En fin de cette premiére partie A, nous tenons à souligner, qu'une partie de cette étude qui s'est rétrécie à comparer MongoDB avec HBase, a été synthétisée dans un article scientifique publié dans le journal international spécialisé dans le domaine IJCSSE [93].

Dans la deuxiéme partie du manuscrit, nous nous intéréssons à l'implantation de référence des clouds de stockage HDFS d'Hadoop. HDFS s'appuie sur la séparation des métadonnées aux données avec isolation et centralisation des métadonnées par rapport aux serveurs de stockage de données. Dans ce contexte, nous proposons une approche qui permet de rétablir le service de métadonnées d'Hadoop, en suggérons une solution mixte entre centralisation et distribution des métadonnées, afin d'améliorer la performance et l'extensibilité du modèle.
L'approche proposée, emploie un parallélisme modéré des métadonnées, en renforçant le parallélisme massif des données, qui est la clé de réussite des architectures modernes et délègue une partie de la responsabilité des métadonnées aux serveurs de stockage. Le parallélisme massif qui s'étend aux métadonnées peut affecter la fiabilité du service et la cohérence de ces informations. La centralisation du service des métadonnées dans une seule machine peut constituer un goulot d'étranglement qui peut dégrader les performances du système global dans le futur, vu l'accroissement massif très rapide des données qui engendre automatiquement un autre accroissement massif des métadonnées.
Dans cette optique nous avons présenté notre architecture hybride HDFS avec un remaniement interne et de nouvelles fonctions affectées aux composants HDFS. Reste à implémenter, tester et comparer les résultats avec le modèle de référence Hadoop en premier lieu et améliorer les performances de l'approche proposée en s'appuyant sur les travaux de Kishor et Venkateswarlu [94] en deuxième lieu.
Notons enfin, que le modèle proposé a fait l'objet d'un article scientifique publié dans un journal international spécialisé dans le domaine [95].

Perspectives

Gardez à l'esprit que nous sommes encore à un stade précoce de la prise en charge parfaite des énormes volumes de données produits, hébergés et traités pour obtenir une vue à 360 degrés du marché et anticiper les changements et les renversements dans les besoins des clients.
En perspectives de l'étude développée dans la partie A, nos prochaines études d'évaluations, vont consister à passer à des échelles plus élevées avec l'augmentation du nombre d'opérations

effectuées pour atteindre des Workloads dépassant 20 000 opérations, et le nombre d'enregistrements insérés pour avoir une grande base de données test hébergeant des millions d'enregistrements.

Notons aussi, que notre étude comparative s'est articulée sur une architecture Single Node, les tests et les évaluations des performances dans une architecture distribuée en multipliant les nœuds progressivement dans un environnement distribué, peuvent faire l'objet d'autres travaux dans le futur.

Nous prévoyons aussi de développer d'autres études comparatives qui opposent les systèmes NoSQL avec les solutions SQL utilisées dans les environnements Cloud Computing ainsi qu'avec les nouveaux systèmes NewSQL, peuvent faire l'objet d'autres études comparatives futures.

Dans la partie B et en perspectives de la nouvelle architecture proposée, nos objectifs futurs seront focalisés sur la proposition d'une nouvelle technique de stockage en s'inspirant du travail réalisé par Jain, Chaudhary, et Bhatnagar [96].

L'organisation des métadonnées sous forme de base de données relationnelle ou orientée objet, la hiérarchisation des métadonnées peuvent être très utile pour améliorer le service des métadonnées. Ainsi, les métadonnées peuvent bien être organisées et stockées dans ces modèles en raison de leurs volumes réduits par rapport aux données et de leurs types et tailles connues préalablement. Une structuration des métadonnées stockées dans le serveur central de métadonnées (NameNode) d'HDFS, peut être proposée, pour bénéficier des différentes techniques d'accès et de stockage des données en mémoire, utilisées dans le domaine, à savoir hachage, B-arbre et fichiers indexés.

Autres traitements particuliers de certaines métadonnées peuvent constituer des pistes de recherche intéressantes, à savoir la préservation de la cohérence des métadonnées qui peut être atteinte à cause des duplications des serveurs centrales de métadonnées et l'utilisation de caches chez les clients.

Néanmoins, face au déluge de données, il est nécessaire de se poser la question de conservation de toutes ces données manipulées. La notion de qualité de l'information et l'analyse des Big Data peut couvrir un axe de recherche de plus en plus important. Idéalement, les avantages de scalabilité de MapReduce pourraient être combinés avec les performances et l'efficacité des avantages des bases de données parallèles pour parvenir à un système hybride bien adapté pour le marché des SGBD analytiques qui peut prendre en charge les exigences futures des applications de données intensives [97, 98, 99].

Annexes : Déploiement de l'environnement Soft

Annexe 1 : YCSB

Le déploiement d'YCSB requiert l'installation de Java, Maven et Git.

1.1 Java

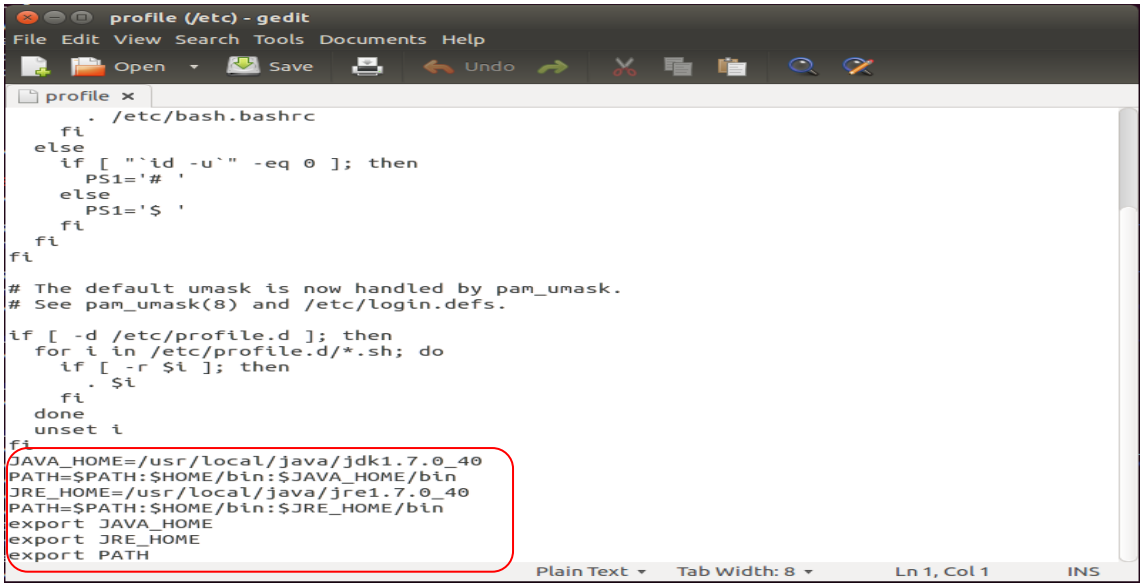
1. Téléchargement des archives binaires d'Oracle Java JDK et JRE du site : http://www.oracle.com/technetwork/java/javase/downloads/index.html
2. Copiage des archives binaires d'Oracle Java du répertoire : /home/"nom_d'utilisateur"/Téléchargements.

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

3. Extraction des archives binaires de Java dans le répertoire /usr/local/java :

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

4. Connexion en tant que root et ajout de ce qui suit dans le fichier descripteur de variables systèmes**.** /etc/profile :



5. Information d'Ubuntu de l'emplacement d'Oracle Java JDK/JRE et indication au système que la nouvelle plateforme Java est disponible :

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

6. Définitionde l'environnement Java par défaut d'Ubuntu par Oracle Java :

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

7. Prise en compte de la variable PATH par le système :

```
. /etc/profile
```

1.2 Maven

1. Téléchargement d'apache Maven du sitehttps://maven.apache.org/download.cgi
2. Exécution des commandes suivantes :

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

1.3 Git

– Installation et configuration de Git :

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

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

1.4 YCSB

1. Test du bon fonctionnement de toutes les dépendances :

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

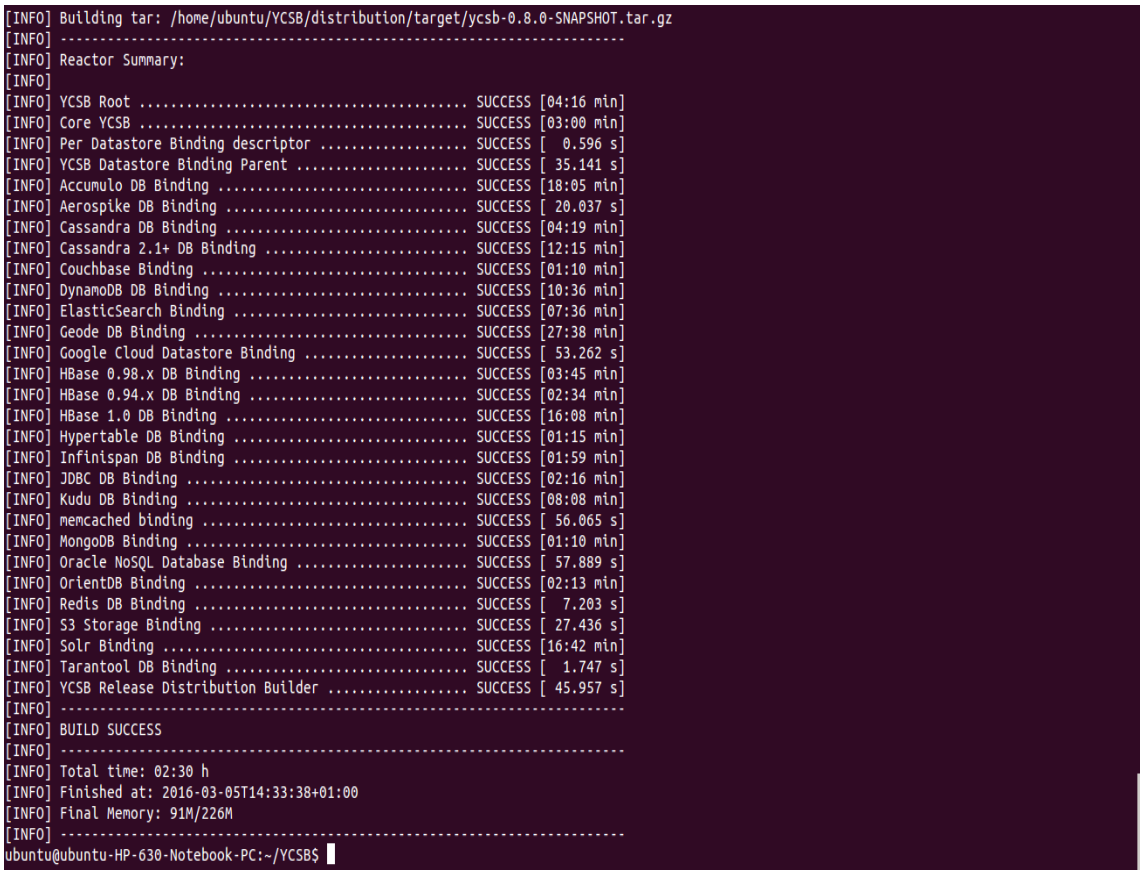
2. Téléchargement de la source YCSB :

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

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

3. Compilationd’YCSB :

```
mvn clean package
```


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

Annexe 2 : MongoDB

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

1. Importation la clé publique de 10gen dans notre système :

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

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

2. Ajout de l'url de MongoDB dans /etc/apt/sources.list.d/mongodb.list.:

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

3. Installation de MongoDB :

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

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

4. Vérification de la version de MongoDB :

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

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

5. Lancement de mongo shell pour interroger les donnéespar l'option « mongo » :

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

6. Arrêt de MongoDB :

```
sudo service mongod stop
```

Annexe 3 : CouchBase

1. Installation de libssl :

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

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

2. Exécution dela commande :

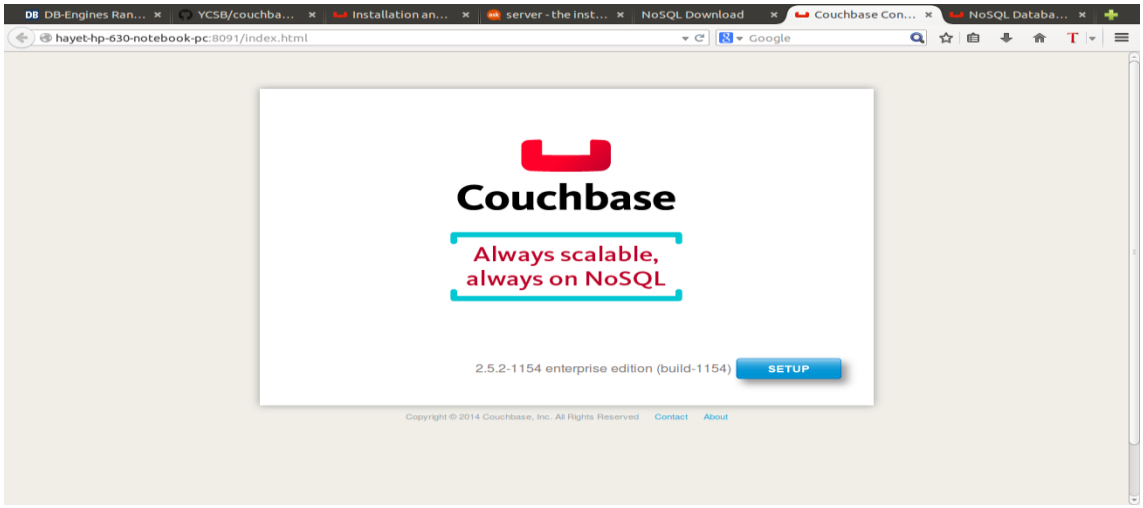
```
sudo dpkg -i couchbase-server-enterprise_2.5.2_x86.deb
```

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

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

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

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

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

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

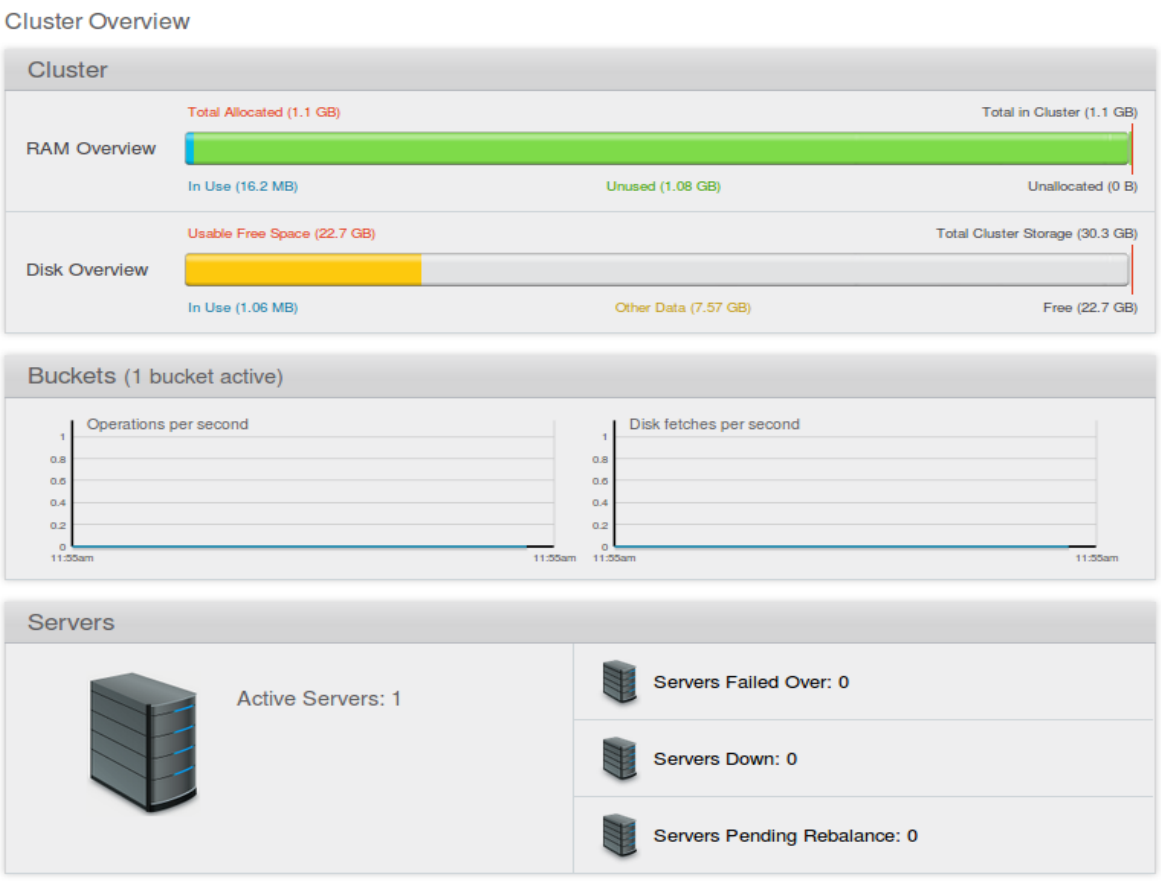
3. Démarrage de CouchBase :

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

4. Accès à l'interface graphique :



5. Après création d'un 'Default Buket' et configuration du serveur, l'interface qui permet de visualiser toutes les informations nécessaires, s'affiche :



Annexe 4 : Cassandra

1. Téléchargement de Cassandra du site : http://cassandra.apache.org/download/

2. Extraction du fichier téléchargé dans / tmp et relocalisation du fichier dans Home :

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

3. Création des répertoires suivants :

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

4. Modification du fichier de configuration « cassandra.yaml »:

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

5. Définition du répertoire Log :

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

6. Démarrage de Cassandra :

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

7. Vérification de la connexion à l'instance Cassandra avec nodetool :

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

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

Datacenter: datacenter1
==========
Address    Rack        Status State   Load            Owns                Token
                                                                          9052959872453055328
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -9109978298695564668
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -9103208520055629798
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -9088728925272495462
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8990953282109418954
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8879162094196454305
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8818043954455636578
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8730166131524687185
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8672826070244241237
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8672034074208567492
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8513108959011133362
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8509321140474918759
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8401836198234140580
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8384904085193139809
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8322552040103674892
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8317759320813479851
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8311917673697818931
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8168766447481534252
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -8117779682610303280
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7995188559701218116
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7993228157876842222
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7929079150165027774
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7860179453034889017
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7787377756692757750
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7719463177043259343
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7540481259536583977
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7451877945368807830
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7394324980012103665
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7226891571066326829
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7221753625496618653
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7124490554840420174
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7122743112580771342
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -7092350078853911637
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -6953720549496199461
127.0.0.1  rack1       Up     Normal  96,9 KB         ?                   -6937145278184369493
```

8. Démarrage de cqlsh en utilisant la commande «cqlsh »:

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

Annexe 5 : Hadoop et ses composants

Le déploiement d'HBase requiert l'installation d'Hadoop et ses composants.

5.1 Création de groupe et utilisateurs Hadoop

– Création d'un groupe d'utilisateurs avec tous les droits, pour exécuter un nœud Hadoop :

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

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

L'utilisateur « hduser » est créé avec le mot de passe « hduser », et ajouté au groupe hadoop.

5.2 Configuration SSH

1. Installation de openssh-server pour avoir un accès SSH exigé par Hadoop pour gérer les différents nœuds :

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

2. Génération d'une clé SSH pour l'utilisateur hduser :

```
ubuntu@ubuntu-HP-630-Notebook-PC:~$ su hduser
Password:
hduser@ubuntu-HP-630-Notebook-PC:/home/ubuntu$ ssh-keygen -t rsa -P ""
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hduser/.ssh/id_rsa):
Created directory '/home/hduser/.ssh'.
Your identification has been saved in /home/hduser/.ssh/id_rsa.
Your public key has been saved in /home/hduser/.ssh/id_rsa.pub.
The key fingerprint is:
d2:12:ba:6a:76:7b:5e:5f:af:bd:4a:4b:2e:e2:9a:cb hduser@ubuntu-HP-630-Notebook-PC
The key's randomart image is:
+--[ RSA 2048]----+
|                 |
|                 |
|      .          |
|     . o         |
|    . o S        |
|     . o         |
|    .   .   +    |
|  o....o...= +   |
| o...+E+....=o+. |
+-----------------+
hduser@ubuntu-HP-630-Notebook-PC:/home/ubuntu$
```

Une clé RSA est créée avec un mot de passe vide.

3. Autorisation de la nouvelle clé nouvellement créée, pour accéder au SSH de la machine :

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

4. Test de la connexion SSH à partir de l'utilisateur hduser :

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

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

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

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

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

5.3 Installation

1. Téléchargement du paquet Hadoop depuis le site :

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

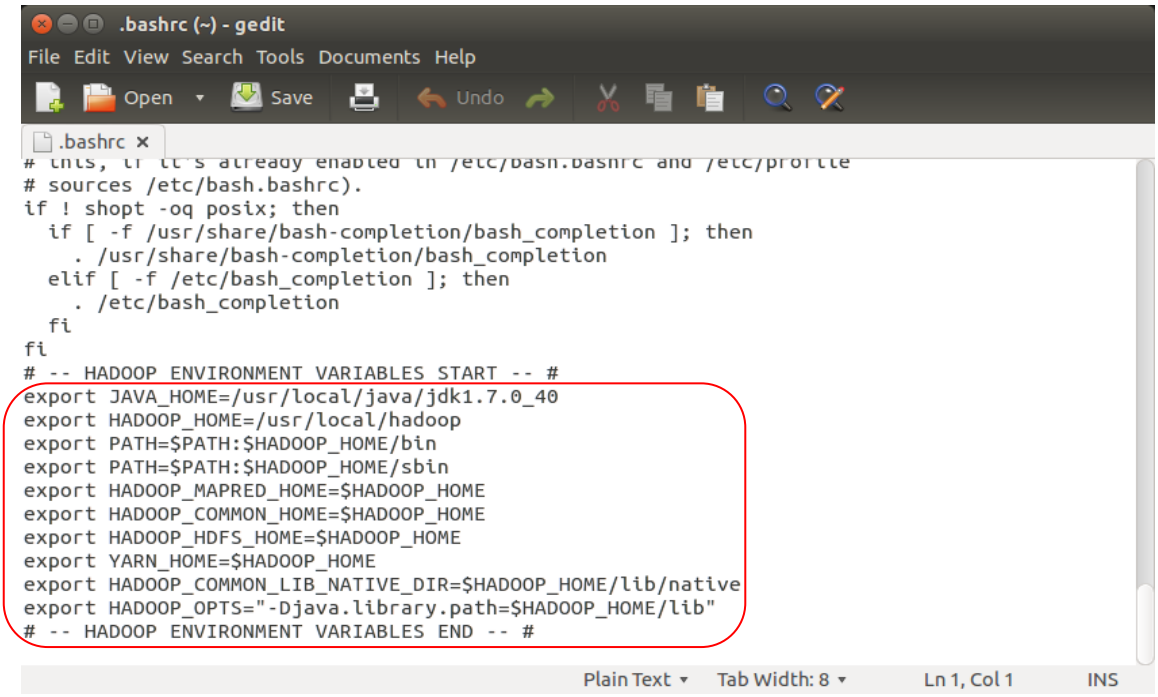
2. Modification du fichier « sysctl.conf » en ajoutant les lignes :

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

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

5.4 Mise à jour des fichiers de configuration Hadoop

5.4.1. bashrc



Exécution de la commande :

```
hduser@ubuntu-HP-630-Notebook-PC:~$ source ~/.bashrc
```

5.4.2 hadoop-env.sh

Définition de la variable JAVA_HOME dans le fichier, en ajoutant la ligne :

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

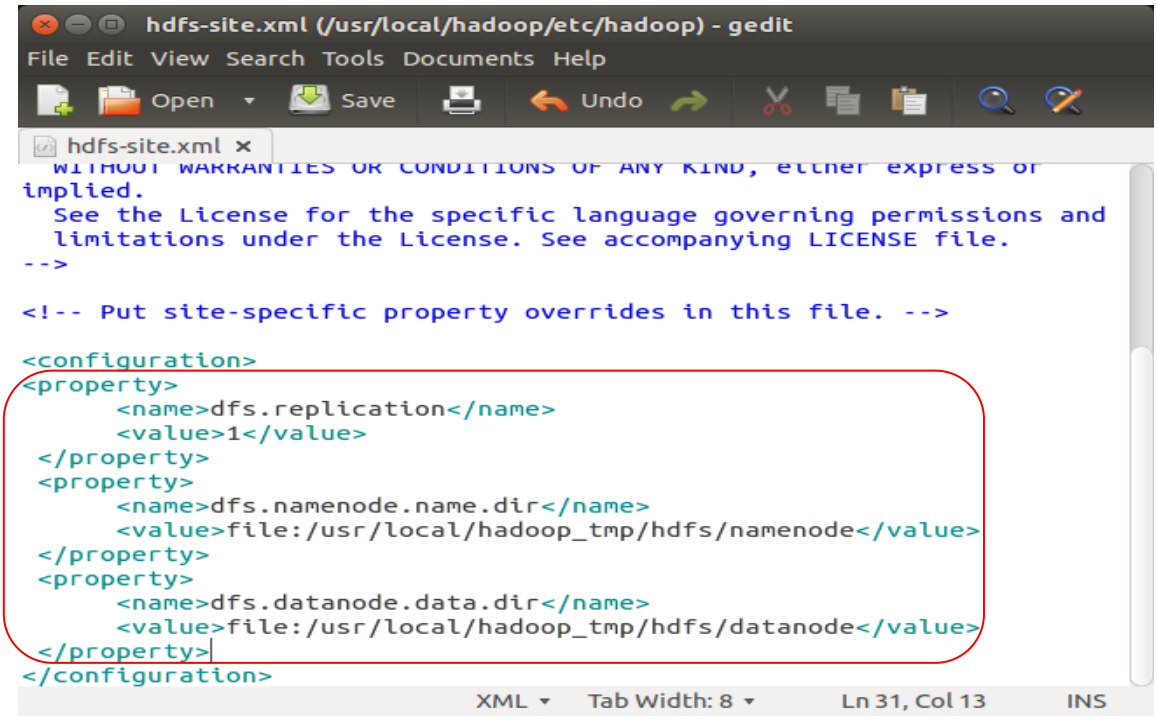
5.4.3 core-site.xml

Ajout des lignes suivantes dans la balise <configuration> :

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

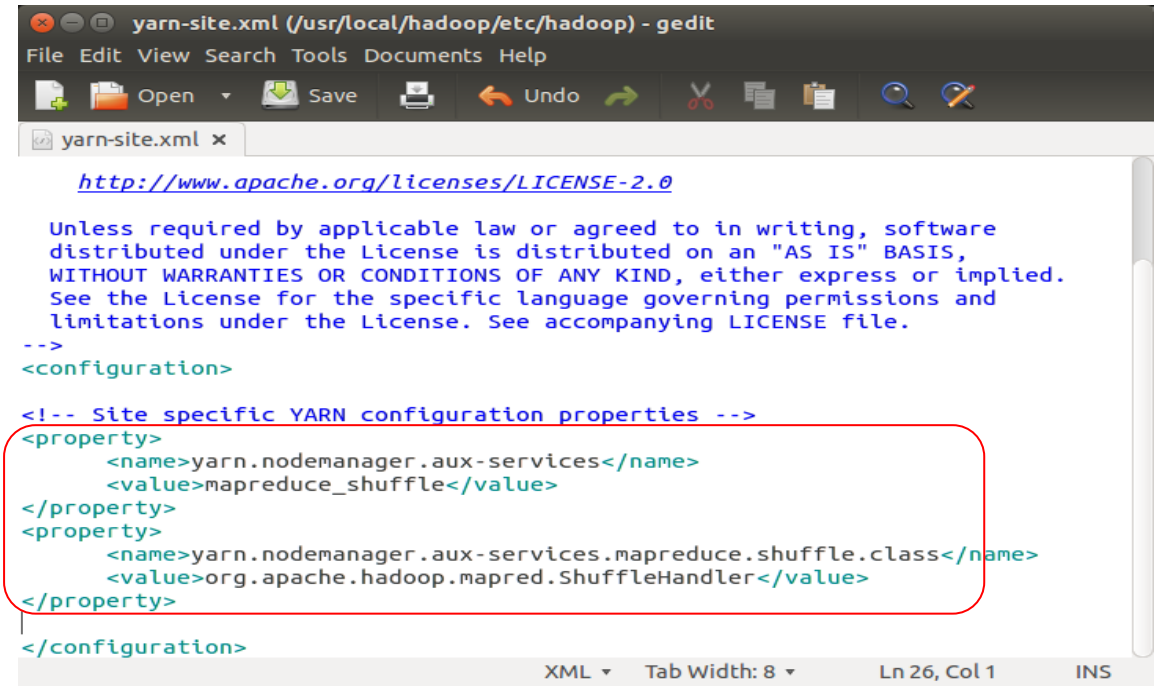
5.4.4 hdfs-site.xml

Ajout des lignes suivantes dans la balise <configuration> :



5.4.5 yarn-site.xml

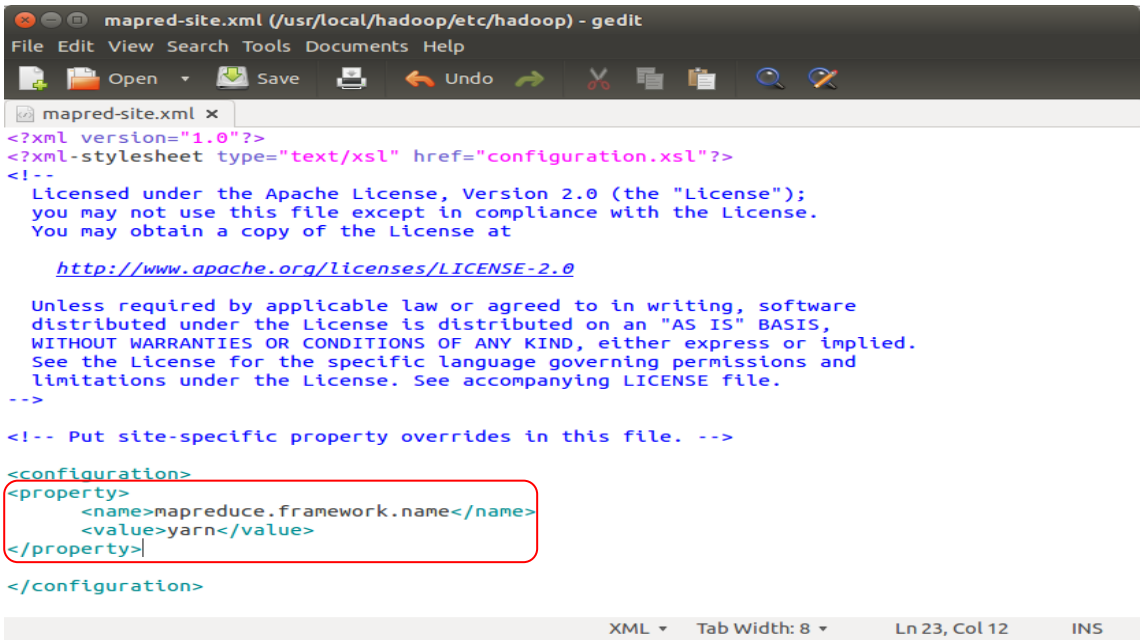
Ajout des lignes suivantes dans la balise <configuration> :



5.4.6 mapred-site.xml

1. Copiage du modèle du fichier mapred-site.xml.template :

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



5.5 Formatage HDFS et démarrage d'Hadoop

1. Formatage du nouveau système de gestion de fichier HDFS :

```
hduser@ubuntu-HP-630-Notebook-PC:~$ hdfs namenode -format
16/03/07 12:10:06 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = ubuntu-HP-630-Notebook-PC/127.0.1.1
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 2.6.0
STARTUP_MSG:   classpath = /usr/local/hadoop/etc/hadoop:/usr/local/hadoop/share/hadoop/common/lib/commons-configuration-1.6.jar:/usr/local/hadoo
p/share/hadoop/common/lib/zookeeper-3.4.6.jar:/usr/local/hadoop/share/hadoop/common/lib/curator-recipes-2.6.0.jar:/usr/local/hadoop/share/hadoop
/common/lib/commons-io-2.4.jar:/usr/local/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar:/usr/local/hadoop/share/hadoop/common/lib/java-
xmlbuilder-0.4.jar:/usr/local/hadoop/share/hadoop/common/lib/asm-3.2.jar:/usr/local/hadoop/share/hadoop/common/lib/servlet-api-2.5.jar:/usr/loca
l/hadoop/share/hadoop/common/lib/apacheds-kerberos-codec-2.0.0-M15.jar:/usr/local/hadoop/share/hadoop/common/lib/xz-1.0.jar:/usr/local/hadoop/sh
are/hadoop/common/lib/log4j-1.2.17.jar:/usr/local/hadoop/share/hadoop/common/lib/stax-api-1.0-2.jar:/usr/local/hadoop/share/hadoop/common/lib/je
ttison-1.1.jar:/usr/local/hadoop/share/hadoop/common/lib/xmlenc-0.52.jar:/usr/local/hadoop/share/hadoop/common/lib/jetty-util-6.1.26.jar:/usr/lo
cal/hadoop/share/hadoop/common/lib/hadoop-auth-2.6.0.jar:/usr/local/hadoop/share/hadoop/common/lib/junit-4.11.jar:/usr/local/hadoop/share/hadoop
/common/lib/jackson-xc-1.9.13.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-beanutils-core-1.8.0.jar:/usr/local/hadoop/share/hadoop/comm
on/lib/jackson-mapper-asl-1.9.13.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-codec-1.4.jar:/usr/local/hadoop/share/hadoop/common/lib/c
urator-framework-2.6.0.jar:/usr/local/hadoop/share/hadoop/common/lib/jets3t-0.9.0.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-beanutil
s-1.7.0.jar:/usr/local/hadoop/share/hadoop/common/lib/activation-1.1.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-httpclient-3.1.jar:/u
sr/local/hadoop/share/hadoop/common/lib/jasper-runtime-5.5.23.jar:/usr/local/hadoop/share/hadoop/common/lib/guava-11.0.2.jar:/usr/local/hadoop/s
hare/hadoop/common/lib/paranamer-2.3.jar:/usr/local/hadoop/share/hadoop/common/lib/jersey-core-1.9.jar:/usr/local/hadoop/share/hadoop/common/lib
/hadoop-annotations-2.6.0.jar:/usr/local/hadoop/share/hadoop/common/lib/jasper-compiler-5.5.23.jar:/usr/local/hadoop/share/hadoop/common/lib/com
mons-cli-1.2.jar:/usr/local/hadoop/share/hadoop/common/lib/api-util-1.0.0-M20.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-logging-1.1.
3.jar:/usr/local/hadoop/share/hadoop/common/lib/curator-client-2.6.0.jar:/usr/local/hadoop/share/hadoop/common/lib/slf4j-api-1.7.5.jar:/usr/loca
l/hadoop/share/hadoop/common/lib/api-asn1-api-1.0.0-M20.jar:/usr/local/hadoop/share/hadoop/common/lib/netty-3.6.2.Final.jar:/usr/local/hadoop/sh
are/hadoop/common/lib/jaxb-api-2.2.2.jar:/usr/local/hadoop/share/hadoop/common/lib/hamcrest-core-1.3.jar:/usr/local/hadoop/share/hadoop/common/l
ib/jetty-6.1.26.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-lang-2.6.jar:/usr/local/hadoop/share/hadoop/common/lib/jackson-jaxrs-1.9.1
3.jar:/usr/local/hadoop/share/hadoop/common/lib/mockito-all-1.8.5.jar:/usr/local/hadoop/share/hadoop/common/lib/httpcore-4.2.5.jar:/usr/local/ha
doop/share/hadoop/common/lib/jsch-0.1.42.jar:/usr/local/hadoop/share/hadoop/common/lib/htrace-core-3.0.4.jar:/usr/local/hadoop/share/hadoop/comm
```

2. Démarrage d'Hadoop :

```
start-dfs.sh
start-yarn.sh
```

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

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

5.6 HBase

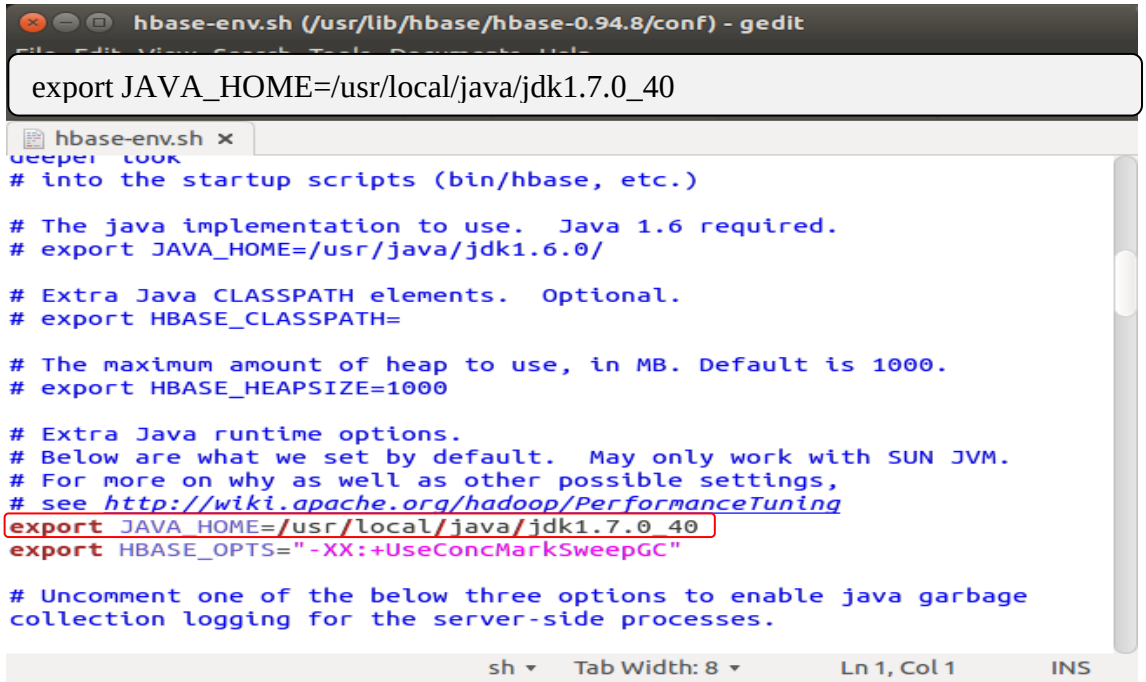
Si le processus de déploiement d'Hadoop aboutit, l'installation d'HBase sera plus aisée. Pour le faire, il faut commencer par choisir une version stable conforme à la configuration disponible. Voici les étapes à suivre pour effectuer l'installation :

1. Téléchargement d'HBase du site : http://www.apache.org/dyn/closer.cgi/hbase/
2. Connexion en tant que « hduser » et décompression du fichier téléchargé :

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

3. Configuration d'HBase avec java :

Ouverture de fichier « hbase-env.sh » et définitiondu chemin de java installé :



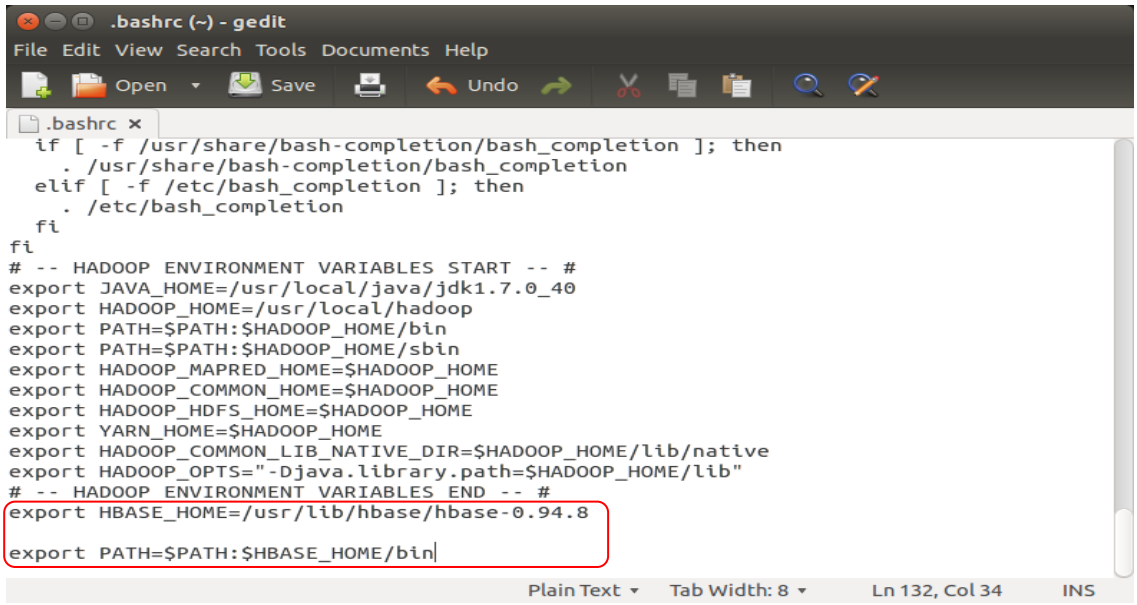
4. Définition du chemin de HBASE_HOME dans le fichier « .bashrc »:

gedit ~/.bashrc

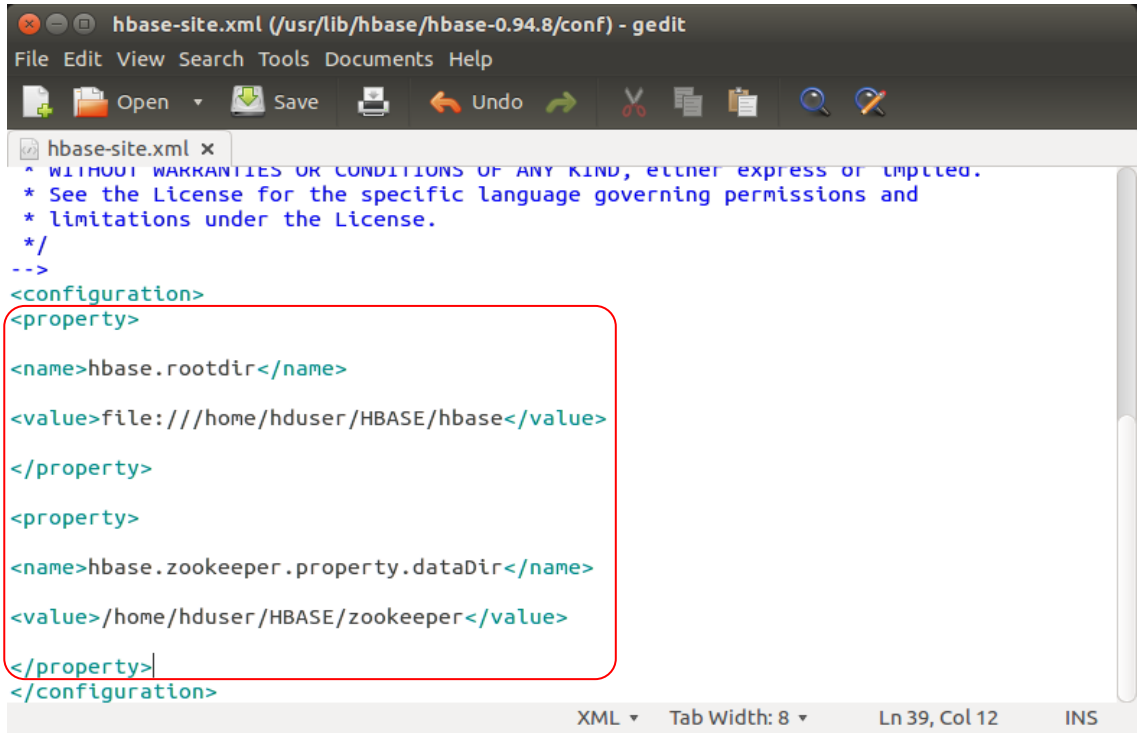
Ajouter :

export HBASE_HOME=/usr/lib/hbase/hbase-0.94.8

export PATH=$PATH:$HBASE_HOME/bin



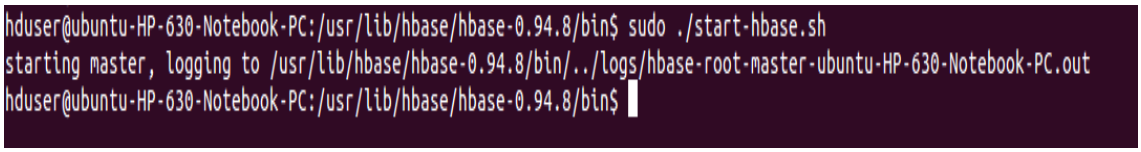
5. Modification du fichier « HBase-site.xml » pour compléter l'installation d'HBase :



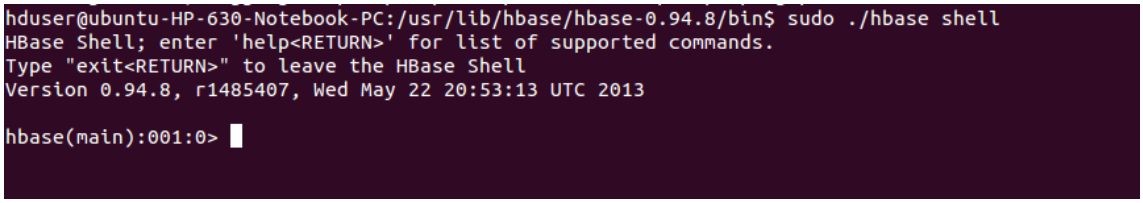
6. Changement de l'entrée 127.0.1.1, dans le répertoire « /etc/hosts », par 127.0.0.1

cd /etc
gedit hosts

7. Démarrage d'HBase :



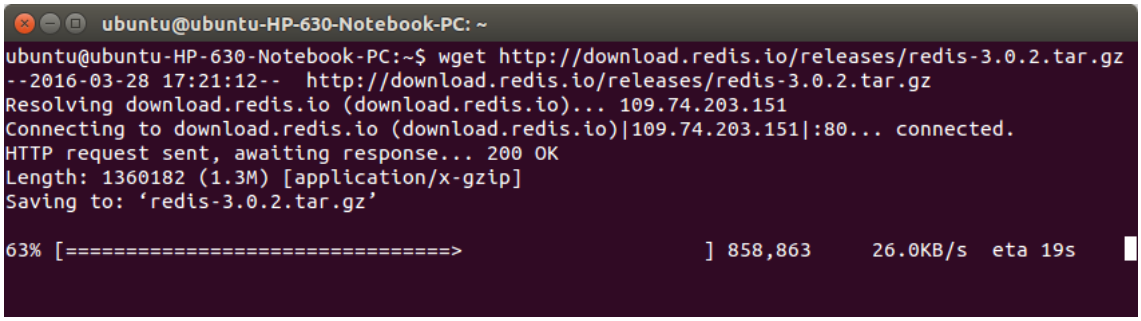
8. Lancement du shell par la commande « sudo./hbase shell» :



Annexe 6 : Redis

1. Téléchargement de Redis à partir du terminal :

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

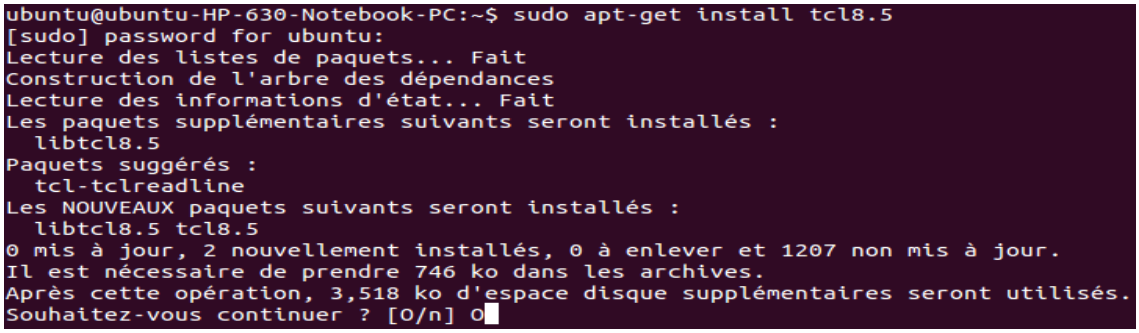


2. Extraction et compilation de Redis :

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

3. Installation de « tcl8.5 » :

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



4. Exécution et installation du test recommandé :

```
make test
sudo make install
```

5. Installation du script :

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

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

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

6. Démarrage et arrêt de Redis :

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

7. Vérification de Redis :

```
src/redis-cli
```

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

Annexe 7 : OrientDB

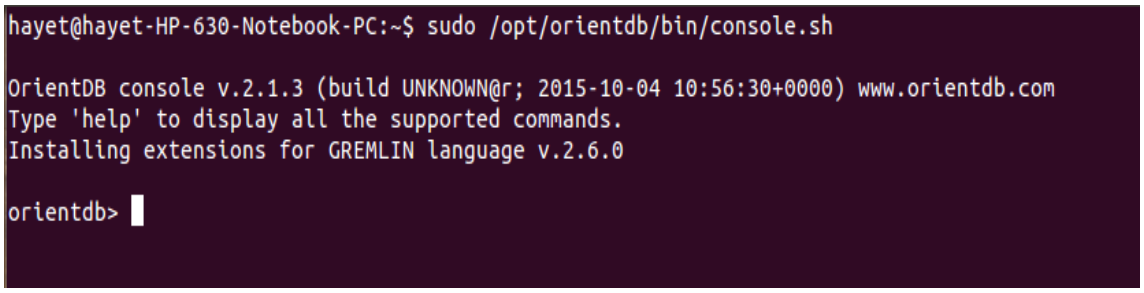
1. Téléchargement et installation de la version 2.1.3 de la communauté OrientDB :

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

2. Décompression et renommagedu fichier téléchargé pour le rendre plus facile àmanipuler :

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

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



4. Vérifier que le serveur est à l'écoute sur les ports 2424 (pour les connexions binaires) et 2480 (pour les connexions HTTP), en exécutant dans un autre terminal les commandes :

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

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

5. Connexion à l'instance du serveur :

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

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

orientdb> connect remote:127.0.0.1 root
Enter password:

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

Annexe 8 : Workload A


```
# Yahoo! Cloud System Benchmark
# Workload A: Update heavy workload
#   Application example: Session store recording recent actions
#
#   Read/update ratio: 50/50
#   Default data size: 1 KB records (10 fields, 100 bytes each, plus key)
#   Request distribution: zipfian

recordcount=1000
operationcount=1000
workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=true

readproportion=0.5
updateproportion=0.5
scanproportion=0
insertproportion=0

requestdistribution=zipfian
```

Annexe 9 : Workload B

```
# Copyright (c) 2010 Yahoo! Inc. All rights reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"); you
# may not use this file except in compliance with the License. You
# may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied. See the License for the specific language governing
# permissions and limitations under the License. See accompanying
# LICENSE file.

# Yahoo! Cloud System Benchmark
# Workload B: Read mostly workload
#   Application example: photo tagging; add a tag is an update, but most operations are to read
tags
#
#   Read/update ratio: 95/5
#   Default data size: 1 KB records (10 fields, 100 bytes each, plus key)
#   Request distribution: zipfian

recordcount=1000
operationcount=1000
workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=true

readproportion=0.95
updateproportion=0.05
scanproportion=0
insertproportion=0

requestdistribution=zipfian
```

Annexe 10 : Workload C

```
# Copyright (c) 2010 Yahoo! Inc. All rights reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"); you
# may not use this file except in compliance with the License. You
# may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied. See the License for the specific language governing
# permissions and limitations under the License. See accompanying
# LICENSE file.

# Yahoo! Cloud System Benchmark
# Workload C: Read only
#   Application example: user profile cache, where profiles are constructed elsewhere (e.g.,
Hadoop)
#
#   Read/update ratio: 100/0
#   Default data size: 1 KB records (10 fields, 100 bytes each, plus key)
#   Request distribution: zipfian

recordcount=1000
operationcount=1000
workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=true

readproportion=1
updateproportion=0
scanproportion=0
insertproportion=0

requestdistribution=zipfian
```

Annexe 11 : Workload D

```
# Copyright (c) 2010 Yahoo! Inc. All rights reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"); you
# may not use this file except in compliance with the License. You
# may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied. See the License for the specific language governing
# permissions and limitations under the License. See accompanying
# LICENSE file.

# Yahoo! Cloud System Benchmark
# Workload D: Read latest workload
#   Application example: user status updates; people want to read the latest
#
#   Read/update/insert ratio: 95/0/5
#   Default data size: 1 KB records (10 fields, 100 bytes each, plus key)
#   Request distribution: latest

# The insert order for this is hashed, not ordered. The "latest" items may be
# scattered around the keyspace if they are keyed by userid.timestamp. A workload
# which orders items purely by time, and demands the latest, is very different than
# workload here (which we believe is more typical of how people build systems.)

recordcount=1000
operationcount=1000
workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=true

readproportion=0.95
updateproportion=0
scanproportion=0
insertproportion=0.05

requestdistribution=latest
```

Annexe 12 : Workload E

```
# Copyright (c) 2010 Yahoo! Inc. All rights reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"); you
# may not use this file except in compliance with the License. You
# may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied. See the License for the specific language governing
# permissions and limitations under the License. See accompanying
# LICENSE file.

# Yahoo! Cloud System Benchmark
# Workload E: Short ranges
#   Application example: threaded conversations, where each scan is for the posts in a given thread (assumed to be clustered by thread id)
#
#   Scan/insert ratio: 95/5
#   Default data size: 1 KB records (10 fields, 100 bytes each, plus key)
#   Request distribution: zipfian

# The insert order is hashed, not ordered. Although the scans are ordered, it does not necessarily
# follow that the data is inserted in order. For example, posts for thread 342 may not be inserted contiguously, but
# instead interspersed with posts from lots of other threads. The way the YCSB client works is that it will pick a start
# key, and then request a number of records; this works fine even for hashed insertion.

recordcount=1000
operationcount=1000
workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=true

readproportion=0
updateproportion=0
scanproportion=0.95
insertproportion=0.05

requestdistribution=zipfian

maxscanlength=100

scanlengthdistribution=uniform
```

Annexe 13 : Workload F

```
# Copyright (c) 2010 Yahoo! Inc. All rights reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"); you
# may not use this file except in compliance with the License. You
# may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied. See the License for the specific language governing
# permissions and limitations under the License. See accompanying
# LICENSE file.

# Yahoo! Cloud System Benchmark
# Workload F: Read-modify-write workload
#   Application example: user database, where user records are read and modified by the user
or to record user activity.
#
#   Read/read-modify-write ratio: 50/50
#   Default data size: 1 KB records (10 fields, 100 bytes each, plus key)
#   Request distribution: zipfian

recordcount=1000
operationcount=1000
workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=true

readproportion=0.5
updateproportion=0
scanproportion=0
insertproportion=0
readmodifywriteproportion=0.5

requestdistribution=zipfian
```

Annexe 14 : Exemple d'un programme MapReduce

L'exemple classique du framework MapReduce est le compteur de mots : **WordCount**. WordCount permet de compter le nombre d'occurrences d'un mot dans un fichier. En entrée l'algorithme reçoit un fichier texte qui contient les mots suivants : **voiture la le elle de elle la se la maison voiture**……

Dans notre exemple, la clé d'entrée correspond au numéro de ligne dans le fichier et tous les mots sont comptabilisés à l'exception du mot « se ».

Le résultat de la fonction Map est donné ci-dessous :

(voiture, 1) /(la,1) /(le,1) / (elle,1) / (de,1) / (elle,1) / (la,1) / (la,1) / (maison,1) / (voiture,1)

Avant d'éxécuter la fonction Reduce, deux opérations intermédiaires doivent être exécutées pour préparer la valeur de son paramètre d'entrée. La première opération appelée *shuffle* permet de grouper les valeurs dont la clé est commune. La seconde opération appelée sort permet de trier par clé. A la différence des fonctions Map et Reduce, *shuffle* et *sort* sont des fonctions fournies par le framework Hadoop, donc, il n'a pas à les implémenter.

Ainsi, après l'exécution des fonctions *shuffle* et *sort* le résultat de l'exemple est le suivant :

(de, [1]) / (elle, [1,1]) / (la, [1, 1,1]) / (le, [1]) / (maison, [1]) / (voiture, [1,1]

Suite à l'appel de la fonction Reduce, le résultat de l'exemple est le suivant :

(de, 1) / (elle, 2) / (la, 3) / (le, 1) / (maison, 1) / (voiture, 2)

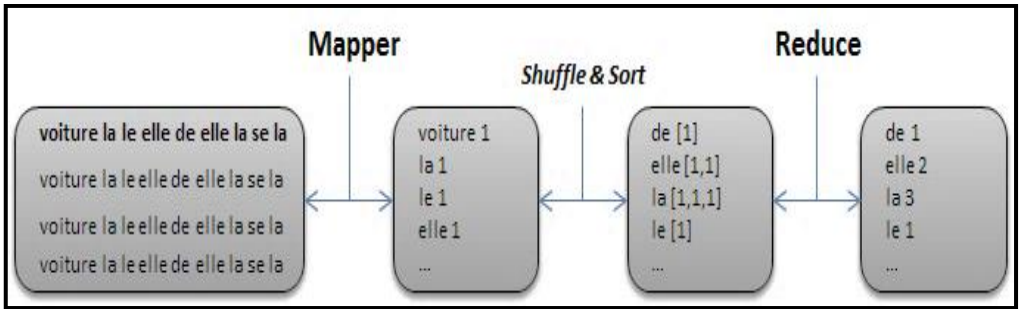


Figure: Exemple d'un programme MapReduce (WordCount) [100]

Le code JAVA de la fonction Map () et Reduce () de cet exemple est le suivant :

- Implémentation de la fonction Map

```
public class WordCountMapper extends Mapper<LongWritable, Text,
Text, IntWritable> {
  public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
StringTokenizeritr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
         String string = itr.nextToken();
         if (!string.equals("se")) {
            Text word = new Text();
word.set(string);
context.write(word, new IntWritable(1));} } } }
```

- Implémentation de la fonction Reduce

```
Public class WordCountReducer extends Reducer<Text, IntWritable,
Text, IntWritable>{
Public void reduce(TextKey, Iterable<IntWritable>values, Context
context)
Int sum=0;
For(IntWritablecurrent:values){
Sum+=current.get();
}
Context.write(Key, new IntWritable(sum));}}
```

Table des figures

Liste des tableaux

Références bibliographiques

1. Abramova, V, Bernardino, J and Furtado, P (2014). Experimental evaluation of NoSQL databases. International Journal of Database Management Systems (IJDMS), Volume6, No.3, pp. 1-16.

2. Lyman, P, Varian, HR, Swearingen, K, Charles, P, et al. (2003). How much information? UC Berkeley. http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/print.htm.

3. Bohn, RE and Short, J (2009). How Much Information?Report on American Consumers. UC San Diego. http://group47.com/HMI_2009_ConsumerReport_Dec9_2009.pdf.

4. Gantz, J and Reinsel, D (2012). Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East. https://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf.

5. Zhang, Q, Cheng L, and Boutaba, R (2010). Cloud computing state-of-the-art and research challenges. Journal of Internet Services and Applications. Volume 1, Issue 1, pp. 7-18.

6. Agrawal, D, Das, S and El Abbadi, A (2010). Big Data and Cloud Computing: New Wine or just New Bottles? In Proceedings of the VLDB Endowment.Volume 3, Issue 1-2, pp. 1647-1648.

7. Agrawal, D, Das, S and El Abbadi, A (2011). Big Data and Cloud Computing: Current State and Future Opportunities. In Proceedings of the 14th International Conference on Extending Database Technology (EDBT/ICDT'11), Uppsala, Sweden - March 21-24, 2011, pp. 530-533.

8. Hurwitz, J, Nugent, A, Halper, F, and Kaufman, M (2013). Big data for dummies? Ebook. https://eecs.wsu.edu/~yinghui/mat/courses/fall%202015/resources/Big%20data%20for%20dummies.pdf.

9. Popescu, A, and Bacalu, AM (2012). Big Data Causes Concern and Big Confusion. A Big Data definition to Help Clarify the Confusion. Rapport de Thor Olavsrud (CIO) pour ITWorld résultats d'une enquête sur BigData.

10. Fermigier, S (2012). Big Data and Open Source: Une Convergence Inévitable. Livre Blanc (http://fermigier.com/blog/2012/03/new-whitepaper-big-data-open-source/).

11. Sawant, N and Shah, H (2013). Big Data Application Architecture QandA : A Problem - Solution Approach. Distributed to the book trade Worldwide Springer Science+Business Media. Isbn-13 : 978-1-4302-6292-3.

12. Vaquero, L, Rodero-Merino, L, Caceres, J and Lindner M (2009). A break in the clouds : towards a cloud definition. ACM SIGCOMM computer communications review. Volume 39, Issue 1, January 2009, pp 50-55.

13. Armbrust, M, and al (2009). Above the clouds : A Berkeley view of cloud computing. UC Berkeley Technical Report.

14. Yadav, S (2013). Comparative Study on Open Source Software for Cloud Computing Platform : Eucalyptus, OpenStack and OpenNebula. International Journal Of Engineering And Science Volume 3, Issue 10, pp 51-54.

15. Meghnine, Y (2014). Les bases de données dans les nuages. Exposé dans le cadre de la formation PGS Management des Systèmes d'Information (Université d'Oran1).

16. Léonard, M (2014). L'avenir du NoSQL. http://www.leonardmeyer.com/wp-content/uploads/ 2014/06/avenirDuNoSQL.pdf.

17. Rakesh, K, Parashar, BB, Gupta, S, Sharma, Y and Gupta, N (2014). Apache Hadoop, NoSQL and NewSQL Solutions of Big Data. International Journal of Advance Foundation and Research in Science and Engineering (IJAFRSE) Volume 1, Issue 6.

18. Steve, R (2018). Le guide complet du cloud computing - Partie 1. https://www.zdnet.fr/actualites/le-guide-complet-du-cloud-computing-partie-1-39865000.htm, (Consulté en Mai 2018).

19. Venet , R (2011). Cloud Computing : avantages et inconvénients. http://www.renaudvenet.com/cloud-computing-avantages-et-inconvenients-2011-01-26.html, (Consulté en Septembre 2014).

20. Codd, EF (1970). A relational model of data for large shared data banks, Communications of the ACM, Volume 13, No. 6, pp. 377-387.

21. Jatana, N, Puri, S, Ahuja, M, Kathuria, I and Gosain, D (2012), A survey and comparison of relational and non-Relational Databases. International Journal of Engineering Research and Technology (IJERT). ISSN: 2278-0181, Vol 1, Issue 6.

22. Abadi, DJ (2009). Data Management in the Cloud: Limitations and Opportunities. IEEE Data Engineering Bulletin, Volume 32, No. 1, pp. 3-12.

23. Rahman, H, Mehedi, A, and Akhter, S (2013). Simulation and Modeling Techniques: A GreenCloud Tutorial. https://fr.slideshare.net/habibur01/a-tutorial-on-greencloud. (Consulté en Janvier 2016).

24. Oracle Real Application Clusters (RAC), (2013). An Oracle White Paper.

25. MySQL Cluster CGE (2013). https://www.mysql.fr/ products/ cluster. (Consulté en Novembre 2016).

26. Kouedi, E (2012). Approche de migration d'une base de données relationnelle vers une base de données NoSQL orientée colonne, Mémoire de Master (Université de YaoundeI).

27. Lasalle, MF (2006). Cours du Relationnel à l'objet : Limites du Relationnel, http://circe.univ-fcomte.fr/Marie-France-Lasalle/bda/COURSHTM/cours/chap01/lec02/page01.htm.

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

29. Whitehorn, M (2015). Quand faut-il envisager d'utiliser une base de données NoSQL (plutôt qu'une base relationnelle) ? Le Grand Guide des Bases de Données (part. 2) : au-delà du relationnel (University de Dundee).

30. Rith, J, Lehmayr, PS and Meyer-Wegener, K (2014). Speaking in Tongues: SQL Access to NoSQL Systems. In Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC '14). Gyeongju, Korea, March 24-28 2014, pp. 855-857.

31. Mc Creary, D and Kelly, A (2014). Making Sense of NoSQL: A guide for managers and the rest of us. Edition : Manning Publications.

32. Di Maglie, M (2012). Adoption d'une solution NoSQL dans l'entreprise, Bachelor HES (Haute École de Gestion de Genève).

33. Han, J, Haihong, E, Le, G and Dual, J (2011). Survey on NoSQL database. In 6th IEEE International Conference on Pervasive Computing and Applications (ICPCA'2011), Port Elizabeth, South Africa, pp. 363-366.

34. Cattell, R (2010). Scalable SQL and NoSQL Data Stores. SIGMOD Record, Volume 39, Issue 4, pp. 12-27, Indiana, USA.

35. Nance, C, Losser, T, Iype, R and Harmon, G (2013). NoSQL vs RDBMS - Why There is Room for Both. In Proceedings of the Southern Association for Information Systems Conference, Savannah, GA, USA, March 8–9, 2013, pp. 111-116.

36. Chen, Y, Qin, X, Bian, H, Chen, J et al. (2014). A Study of SQL-on-Hadoop Systems. Big Data Benchmarks, Performance Optimization, and Emerging Hardware - 4th and 5th Workshops, BPOE 2014, Salt Lake City, USA, LNCS, Volume 8807 (Springer), pp. 154-166.

37. Abadi, D, Babu, S, Ozcan, F and Pandis, I (2015). Tutorial: SQL-on-Hadoop Systems. In Proceedings of the VLDB Endowment, Indiana,USA, Volume 8, Issue 12, pp. 2050-2051.

38. Aslett, M (2011). How will the database incumbents respond to NoSQL and NewSQL? https://451research.com/report-short?entityId=66963. (Consulté en Novembre 2016).

39. Grolinger, K, Higashino, WA, Tiwari, A and Capretz, MAM (2013). Data management in cloud environments: NoSQL and NewSQL data stores. Journal of Cloud Computing: Advances, Systems and Applications, Volume 2, Issue 1, pp. 1-24.

40. Kumar, R, Gupta, N, Maharwal, H, Charu, S et al. (2014). Critical Analysis of Database Management Using NewSQL. International Journal of Computer Science and Mobile Computing (IJCSMC), Volume 3, Issue. 5, pp.434-438.

41. Gilbert, S and Lynch, N (2002). Brewer's Conjecture and the Feasibility of Consistent Available Partition Tolerant Web Services, ACM SIGACT, Volume 33, Issue2, June 2002, pp. 51-59.

42. Abadi, D (2012). Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story. Journal Computer. Volume 45, Issue 2, pp 37-42. DOI: 10.1109/MC.2012.33.

43. Degroodt, N (2011). L'élasticité des bases de données sur le cloud computing. Mémoire de Master (Université Libre de Bruxelles).

44. Indrawan-Santiago, M (2012). Database Research: Are We at a Crossroad? Reflection on NoSQL. In Proceedings of the 15th International Conference on Network-Based Information Systems (NBiS'2012), September 26 - 28, 2012, Melbourne, pp. 45-51.

45. Hecht, R, Jablonski, S (2011). NoSQL evaluation: A use case-oriented survey. In Proceedings of the International Conference on Cloud and Service Computing, IEEE. November 22-24 2012, Huashan Road, Shanghai, China pp. 336-341.

46. Piazza, AG (2013). NoSQL Etat de l'art et benchmark. Bachelor HES (Haute École de Gestion de Genève).

47. Ma, Z and Yan, L (2016). A Review of RDF Storage in NoSQL Databases. In book: Managing Big Data in Cloud Computing Environments, IGI Global, pp. 210-229. Doi: 10.4018/978-1-4666-9834-5.ch009.

48. Mesnier, V (2015). NewSQL. http://air.imag.fr/index.php/NewSQL#Enseignants. (Consulté en Mars 2017).

49. Hashem, H (2016). Modélisation intégratrice du traitement BigData. Thèse de Doctorat (Université Paris Saclay).

50. Hadrien, F (2015). SQL, NoSQL, NewSQL stratégie de choix, Bachelor HES (Haute École de Gestion de Genève).

51. Piekos, J (2015). SQL vs NoSQL vs NewSQL : Finding the right solution. http://dataconomy.com/2015/08/sql-vs-nosql-vs-newsql-finding-the-right-solution/ (Consulté en Janvier 2018).

52. Solid IT, (2018). DB-Engines Ranking http://db-engines.com/en/ranking. (Consulté en Juillet 2018).

53. MongoDB (2016). Release Notes. http://docs.mongodb.org/manual/release-notes. (Consulté en Juin 2017).

54. Abramova, V and Bernardino, J (2013). NoSQL databases: MongoDB vs Cassandra. In Proceedings of the International C* Conference on Computer Science and Software Engineering (C3S2E '13). July 10 - 12, 2013, Porto, Portugal, Pages 14-22.

55. Heinrich, L (2012). Architecture NoSQL et réponse au Théorème CAP. Bachelor HES (Haute École de Gestion de Genève).

56. Datastax Academy (2017). Planet Cassandra, A Datastax Community Service https://academy.datastax.com/planet-cassandra/tour-of-configuration-files. (Consulté en en Décembre 2017).

57. Goebel, N (2013). A Quick Introduction to Apache Cassandra. https://www.sitepoint.com/a-quick-introduction-to-apache-cassandra/. (Consulté en Avril 2015).

58. Doan, DH (2014). Modèle de stockage physique dans Cassandra. http://www.infoq.com/fr/articles/modele-stockage-physique-cassandra. (Consulté en Avril 2015).

59. Travers, N, Fournier, R, S'niehotta, RF and Rigaux, P (2014). Systèmes NoSQL : la réplication. http://b3d.bdpedia.fr/replication.html. (Consulté en Juin 2016).

60. Devoxx (2012). Introduction à la base de données NoSQL Cassandra. http://soat.developpez.com/articles/cassandra/. (Consulté en Novembre 2016).

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

62. Jiang, Y (2012). HBase Administration Cookbook. Packt Publishing Open source community experience distilled. Isbn : 1849517142 9781849517140.

63. Apache HBase (2016). Apache HBase Reference Guide. http://hbase.apache.org/book.html#arch.overview (Consulté en Août 2016).

64. Journal du Net (2013). Comparatif des bases de données NoSQL. https://www.journaldunet.com/developpeur/outils/comparatif-des-bases-nosql/. (Consulté en Février 2015).

65. Objelean, A (2015). Introduction to Couchbase-NoSQL Document Database. http://alexo.github.io/articles/Introduction_to_couchbase_no_sql_document_database/. (Consulté en Mai 2016).

66. Couchbase Server (2017). Why Couchbase ? http://developer.couchbase.com/documentation/server/4.1/architecture/architectureintro.html. (Consulté en Mai 2016).

67. Potsangbam, H (2015). Learning CouchBase: Design documents and implement real world e-commerce applications with Couchbase. Packt Publishing Open source community experience distilled. ISBN 139781785288593.

68. OrientDB Manual (2016). Data Modeling.Multi-Model https://www.orientdb.com/docs/last/datamodeling/Tutorial-Document-and-graph-model.html. (Consulté en Juin 2016).

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

70. Datastax (2015). Benchmarking Top NoSQL Databases : Apache Cassandra, Couchbase, HBase, and MongoDB. https://www.datastax.com/wp-content/themes/datastax-2014-08/files/NoSQL_Benchmarks_EndPoint.pdf. (Consulté en en Décembre 2016).

71. Park, HJ (2016). A Study about Performance Evaluation of Various NoSQL Databases. The Journal of Korea Institute of Information, Electronics, and Communication Technology, Volume 9, Issue 3, pp. 298-305.

72. Martins, G, Bezerra, P, Gomes R, Albuquerque F, Costa A (2015). Evaluating performance degradation in NoSQL databases generated by virtualization. In Proceedings IEEE of 8th Latin American Network Operations and Management Symposium (LANOMS), 1-3 October 2015, João Pessoa, Brazil, pp. 84-91.

73. Abramova, V, Bernardino, J and Furtado, P (2015). SQL or NoSQL? Performance and scalability evaluation. Int. J. Business Process Integration and Management (IJBPIM), Volume 7, Issue 4, pp. 314-321.

74. Lungu, I and Tudorica, BG (2013). The development of a benchmark tool for NoSQL databases. Database Systems Journal, Volume 4, Issue 2, p 13-20.

75. Carlin, S., and Curran, K. (2011), Cloud Computing Security, International Journal of Ambient Computing and Intelligence (IJACI), Volume 3, Issue 1, pp.14-19.

76. Chesnot, G. (2012). Cloud Computing, Big Data, Parallélisme, Hadoop – Stockage de données du Futur. Paris: Vuibert edition. Isbn: 978-2-311-01195-1.

77. Apache HBase (2016). Apache HBase Reference Guide. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. (Consulté en Septembre 2016).

78. Apache HBase (2016). MapReduce Tutorial. https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html. (Consulté en Septembre 2016).

79. Dean, J and Ghemawat, S (2008). MapReduce: simplified data processing on large clusters. ACM - 50th anniversary 1958 – 2008, New York, NY, USA. Volume 51, Issue 1, January 2008, pp 107-113. Doi: 10.1145/1327452.1327492.

80. Jain, A., and Bhatnagar, V. (2016). Movie Analytics for Effective Recommendation System using Pig with Hadoop, International Journal of Rough Sets and Data Analysis (IJRSDA), Volume 3, Issue 2, pp. 82-100.

81. Ahuja, SP, Sanjay, P and Moore, B (2013). State of Big Data Analysis in the Cloud. Journal of Network and Communication Technologies (NCT), Vol. 2, Issue 1, pp. 62-68.

82. TechTarget (2014). Tout savoir sur Hadoop : Vulgarisation de la technologie et les stratégies de certains acteurs. http://docs.media.bitpipe.com/io_10x/io_108885/item_951685/ Handbook_Tout%20savoir%20sur%20Hadoop%201ere%20partie.pdf.

83. Gerlier, J and Chen, S (2011). Prototypage et évaluation de performances d'un service de traçabilité avec une architecture distribuée basée sur Hadoop. http://docplayer.fr/storage/18/673047/673047.pdf.

84. Benjamin, R (2014). Hadoop/Big Data, Université de Nice Sophia-Antipolis, p 114. http://myuuu.fr/cours/MBDS/BigData/mbds_big_data_hadoop_2013_2014_cours_1.pdf

85. White, T (2015). Hadoop: The definitive guide 4th edition. Publisher : O'Reilly Media. Isbn : 978-1-491-90163-2.

86. White, T (2012). Hadoop: The definitive guide 3rd edition. Publisher : O'Reilly Media. ISBN : 13-978-1449311520.

87. Baron, M (2014). Généralités sur HDFS et MapReduce. https://mbaron.developpez.com/tutoriels/bigdata/hadoop/introduction-hdfs-map-reduce/. (Consulté en Octobre 2015).

88. Shafer, J., Rixner, S., and Cox, A.L. (2010). The Hadoop Distributed File System: Balancing Portability and Performance. In IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). IEEE: Rice Univ., Houston, TX, USA; 122-133.

89. Parageaud, C (2013). Big Data : La jungle des différentes distributions open source Hadoop. http://blog.ippon.fr/2013/05/14/big-data-la-jungle-des-differentes-distributions-open-source-hadoop/. (Consulté en Avril 2015).

90. Agrawal, D., El Abbadi, A., Antony, S., and Das, S. (2010). Data Management Challenges in Cloud Computing Infrastructures. In Proceedings of the 6th international conference on Databases in Networked Information Systems Pages 1-10 Springer-Verlag (DNIS'10), Aizu-Wakamatsu, Japan.

91. Marz, N., and Warren, J. (2015). Big Data: Principles and best practices of scalable realtime data systems. New Jersey: Manning Publications. Isbn: 9781617290343.

92. Patra, P.K, Singh, H., Singh, R., Das, S., Dey, N., and Drugarin, C.V.A. (2016). Replication and Resubmission Based Adaptive Decision for Fault Tolerance in Real Time Cloud Computing: A New Approach, International Journal of Service Science, Management, Engineering, and Technology (IJSSMET), Volume 7, Issue 2, pp. 46-60.

93. Matallah, H., Belalem, G., and Bouamrane, K. (2017). Experimental comparative study of NoSQL databases: HBASE versus MongoDB by YCSB. International Journal of Computer Systems Science and Engineering (IJCSSE), Volume 32, Issue 4, pp 307-317.

94. Kishor, D.R., and Venkateswarlu, N.B. (2016). A Novel Hybridization of Expectation-Maximization and K-Means Algorithms for Better Clustering Performance, International Journal of Ambient Computing and Intelligence (IJACI), Volume 7, Issue 2, pp. 47-74.

95. Matallah, H and Belalem, G (2014). New architecture of storage and access in Big Data and Cloud Computing. In Proceedings of the 2nd International Conference on Distributed Systems and Decision (ICDSD'14). Oran, Algeria, pp. 59-66.

96. Jain, S., Chaudhary, H. and Bhatnagar, V. (2013). An information security-based literature survey and classification framework of data storage in DNA, International Journal of Networking and Virtual Organisations (IJNVO), Vol. 13, Issue 2, pp. 176-201.

97. Abouzeid, A., Pawlikowski, K. B., Abadi, D., Rasin, A., and Silberschatz, A. (2009). HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads", In VLDB '09 Lyon, France: VLDB Endowment.

98. Stonebraker, M., Abadi, D., Dewitt, D.J., Madden, S., Paulson, E., Pavlo, A., and Rasin, A. (2010). MapReduce and Parallel DBMSs: Friends or Foes? Journal of Communications of the ACM, Volume 53, Issue 1, 64-71.

99. Castillo, C., Tantawi, A., Steinder, M., and Pacifici, G. (2010). On the Modeling and Management of Cloud Data Analytics. In Danilo Ardagna, Li Zhang. Run-time Models for Self-managing Systems and Applications. Berlin. Springer Basel AG. pp. 153-174.

100. Cloudera (2016). Documentation/Hadoop Tutorial/Example: WordCount v1.0 https:// www.cloudera.com/ documentation/other/tutorial/CDH5/topics/ht_usage.html.

Résumé

La révolution technologique intégrant de multiples sources d'informations, la vulgarisation de l'informatique dans les différents secteurs et domaines ont amené à l'explosion de la volumétrie des données, qui reflète le changement d'échelle des volumes, du nombre et de types. Ces accroissements massifs ont poussé à l'évolution des manières de gestion, de stockage, de localisation et d'accès aux données. Les dernières étapes de cette évolution informatique ont émergé de nouvelles technologies : Cloud Computing et Big Data.

Le Big Data est un ensemble de technologies basées sur les bases de données NoSQL « Not Only SQL » permettant le passage à grande échelle en volumes, en nombres et en types de données. Les grandes entreprises du domaine informatique voient dans les nouveaux systèmes NoSQL, de nouvelles solutions permettant de répondre à leurs besoins d'évolutivité. Plusieurs solutions open-source et payantes de modèles NoSQL sont disponibles sur le marché.

Dans la première contribution, nous développons une étude comparative sur les performances de six solutions NoSQL très répandues dans le marché, à savoir : MongoDB, CouchBase, Cassandra, HBase, Redis, OrientDB à l'aide du Benchmark 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 à savoir YCSB. La finalité est d'apporter l'assistance et l'aide nécessaire aux acteurs intéressés de Big Data et de Cloud Computing pour d'éventuelles prises de décision sur le choix de la meilleure solution appropriée pour leurs entreprises.

Notre deuxième contribution consiste à proposer des remaniements internes dans l'architecture de l'implantation de référence des Clouds de stockage et de Big Data, à savoir HDFS d'Hadoop. Ce dernier adopte un service de métadonnées séparé aux données avec isolation et centralisation des métadonnées par rapport aux serveurs de stockage de données. Nous proposons une approche qui permet d'améliorer le service de métadonnées d'HDFS, afin de maintenir la cohérence sans compromettre les performances des métadonnées en employant un parallélisme modéré des métadonnées, entre centralisation et distribution des métadonnées, dans le but d'accentuer la performance et l'extensibilité du modèle.

Mots clés : Cloud Computing, BigData, NoSQL, YCSB, Hadoop, HDFS, MapReduce, Métadonnées.

Abstract

The technological revolution integrating multiple information sources and extension of computer science in different sectors led to the explosion of the data quantities, which reflects the scaling of volumes, numbers and types. These massive increases have resulted in the development of new locations techniques and access to data. The final steps in this evolution have emerged new technologies : Cloud Computing and Big Data.

Big Data is a set of technologies based on NoSQL databases allowing scalability of volumes, numbers and types of data. The important companies in the IT sector find these NoSQL systems, new solutions to respond to scalability needs. Multiple open-source and proprietary models of NoSQL are available on the market.

In the first contribution, we develop a comparative study about the performance of six solutions widely employed MongoDB, Couchbase, Cassandra, HBase, Redis and OrientDB, using the YCSB tool. The latter being a very powerful test tool, used in multiple evaluation works of NoSQL databases. The purpose is to provide assistance and support to actors interested of Big Data and Cloud Computing for eventual decisions for the choice of solutions to be adopted.

Our second contribution consists in proposing internal revisions in the architecture of the reference implementation of storage clouds and Big Data, namely HDFS of Hadoop. This latter is based on the separation of metadata to data that consists in the centralization and isolation of the metadata of storage servers. We propose an approach to improve the service metadata for Hadoop to maintain consistency without much compromising performance and scalability of metadata by suggesting a mixed solution between centralization and distribution of metadata to enhance the performance and scalability of the model.

Keywords : Cloud Computing, BigData, NoSQL, YCSB, Hadoop, HDFS, MapReduce, Metadata.

ملخـــص

أدت الثورة التكنولوجية التي أدمجت المصادر المتعددة للمعلومات، وكذا تعميم تكنولوجيا المعلومات في مختلف القطاعات والمجالات إلى انفجار حجم البيانات، مما يعكس تغير السلم عل مستوى الأحجام، الأعداد والأنواع. وقد أدت هذه الزيادات الهائلة إلى تغييرات في طرق تسيير، تخزين والتوصل إلى موقع البيانات. المراحل الأخيرة من تطور المعلوماتية أفرز تقنيات جديدة: الحوسبة السحابية والبيانات الضخمة.

البيانات الضخمة هي مجموعة من التقنيات المبنية على قواعد بيانات NoSQL "ليس فقط SQL"، مما يسمح بقياس واسع النطاق في الأحجام والأرقام وأنواع البيانات. ترى الشركات الكبيرة في مجال تكنولوجيا المعلومات، في أنظمة NoSQL الجديدة، حلولًا جديدة لتلبية احتياجات قابلية التوسع، حيث أن العديد من الحلول المفتوحة المصدر والمدفوعة الثمن من طراز NoSQL، متوفرة بقوة في السوق.

في أول إسهام لنا، قمنا بإنجاز دراسة مقارنة لأداء ستة حلول NoSQL جد معروفة في المجال: MongoDB، CouchBase، Cassandra، HBase، Redis، OrientDB باستخدام YCSB Benchmark. هذا الأخير هو أداة معروفة جيدا لقوة اختبارها حيث تستخدم في العديد من أعمال التقييم لقواعد البيانات NoSQL، ألا وهي YCSB. الهدف المرجو هو توفير التوجيهات والمساعدة اللازمة للمهتمين في مجال البيانات الكبيرة والحوسبة السحابية لاتخاذ قرارات محتملة بشأن اختيار أفضل الحلول لشركاتهم.

يتمثل إسهامنا الثاني في اقتراح مراجعات داخلية في بنية التنفيذ المرجعي للحوسبة السحابية والبيانات الضخمة HDFS ل Hadoop، حيث يعتمد هذا الأخير خدمة للبيانات الوصفية منفصلة عن المعطيات المخزنة مع عزل وتمركز البيانات المصنفة عن خوادم تخزين البيانات. نقترح نهجًا يحسِّن خدمة البيانات الوصفية لنظام HDFS للمحافظة على التناسق دون المساس بأداء البيانات الوصفية عن طريق استخدام التوازي المعتدل للبيانات الوصفية، بين المركزية وتوزيع البيانات الوصفية، للتأكيد على أداء وقابلية تطوير النموذج.

الكلمات المفتاحية : NoSQL, YCSB, Hadoop, البيانات الضخمة، الحوسبة السحابية، البيانات الوصفية، HDFS, MapReduce