h e g

SQL, NoSQL, NewSQL stratégie de choix

Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES

par :

Hadrien Furrer

Conseiller au travail de Bachelor :

Rolf Hauri directeur de travail

Genève, le 16 Février 2015

Haute École de Gestion de Genève (HEG-GE)

Filière Informatique de gestion

Hes·so GENÈVE
Haute Ecole Spécialisée
de Suisse occidentale

Déclaration

Ce travail de Bachelor est réalisé dans le cadre de l'examen final de la Haute école de gestion de Genève, en vue de l'obtention du titre d'informaticien de gestion. L'étudiant accepte, le cas échéant, la clause de confidentialité. L'utilisation des conclusions et recommandations formulées dans le travail de Bachelor, sans préjuger de leur valeur, n'engage ni la responsabilité de l'auteur, ni celle du conseiller au travail de Bachelor, du juré et de la HEG.

« J'atteste avoir réalisé seul le présent travail, sans avoir utilisé des sources autres que celles citées dans la bibliographie. »

Fait à Genève, le 16 Février 2015

Hadrien Furrer

Remerciements

Je voudrais remercier mon directeur de travail Monsieur Rolf Hauri de m'avoir suivi durant le temps consacré à cette thèse de Bachelor. J'ai bénéficié de ses conseils qui m'ont indiqué la voie à suivre dans ce travail.

Je tiens aussi à remercier ma famille et mes amis qui m'ont soutenu dans cette dernière épreuve indispensable à l'obtention de mon titre d'informaticien de gestion.

Résumé

Depuis les années 1980, les systèmes de gestion de bases de données relationnelles n'ont cessé de prendre de l'importance en regard des autres systèmes de gestion de données. Aujourd'hui, encore utilisés par la majorité des entreprises ils sont toujours appréciés pour leurs capacités à assurer une forte cohérence des données et garantir une fiabilité lors des transactions. Cependant, l'émergence des systèmes décisionnels et l'explosion des volumes de données à traiter ont conduit beaucoup de sociétés à dénormaliser leur modèle de données. Cette technique visant à regrouper l'information dans des agrégats a pour but d'optimiser les temps de réponses en rompant avec les trois formes normales si chères au SGBDR.

Le Big data a conduit les grands acteurs de l'internet (Google, Facebook, et Amazon etc..) à développer puis adopter des technologies alternatives nommées NoSQL. Celles-ci leurs permettent de supporter une montée en charge horizontale tout en assurant une flexibilité du modèle de données. Dès lors, le NoSQL apparaît comme une solution à l'entreprise désirant gérer des montées en charges et des volumes importants. Cependant, cette technologie sacrifie à dessin la cohérence au bénéfice de la disponibilité. Dans ce modèle, les propriétés ACID sont souvent mises de côté pour la performance. En outre, la flexibilité offerte par le sans-schéma et l'abandon du SQL en font une technologie flexible et particulièrement appréciée des développeurs. Ils découvrent un SGBD où l'application devient maîtresse du schéma de la base de données. Plus d'interminables disputes avec un DBA qui impose un schéma non flexible de la base de données.

La difficulté à gérer la faible cohérence des données pour les développeurs a conduit les grands ténors du web à développer le NewSQL. Ce nouveau SGBDR permet une scalabilité horizontale, une souplesse du schéma et une forte cohérence des données grâce à des transactions ACID. Le NewSQL est aussi jeune qu'il est plein de promesses. Il n'a pas le retour d'expérience des SGBDR et du NoSQL.

Dans le document qui va suivre, sera cité les critères d'adoptions de chaque technologie. Ils seront mis ensemble à la fin dans un tableau de synthèse. Celui-ci pourra orienter la stratégie de choix d'une ou plusieurs d'entre elles.

Contenu

1. Introduction

Depuis les années 1980, les systèmes de gestion de bases de données relationnelles ont pris le pas sur les autres modèles de données (modèle hiérarchique et réseau). Appréciés des entreprises pour sa structuration de données fortement cohérentes et le support de transaction ACID, il est aujourd'hui remis en question par l'apparition de nouveaux besoins liés à l'émergence du Big Data. Ce phénomène implique que certaines entreprises doivent maintenant gérer des volumes gigantesques de données en croissances exponentielles. Répondant à cette problématique, le NoSQL apparaît comme une solution viable mais pas dénuée de défauts. De plus, il présente l'avantage de se décliner en plusieurs types de bases plus adaptées aux besoins spécifiques des entreprises. Cependant, le tout nouveau venu le « NewSQL » semble promettre une architecture combinant les avantages du modèle relationnel et du NoSQL. Dès lors, il semble de plus en plus difficile de choisir les technologies les plus appropriées aux situations. Les questions suivantes sont soulevées:

- Est-ce que l'architecture que j'utilise est la plus adaptée ?
- Est-ce que je dois me tourner vers d'autres architectures?
- Comment choisir l'architecture selon le projet?
- Dans le future technologiquement ou va-t-on?

Par conséquent, ce document vise à orienter leurs stratégies de choix et répondre à ces questions.

2. Le modèle relationnel

2.1 Besoins

Au cours des années 1965-1975 l'utilisation grandissante de l'informatique dans les grandes entreprises à fait naître le besoin d'avoir un modèle de données bien structurées qui sépare la représentation logique des données et leur organisation physique [1].

Le modèle relationnel est né dans les années 1970 du travail d'Edgar Frank Codd. Il publia un article "A relationnal Model of Data for Large Shared Data Banks" qui en pose les bases[2]. Il va à l'encontre des modèles existants (modèle hiérarchique et réseau) qui présentent le désavantage d'avoir des liens forts pour stocker l'information sur le disque (pointeurs physiques).

"Le modèle de données relationnel est aujourd'hui le plus utilisé parce qu'il permet l'indépendance entre la structure physique des fichiers contenant les données et leur organisation logique." (Charrière, n.d.)

En respect des formes normales, il offre une forte cohérence des données et le support de transactions ACID. Nous expliquerons ces dernières notions dans les parties suivantes.

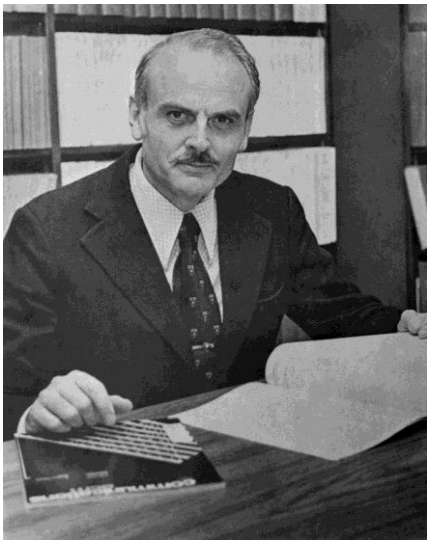


Figure 0.1 Edgar Frank Codd (Clarke, n.d.)

[1] (Wikipedia, 2014a)
[2] (Bruchez, 2013)

2.2 Caractéristiques

Dans cette partie, seront expliqués les caractéristiques essentielles du modèle relationnel.

2.2.1 Représentation

Le passage, ci-dessous, qualifie succinctement bien ce qu'est l'organisation des données du modèle relationnel et sur quoi il est basé.

"Le modèle relationnel est basé sur une organisation des données sous forme de tables. La manipulation des données se fait selon le concept mathématique de relation de la théorie des ensembles, c'est-à-dire l'algèbre relationnelle."

"Les opérations relationnelles permettent de créer une nouvelle relation (table) à partir d'opérations élémentaires sur d'autres tables (par exemple l'union, l'intersection, ou encore la différence)."(Jean-François PILLOU, n.d.)

On représente ainsi les données dans des objets "table" à deux dimensions. Les colonnes (attributs) représentent les valeurs d'un type particulier pour un domaine particulier (p.ex. colonne "nom" de type varchar2). Les lignes contiennent les enregistrements (tuples) de la table. Chacun d'entre eux possède un identifiant qui garantit son unicité.

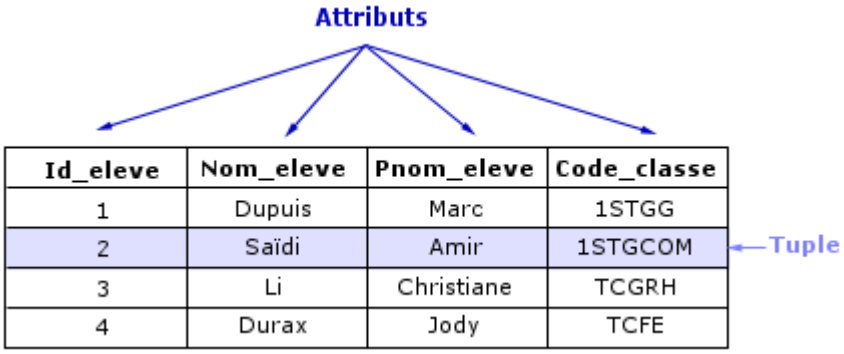


Figure 0.2 Une table élève (Maxicours, n.d.)

Ce modèle supporte trois types de relations entre les tables :

- Relation 1 à 1.

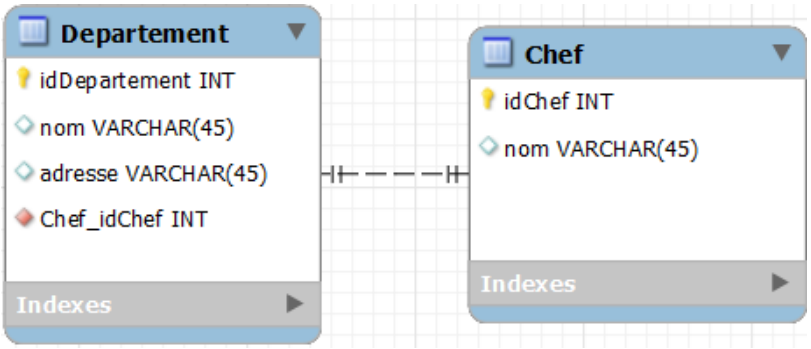


Figure 0.3 notre exemple de relation 1 à 1

- Relation plusieurs à plusieurs.

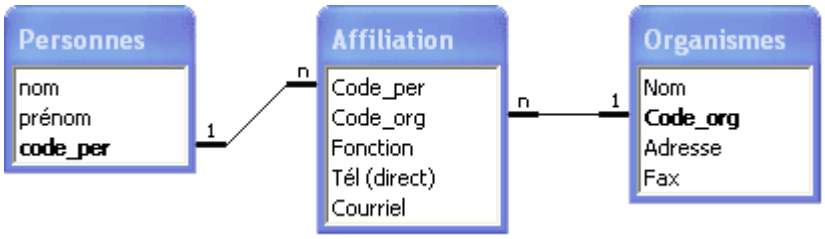


Figure 0.4 relation plusieurs à plusieurs (Sohm, n.d.)

- Relation 1 à plusieurs.

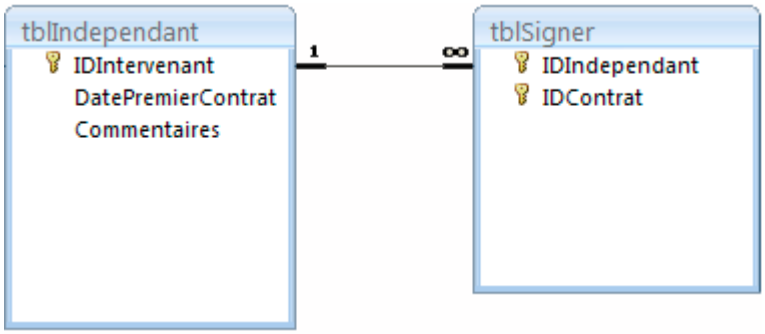


Figure 0.5 relation un à plusieurs (WARIN, 2009)

2.2.2 Langage

Les systèmes de gestion de bases de données supportant le modèle relationnel (SGBDR), utilisent le langage déclaratif "SQL" pour agir sur les données. Ce dernier présente l'avantage d'être devenu un standard pour le dialogue avec les SGBDR.

C'est un langage de définition (LDD), de manipulation (LMD) et de contrôle sur les données (LCD).

LDD : Parce qu'il permet de créer, modifier et supprimer des objets de type "table" en base.

LMD : Parce qu'il permet de lire, modifier, insérer et supprimer des données dans des tables.

LCD : Parce qu'il permet de définir des autorisations aux utilisateurs d'une base de données.[3]

Par une syntaxe facile d'accès (pour un non informaticien) il permet d'effectuer des opérations CRUD sur les données.

"Les instructions SQL s'écrivent d'une manière qui ressemble à celle de phrases ordinaires en anglais. Cette ressemblance voulue vise à faciliter l'apprentissage et la lecture.

C'est un langage déclaratif, c'est-à-dire qu'il permet de décrire le résultat escompté, sans décrire la manière de l'obtenir."(Wikipedia, 2015)

CRUD signifie :

Create : créer une donnée en base.

Read : lire une donnée stockée en base.

Update : Mis à jour d'une donnée stockée en base.

Delete : Suppression d'une donnée stockée en base.

C'est à l'aide de ce langage qu'on commence et termine une transaction ACID. Nous verrons plus loin ce que c'est.

[3] (PILLOU, n.d.)

2.2.3 Formes normales

L'objectif de la normalisation des données est d'éliminer les redondances qui sont sources d'incohérences dans le contexte transactionnel d'un SGBDR[4]. De plus, la suppression des doublons permet de gagner en espace de stockage. Le passage, ci-dessous, explique bien cet objectif.

"Le but essentiel de la normalisation est d'éviter les anomalies transactionnelles pouvant découler d'une mauvaise modélisation des données et ainsi éviter un certain nombre de problèmes potentiels tels que les anomalies de lecture, les anomalies d'écriture, la redondance des données et la contre-performance.

La normalisation des modèles de données permet de vérifier la robustesse de leur conception pour améliorer la modélisation (et donc obtenir une meilleure représentation) et faciliter la mémorisation des données en évitant la redondance et les problèmes sous-jacents de mise à jour ou de cohérence" (Wikipedia, n.d.-a)

Plus on respecte les formes normales, mieux on s'assure une forte cohérence des données[5]. Cette dernière est nécessaire voir indispensable pour beaucoup d'entreprises.

Par exemple: une application de comptabilité doit bénéficier d'une forte cohérence des données[6]. Il serait inconcevable qu'un client possède deux soldes de montant différent pour un même compte.

[4] (S.Louis, n.d.)
[5] (Torres, Gonzalez, & SaraTassini, n.d.)
[6] (Vinette, n.d.)

2.2.3.1 Les 6 formes normales:

2.2.3.1.1 Première forme normale

"Une relation est dite de première forme normale, si tous les attributs de la relation contiennent une valeur atomique." (DURIS, n.d.)

Exemple:

Table : Chauffeur (id, nom, voiture)

455, Yves [Polo GTI 3523, Polo GTI 3523]

Cette table n'est pas en première forme normale car l'attribut voiture n'est pas atomique.

2.2.3.1.2 Deuxième forme normale

"Une relation est dite de deuxième forme normale, si :

- *elle est de première forme normale.*
- *tout attribut non clé n'est pas une partie de clé."*(DURIS, n.d.)

Exemple :

Table : Chauffeur (idChauffeur, Nom, Matricule, DateReussitePermis)

Cette table n'est pas de deuxième forme normale.

Table :Chauffeur (IdChauffeur, Nom)

MatriculeChauffeur (IdChauffeur, Matricule, DateReussitePermis)

En revanche, ces deux tables le sont.

2.2.3.1.3 Troisième forme normale

"Une relation est dite de troisième forme normale, si :

- *elle est 2ème forme normale.*
- *tout attribut n'appartenant pas à une clé ne dépend pas d'un autre attribut non clé."*(DURIS, n.d.)

Exemple:

Table : Chauffeur (idChauffeur, Nom, NumTicket, departVille, arriveeVille)

Cette table n'est pas de troisième forme normale.

Chauffeur (idChauffeur, Nom)

Voyage (NumTicket, departVille, arriveeVille)

En revanche, ces deux le sont.

Le modèle relationnel bien formé doit être en respect au minimum de la troisième forme normale.[7]

Le passage, ci-dessous, explique que les formes normales d'ordre supérieur à trois sont à l'appréciation du développeur et de ses besoins.

"Il existe encore de nombreuses façons de normaliser les données. Le principe qui sous-tend toutes ces normalisations est cependant toujours le même : unicité des dépendances, unicité des actions SQL à effectuer pour la mise à jour d'une donnée unique.

Il appartient aussi à chaque développeur de réfléchir à son problème, et de voir quelle forme normale est la plus simple pour résoudre son problème." (Paumard, n.d.)

[7] (DURIS, n.d.)

2.2.4 Transaction ACID

Une des grandes forces du modèle relationnel, c'est le support des transactions ACID. Le passage suivant de Wikipédia explique très bien ce qu'est une transaction ACID.

"Atomicité

La propriété d'atomicité assure qu'une transaction se fait au complet ou pas du tout : si une partie d'une transaction ne peut être faite, il faut effacer toute trace de la transaction et remettre les données dans l'état où elles étaient avant la transaction. L'atomicité doit être respectée dans toutes situations, comme une panne d'électricité, une défaillance de l'ordinateur, ou une panne d'un disque magnétique.

Cohérence

La propriété de cohérence assure que chaque transaction amènera le système d'un état valide à un autre état valide. Tout changement à la base de données doit être valide selon toutes les règles définies, incluant mais non limitées aux contraintes d'intégrité, aux rollbacks en cascade, aux déclencheurs de base de données, et à toutes combinaisons d'évènements.

Isolation

Toute transaction doit s'exécuter comme si elle était la seule sur le système. Aucune dépendance possible entre les transactions. La propriété d'isolation assure que l'exécution simultanée de transactions produit le même état que celui qui serait obtenu par l'exécution en série des transactions. Chaque transaction doit s'exécuter en isolation totale : si T1 et T2 s'exécutent simultanément, alors chacune doit demeurer indépendante de l'autre.

Durabilité

La propriété de durabilité assure que lorsqu'une transaction a été confirmée, elle demeure enregistrée même à la suite d'une panne d'électricité, d'une panne de l'ordinateur ou d'un autre problème. " (Wikipedia, n.d.-c)

Le respect de l'acidité lors d'une transaction, est un gage de sa fiabilité[8].

[8] (Borderie, n.d.)

2.3 Critères

Dans cette partie, nous allons extraire les critères de choix qui vont conduire à l'adoption du modèle relationnel. Nous le choisirons dans le cas où…:

- la forte cohérence des données est nécessaire.
- on a besoin d'un système transactionnel ACID.
- on veut bénéficier d'un langage unique permettant de définir, manipuler et mettre des droits d'accès sur les données.
- on veut bénéficier d'un langage facile d'accès (pour un non informaticien) pour effectuer des opérations CRUD sur les données.
- on veut une représentation des données sous formes de tables à deux dimensions.
- on veut optimiser l'espace disque.
- on veut utiliser la théorie des ensembles pour manipuler les données.
- on veut un modèle de données bien structuré qui sépare la représentation logique des données et leur organisation physique.
- on veut le standard de modèle de données le plus utilisé.

3. Dénormalisation

3.1 Nouveaux besoins

Dans le chapitre précédent nous avons constaté que les systèmes de gestion de bases de données relationnelles sont fortement cohérents, non-redondants, gèrent les transactions ACID et respectent les trois formes normales. Ils sont, par conséquent, parfaitement adaptés dans un contexte transactionnel ou la forte cohérence des données est indispensable.

Par exemple : une application de comptabilité à l'interne d'une entreprise. Ce logiciel devra s'assurer de la fiabilité d'une opération de débit / crédit. Le modèle relationnel peut lui garantir cela. Comme le souligne Brice Davoleau le modèle relationnel n'est pas adapté dans le cas d'une consultation de données importantes.

"Placez-vous maintenant dans l'optique de consultation de données importante : vous devez mener une analyse portant par exemple sur les deux années précédentes afin de remonter les montants de commandes des clients de la ville de Nantes. Cette requête, si elle s'appuie sur une base de données normalisée, nécessitera de faire des jointures entre les différentes tables afin de remonter des informations en nombre très important. Votre système n'est alors clairement pas optimisé.

Pour faciliter la rapidité d'accès aux données lors de la consultation on dénormalise les tables. Dénormaliser ? Comprenez dupliquer l'information, la répéter, autant de fois que nécessaire afin de faire « descendre » les attributs au plus près des données : de cette façon on simplifie les relations et on optimise l'accès aux données. Si nous reprenons l'exemple de la modélisation du client, il n'existera alors plus qu'une seule table client contenant toutes les informations. On introduit de la redondance. L'intérêt ? Pas ou peu de jointure, et donc accès plus rapide. Revert de la médaille, l'information dupliquée prend beaucoup plus de place. " (Brice Davoleau, n.d.)

Comme le mentionne Monsieur Davoleau le problème des bases de données normalisées lors d'une consultation de données massives vient de la nécessité de réaliser un nombre conséquent de jointures. Car les données sont disséminées et non-redondantes dans un modèle relationnel. La solution semble être la duplication de celles-ci afin d'augmenter la performance des requêtes de lecture. Bien sûr, qui dit duplication dit augmentation du volume de l'information stockée en base de données.

L'explosion des volumes de données et l'augmentation des besoins en matière de Business intelligence et de datamining a conduit nombre d'entreprises à dénormaliser une partie de leur modèle de données. Le passage de Monsieur Tranchant, ci-dessous, évoque clairement les besoins en matière d'informatique décisionnel.

"Conscients que l'une des plus grandes richesses d'une entreprise est son information, mais noyés sous de nombreuses données, éparses, déstructurées et hétérogènes, les dirigeants sont face à une problématique de taille : comment analyser ces informations, dans des temps raisonnables ? Celles-ci concernent-elles toutes les mêmes périodes ? Ces décideurs ont besoin qu'on leur expose les faits importants, base de leurs décisions.

C'est ce à quoi l'informatique décisionnelle (aussi nommée DSS pour Decision Support System ou encore BI pour Business Intelligence) est destinée. ***Elle prend une place en constante croissance dans les systèmes d'information (SI) depuis son apparition, dans les années quatre-vingt-dix.***

L'informatique décisionnelle doit produire des indicateurs et des rapports à l'attention des analystes. Elle doit également proposer des outils de navigation, d'interrogation et de visualisation de l'entrepôt. " (Tranchant, n.d.)

L'informatique décisionnel est souvent synonyme de gros volumes de données. On appliquera le datamining afin d'extraire de la connaissance et des tendances par apprentissage automatique de ces données. Ces informations extraites représenteront des indicateurs clés nécessaires au pilotage de l'entreprise. A l'heure d'aujourd'hui, les solutions de Business intelligence et leurs outils sont de plus en plus demandés par les entreprises. Ces solutions s'appuient sur le modèle dimensionnel en étoile. Il est hautement dénormalisé (entendre informations redondantes). De plus, ces indicateurs fournissent une vue d'ensemble de l'entreprise et une aide à la décision. Des informations capitales ressortent de tels outils. Cela permet à l'entreprise d'opérer des changements de stratégies si besoin est.

3.2 Réponses

Comme vu, ci-dessus, il sera nécessaire de dénormaliser les bases de données afin de pouvoir consulter d'importants volumes de données pour en retirer ou non de la connaissance.

Le passage, ci-dessous, de Yazid Grim explique bien le problème du modèle OLTP en regard de l'analyse des données historiques d'une entreprise.

"Un exemple illustrera mieux le fond de ma pensée.

Imaginez que l'on demande à un analyste d'expliquer le fait que les profits de l'entreprise aient baissé durant les trois derniers mois. Voici un des cheminements que pourrait suivre un analyste :

- *Voir les profits sur l'année pour constater la baisse.*
- *Voir le volume des ventes sur les trois derniers mois : les ventes n'ont pas baissé.*
- *Voir les coûts de revient des produits sur les trois derniers mois : les coûts ont sensiblement augmentés pour certains pays.*
- *Voir les coûts de revient des produits par zone de production sur les trois derniers mois : les produits ayant subi une augmentation sont tous produits en Asie.*
- *Pour les produits en question, pour les trois derniers mois, voir le coût de production par usine, ainsi que le coût moyen de la main d'oeuvre et la taxe et comparer ces valeurs avec celles des trois mois avant l'augmentation : les coûts sont à peu près les mêmes, la raison n'est pas là.*
- *Pour les produits en question, comparer le coût de la matière première avec les chiffres des trois mois avant la hausse des coûts : BINGO ! Le prix de la matière s'est envolé ces trois derniers mois. Il faut maintenant voir les raisons de cette augmentation, revoir les contrats avec les fournisseurs, réguler la production ... Il faut agir !*

Observez bien ce processus. Nous avons commencé par un simple tableau des profits sur l'année et nous avons finis par une synthèse des prix des matières premières pour une liste de produits fabriqués en Asie, et pour les trois derniers mois ... Est-ce que des rapports peuvent supporter un tel processus ? Difficilement, les rapports sont

parfaits pour distiller une information précise, structurée. Mais supporter un tel processus cognitif impliquerait trop de travail (sollicitation perpétuelle du département informatique pour la génération de rapports, perte de temps et d'efficacité d'aller d'un rapport à un autre).

Remarquez aussi les différents niveaux d'agrégation par lesquels notre analyste est passé. L'analyse a commencé par un cumul annuel, et s'est vite transportée vers des cumuls par usine sur une période donnée, sur une zone géographique donnée. Cette charge de travail serait très difficilement supportable par une base de données de production OLTP classique (surtout quand elle implique plusieurs Go de données). Donc la solution impliquerait d'utiliser une structure orienté analyse à l'inverse des bases OLTP qui sont orientées production."(Grim, n.d.)

Les bases de données OLAP (online analytical processing) et leurs axes dimensionnels permettent efficacement de palier au problème de l'OLTP pour faire de l'analyse exégétique sur des masses importantes de données.

"Les applications de type OLAP sont couramment utilisées en informatique décisionnelle, dans le but d'aider la direction à avoir une vue transversale de l'activité d'une entreprise."(Wikipedia, 2014i)

Ces applications OLAP se basent sur un modèle dimensionnel en étoile pour modéliser et ainsi stocker les données dans des entrepôts de données prêts pour l'analyse.

L'historique des activités de l'entreprise sera stocké dans une ou plusieurs tables de fait. Cela représente une quantité d'informations conséquentes. La centralisation de toutes ces données, dans une ou plusieurs tables, a pour but d'avoir une requête performante pouvant être exécutée sur leur globalité. Les dimensions sont les axes d'analyses effectuées sur les données (mesures) de la table de fait. Ils constituent chaque dimension de l'hyper cube utilisé pour l'analyse effectuée sur l'historique de l'entreprise.

Il existe plusieurs façons de dénormaliser un modèle relationnel. Le modèle dimensionnel utilise la fusion par dénormalisation de colonne.

Le passage, ci-dessous, cite les quatre façons de dénormaliser.

"La dénormalisation peut être effectuée de différentes façons :

- *Partitionnement horizontal : utilisé pour diviser une table en plusieurs tables contenant les mêmes colonnes, mais moins de lignes.*
- *Partitionnement vertical : utilisé pour diviser une table en plusieurs tables contenant le même nombre de lignes, mais moins de colonnes.*
- *Fusion de tables : permet de fusionner des tables afin d'éliminer la jointure entre elles.*
- *Dénormalisation de colonne : permet de répéter une colonne dans plusieurs tables afin d'éviter d'avoir à créer des jointures entre les tables."* (Sybase, 2012)

"Il convient d'être prudent lorsqu'on renonce à la forme normale. Il n'est pas garanti qu'une forme dénormalisée améliore les temps d'accès. En effet, la redondance peut entraîner une explosion des volumes de données qui peuvent écrouler les performances ou saturer les disques durs."(Wikipedia, 2014c)

Le modèle dimensionnel mise à fond sur la dénormalisation afin de pouvoir avoir des requêtes avec des temps de réponses acceptables et permettre de faire des analyses selon ses axes.

" Dans les datawarehouse ou datamart, le modèle de données « en étoile » est typique des structures multidimensionnelles stockant des données atomiques ou agrégées. Souvent considéré (à tort) comme un modèle dénormalisé, le modèle en étoile permet une économie de jointures à l'interrogation, ce qui le rend optimisé pour les requêtes d'analyse." (Wikipedia, 2014b)

De plus, la notion de redondance de l'information implique une possible incohérence de celle-ci (comme l'explique le passage de Nicolas Larousse). Des données redondantes peuvent ne pas être à jours. Nous aurons des versions différentes de celles-ci. Par conséquent, il sera nécessaire de mettre à jour toutes les données redondantes avant une lecture qui nécessite une forte cohérence de celles-ci. De surcroît, lors d'une opération de mise à jour d'une donnée il faudra s'assurer de répliquer celle-ci sur les données similaires (qui peuvent être nombreuses).

"La vérification de la cohérence se situe à plusieurs niveaux:

- *.....*
- *Éliminer les problèmes d'incohérence dus à la redondance :* ***en effet la duplication des données rend délicats la maintenance et l'évolution de la relation."*** *(Larousse, 2006)*

3.3 Exemples de mises en œuvres

Pour bien comprendre la technique de dénormalisation il faut partir d'un modèle relationnel. Nous avons des employés qui sont dans des départements et des villes. La table "Employe" référence les tables "Ville" et "Departement". Voir la figure ci-dessous:

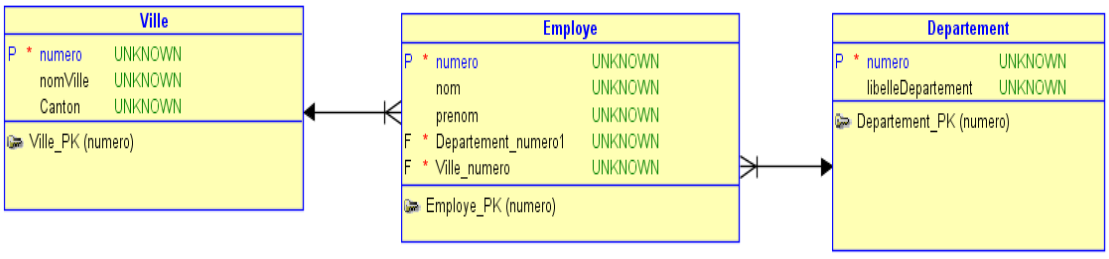


Figure 1.1 mon modèle relationnel

Pour afficher les employés qui sont dans le département "vente" à Lausanne, il faudra dans la requête joindre les tables "Ville" et "Departement" avec la table "Employe".

La requête devra, par conséquent, parcourir l'entier de la table "Ville" pour retrouver la ville Lausanne et l'intégralité de la table "Departement" pour y retrouver le département "vente".

Dans le cas où ces tables contiennent beaucoup de données, leur indexation (les tables) permettra d'accélérer les temps pour la recherche de données au sein de celles-ci. Un moteur de base de données performant permet aussi d'accélérer la recherche d'informations dans une base de données. Cependant, si les temps de réponses restent toujours insatisfaisants après avoir indexé les colonnes des tables affichées par la requête, il faudra dénormaliser le modèle de données.

Voici, ci-dessous, le modèle dénormalisé (figure 1.2) de la figure 1.1

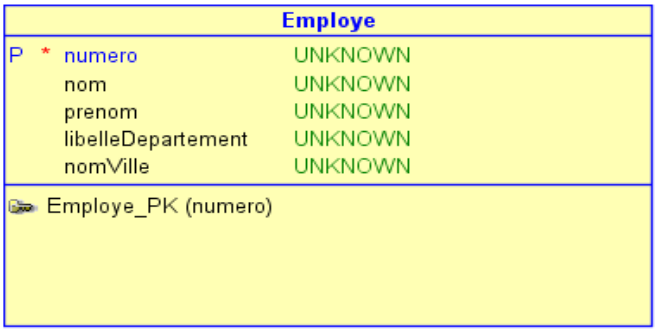


Figure 1.2 mon modèle dénormalisé

J'ai regroupé les colonnes "libelleDepartement" et "nomVille" dans la table "Employe". Cela permet de réaliser notre requête sans jointure. Cette technique permet d'accélérer l'exécution (diminution du temps d'exécution) de celle-ci. Nous avons appliqué la dénormalisation de colonne dans ce cas. De par mon expérience de développeur auprès d'un partenaire de l'ERP "Abacus", j'ai pu constater que cette technique de dupliquer des colonnes dans plusieurs tables est en vigueur dans leurs modèles de données. Cela permet de toute évidence de limiter les jointures fréquentes à réaliser quand on interroge la base de données de ce progiciel de gestion et ainsi augmenter la performance des requêtes en lecture.

3.3.1 Champs calculés

Un autre intérêt de la dénormalisation, c'est la possibilité de stocker des champs calculés dans une table dénormalisée.

Exemple

Imaginons que nous avons une table supplémentaire (niveau de salaire mensuel) dans notre modèle de données normalisés. Celle-ci stocke le niveau de salaire mensuel par classe de l'employé (p.ex. un employé de classe "A" touche 5000CHF par mois et un classe "B" 6000CHF par mois). Si on souhaite connaître le salaire annuel d'un employé XXXX de classe A, il faudra faire une requête qui joindra la table "niveau de salaire" avec la table "Employe" et multipliera par 12 le niveau de salaire de l'employé.

Dans un modèle dénormalisé on peut stocker le salaire annuel de l'employé dans la table Employe. On n'a plus de jointures et de calculs à faire au moment d'exécuter la requête. Par conséquent, on va avoir un meilleur temps de réponse.

3.4 Forces et faiblesses

Parmi ce qui a été dit plus haut, dans ce travail on peut conclure que **les avantages** de la dénormalisation sont:

- Meilleurs performances des requêtes en lecture pour les gros volumes de données.
- Diminution de la complexité des requêtes (diminution du nombre de jointures).
- Approprié et utilisé dans les bases de données multidimensionnelles.
- Plusieurs techniques de dénormalisation.

En revanche, la dénormalisation a comme principaux **désavantages**:

- Redondance de l'information. Donc demande plus d'espace mémoire de stockage.
- Plus de respect des formes normales. Donc possible incohérence des données.
- Difficulté pour opérer des changements sur des données redondantes.

3.5 Critères

Dans cette partie seront extraits les critères de choix qui vont conduire à la dénormalisation d'un modèle relationnel:

- Dans le cas où un besoin de performance sur une requête en lecture se fait sentir. Cependant, il faudra s'assurer que l'indexation des colonnes ne suffit pas à obtenir les données dans un temps de réponse acceptable.
- Analyse de données volumineuses ou de données ayant besoins de partitionnement alors la dénormalisation sera de mise.
- Si l'on cherche à réduire la complexité des requêtes en lecture
- Dans le cas de la constitution d'une base de données multidimensionnelle.
- Dans le cas où l'on dispose de suffisamment d'espace disque
- Si on n'a pas de besoin de forte cohérence des données.

Le besoin de cohérence des données (et par conséquent le respect des formes normales) est souvent un des principaux freins à la dénormalisation. Il faut identifier les données qui ont besoin d'une forte cohérence et les garder dans un modèle relationnel (ou une partie de celui-ci) qui respecte les formes normales.

En conclusion, la dénormalisation se fait en fonction des besoins spécifiques du client. Ce sont les actions voulues sur le modèle de données ainsi que son contenu (entendre quantité d'information) qui vont conduire ou non à la dénormalisation de certaines tables et colonnes.

4. Big Data

4.1 Nouveaux défis

Dans les parties précédentes, le modèle relationnel et la dénormalisation de celui-ci ont été présenté. Nous allons dans ce chapitre aller plus loin en regard des nouveaux défis qu'occasionne la montée en charge subite des bases de données.

Le Big Data, mot inventé par les Anglo-Saxon pour désigner l'explosion des volumes de données, est bien une réalité du XXI^{e} siècle (environ depuis les années deux milles[9]).

Il est décrit par trois termes : Volume, Velocity(rapidité) et Variety (variété).

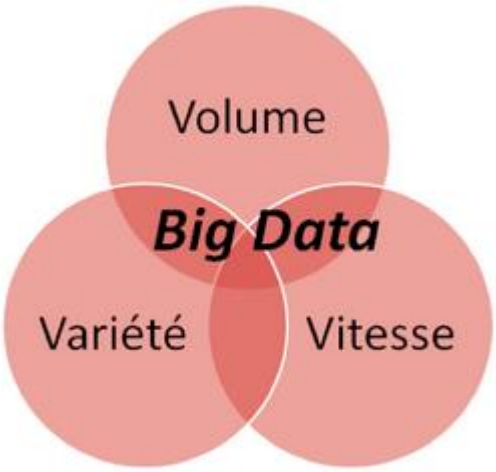


Figure 1.3 les trois "V" du Big Data (Diforti, 2013)

Le volume des données croit de manière exponentielle. Des grands acteurs comme Facebook, Google et Amazon ont été les premiers à constater cette croissance.

"En effet, le monde crée de plus en plus de données chaque jour et ces données sont conservées sur des profondeurs d'historique beaucoup plus importantes. Leur niveau de granularité est de plus en plus fin et de nombreux supports sont dématérialisés." (Diforti, 2013)

La variété des types de données, comme les types semi-structurés (p.ex. format XML et JSON) et les non-structurés (p.ex. photos et vidéos) sont en perpétuelle augmentation due aux nouvelles architectures utilisées.

[9] (Gasnier, 2013)

La vitesse des flux de données. Ils sont en continu. Ils doivent être collectés et analysés en temps réel.[10]

L'augmentation exponentielle de la bande passante facilitée par la diminution du coup des machines et la démocratisation des lignes à haut débits sont un nouveau défi pour de nombreuses sociétés. Le modèle relationnel semble de plus en plus "dépassé" pour gérer ces masses gigantesques de données. Le passage, ci-dessous, de Monsieur "Rudi Bruchez" explique bien ce phénomène.

"*Avec la généralisation des interconnexions de réseaux, l'augmentation de la bande passante sur Internet et la diminution du coût de machines moyennement puissante, de nouvelles possibilités ont vu le jour, dans le domaine de l'informatique distribuée et de la virtualisation, par exemple.*

Le passage au XXI^{e} *siècle a vu les volumes de données manipulés par certaines entreprises ou organismes, notamment ceux en rapport avec Internet, augmenté considérablement. Données scientifiques, réseaux sociaux, opérateur téléphoniques, bases de données médicales, agences nationales de défense du territoire, indicateurs économiques et sociaux, etc.., l'informatisation croissante des traitements en tout genre implique une multiplication exponentielle de ce volume de données qui se compte maintenant en pétaoctets (100'000 téraoctets). C'est ce que les Anglo-Saxon ont appelé le Big Data. La gestion et le traitement de ces volumes de données sont considérés comme un nouveau défi de l'informatique, et les moteurs de bases de données relationnelles traditionnels, hautement transactionnels, semblent totalement dépassés."* (Bruchez, 2013)

De plus, l'avènement de l'internet des objets connectés a fortement contribué à l'augmentation des volumes de données. Ces objets de tous les jours (caméra de surveillance, aspirateur, thermostat etc…) sont maintenant de plus en plus "on line". Ils possèdent une adresse ip et permettent ainsi de converser avec son utilisateur via les standards du web 3.0.

[10] (Diforti, 2013)

Par exemple : voici, ci-dessous, l'exemple d'un lave-linge connecté.

"Le lave-linge connecté, pour sa part, peut être lancé à distance, par exemple en tenant compte des périodes d'heures creuses. Il s'agit d'une alternative au démarrage différé des lave-linge classiques. Si le cycle est terminé et le lave-linge non vidé, le tambour continue à tourner de manière régulière pour éviter que le linge ne se froisse trop et la formation d'odeurs."(Bellamy, 2013)

Comme on peut le voir, ci-dessus, ils étendent leur fonction de base en utilisant internet pour fournir un service à même de satisfaire au mieux leurs utilisateurs. Ainsi notre lave-linge peut être lancé par un utilisateur via son smartphone.

De surcroit, l'avènement des smartphones et leur utilisation toujours plus intensive du web ajoutent un pavé dans la mare. L'article de Guillaume Sarlat explique bien cette problématique.

"Leur activité génère en effet d'énormes volumes de données. Ainsi d'après Eric Schmidt, son ex-PDG, Google crée tous les deux jours autant d'information qu'il en a été créé de l'origine du monde à l'année 2003. Et ces données sont peu structurées : texte libre, musique, photos, vidéos, etc. De ce fait, les outils traditionnels, à savoir les entrepôts de données (data warehouses) classiques et les gestionnaires de bases de données dites relationnelles sont mal adaptés pour les stocker et les traiter." (Guillaume Sarlat (inspecteur des finances), 2014)

Il apparait que les solutions utilisant des bases de données relationnelles ou des entrepôts de données classiques ne sont pas adaptées pour stocker, traiter et lire ces masses conséquentes d'informations de façon efficace.

A noter que "entrepôt de données classiques" signifie qu'il s'appuie sur la dénormalisation d'un modèle relationnel. Il est utilisé pour stocker des données relatives à l'historique de l'entreprise. Il n'y a pas (encore) d'utilisation des "outils" du Big Data pour stocker, traiter et faire de l'analyse en temps réel de flux continu de données.

4.2 Besoins à couvrir

Comme mentionné plus haut, les solutions de bases de données relationnelles et les entrepôts de données classiques ne sont pas en mesures de faire face au Big Data. Ces premières imposent un modèle de données très peut flexibles ainsi qu'une forte cohérence des données. Nous l'avons vu dans le chapitre lié à la dénormalisation que c'est un frein à la performance des requêtes en lecture. Les entrepôts de données classiques s'appuient sur des bases de données relationnelles pour stocker des données **structurées** relatives à l'historique de l'entreprise dans des tables de fait.

Cela pose un problème car, l'explosion des volumes de données toujours plus **non-structurées** (vidéo, images, son, tweet etc…) est un constat. Il est nécessaire de changer ou adapter l'architecture existante pour stocker, traiter et analyser celles-ci[11].

Par conséquent, les entrepôts classiques ne sont pas adaptés pour le Big data. Ils doivent être repensés afin de pouvoir intégrer d'autres sources d'informations de type non-structuré et de faire de l'analyse prédictive en temps réel dessus. Ce type d'analyse est de plus en plus recherché par les entreprises. Elle a pour but de fournir des indicateurs clés tactiques (sur des données actuelles souvent non-structurées). L'objectif, n'est pas seulement d'offrir à la direction des informations uniquement sur l'historique des données de l'entreprise mais aussi sur des flux de données diverses structurées ou non.[12]

Par exemple : connaître les recherches et actions des utilisateurs de quinze à vingt-quatre ans d'un réseau social (publication de photos, like, site web visités avant d'arriver sur le réseau social etc…).

A noter que, Ronald Moulanier, architecte en gestion de l'information chez IBM, soutient que l'idée n'est pas de substituer une plateforme de Big data à un entrepôt de données mais de compléter ce dernier. Il faut intégrer les flux de données continus non structurés et utiliser les outils du Big data pour permettre de les stocker dans l'entrepôt et de faire des analyses sur celles-ci.[13]

[11] (Lévy-Abégnoli, 2013)
[12] (Serries, 2014)
[13] (Moulanier, n.d.)

Un autre besoin lié à l'explosion des volumes de données, c'est la capacité à gérer cette montée en charge horizontalement[14]. On doit pouvoir être capable d'ajouter un nouveau serveur facilement (en se souciant le moins possible de la répartition des données) pour distribuer la charge sur celui-ci. Les bases de données relationnelles sont incapables de gérer une montée en charge horizontale efficace sur des volumes conséquents de données. Le Big Data et ses outils répondent à ce besoin.

De plus, l'évolution constante des types de données ainsi que la croissance des données non-structurées rendent impossible leurs intégrations dans un modèle relationnel avec un schéma fixe. Par conséquent, pour gérer les données hétérogènes et non structurées du Big Data, il est important d'avoir un modèle de données flexible.[15]

Pour conclure, aujourd'hui il est constaté qu'il est de plus en plus nécessaire d'avoir accès à l'information rapidement en tous lieux et à tout moment. Les utilisateurs recherchent une haute disponibilité et des temps de réponses adéquates. Il faut donc une architecture capable de gérer des volumes conséquents de données structurées ou non et permettre de faire remonter l'information rapidement à l'utilisateur. Il est nécessaire pour cela qu'elle soit capable de paralléliser le traitement des requêtes[16].

En résumé, les besoins principaux liés à l'émergence du Big Data sont:

- La capacité à gérer une montée en charge horizontale.
- La parallélisassions des traitements des requêtes.
- Un schéma de données flexible.
- La capacité de stockage, traitement et lecture de données non structurées et semi-structurées.
- La performance des requêtes en lecture.
- La haute disponibilité des données.
- Analyse prédictive sur des flux de données continues.

[14] (Lévy-Abégnoli, 2012)
[15] (CouchBase, 2014c)
[16] (Wikipedia, 2014g)

5. Le NoSQL

5.1 Réponses au Big data

En réponse à l'émergence du Big data, les grandes sociétés de l'internet (Google,Facebook, Amazon etc...) ont chacune développé des solutions alternatives au SQL. Ces dernières ont pour but de répondre aux besoins spécifiques de ces acteurs de l'internet. Les premiers à combler furent la capacité à gérer la montée en charge exponentielle de leur service et la distribution des traitements sur un grand nombre de nœuds. Par conséquent, Google sera le premier en 2004 à proposer un outil nommé "Map/Reduce" qui permet une distribution automatique sur un grand nombre de nœuds de tâches à effectuer. Le succès de ce patron d'architecture engendrera son implémentation java libre Hadoop. Soutenue par Facebook et Yahoo, elle s'inspire du système de fichier distribué GFS (de Google). Hadoop utilise les fonctions de base de Map/Reduce ainsi que son propre système de fichier HDFS.

"Le HDFS est un système de fichiers distribué, extensible et portable développé par Hadoop à partir du GoogleFS. Écrit en Java, il a été conçu pour stocker de très gros volumes de données sur un grand nombre de machines équipées de disques durs banalisés. 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." (Wikipedia, 2014d)

Hadoop permet en combinant le HDFS et le MapReduce d'avoir un système qui gère le stockage et la montée en charge horizontale de grands volumes ainsi qu'une répartition automatique des opérations sur des clusters de nœuds. De plus, il assure la haute disponibilité des données et facilite leur déplacement dans la grappe de serveur.[17] On comprend ainsi mieux son attrait et son utilisation grandissante dans les sociétés qui cherchent à relever les défis imposés par le Big Data.

Les bases de données NoSQL utilisent pour la plupart le pattern MapReduce voir le système Hadoop basé sur celui-ci (c'est notamment le cas de HBase et CouchBase). Elles furent développées par les grands acteurs de l'internet afin de répondre aux besoins engendrés par le Big Data. Elles ont été présentées au public lors d'un important meeting en Juin 2009 à San Francisco.

[17] (Shvachko, n.d.)

"La rencontre meetup NoSQL de San Francisco du 11 juin 2009 a été particulièrement importante pour le développement de cette tendance. Plus de 100 développeurs de logiciels ont assisté à des présentations de solutions telles que Project Voldemort, Cassandra Project, Dynomite, HBase, Hypertable, CouchDB et MongoDB. Le concept du NoSQL avait cependant déjà une bonne décennie d'ancienneté.

….

La rencontre de 2009 à San Francisco est considérée comme l'inauguration de la communauté des développeurs de logiciels NoSQL. Des développeurs qui, selon le magazine Computerworld, " racontent comment ils ont renversé la tyrannie des coûteux et lents SGBD relationnels par des moyens plus simples et plus rapides de manipuler des données ". " (Wikipedia, n.d.-b)

Elles offrent un modèle de données flexibles et permettent de gérer, stocker et traiter les données semi-structurées et non-structurées que nous utilisons de plus en plus aujourd'hui. Le "shéma-less" offert par la plupart d'entre elles permet une souplesse et une flexibilité accrue au développeur d'applications. C'est lui au travers de son programme qui va gérer le modèle de données. Il devient responsable du schéma des données. Plus de luttes incessantes entre les développeurs d'applications qui ont besoins de flexibilités sur les données métiers à représenter et leurs règles et un DBA soucieux de respecter les formes normales. Comme le montre le passage ci-dessous le "sans schéma" offre une grande liberté au développeur et lui permet de stocker des données de types variées.

"L'intérêt des systèmes de stockage NoSQL réside surtout dans les choix d'architecture logicielle qui ont été pris lors de leurs conceptions. Parmi les raisons principales qui ont mené à la création de ces systèmes, on retrouve surtout deux points principaux :

- *La possibilité d'utiliser autre chose qu'un schéma fixe sous forme de tableaux dont toutes les propriétés sont fixées à l'avance ;*
- *La possibilité d'avoir un système facilement distribué sur plusieurs serveurs et avec lequel un besoin supplémentaire en stockage ou en montée en charge se traduit simplement par l'ajout de nouveaux serveurs."*

"Le schéma flexible apporte une plus grande liberté au développeur et lui permet de stocker de façon optimale des ensembles de données dont les entrées peuvent être très disparates." (DORY, 2012)

Elles offrent une haute disponibilité de données très volumineuses et variées dans des temps de réponses bas en se souciant le moins possible de la répartition de celle-ci sur les nœuds. Les directeurs des systèmes d'informations et architectes d'entreprise s'enthousiasment de disposer enfin d'outils permettant de gérer les flux continus de données en temps réel et de faire de l'analyse prédictive sur celles-ci.[18]

De surcroît, ces bases de données s'écartent du langage déclaratif SQL pour manipuler les données au profit des langages procéduraux et objets préférés des développeurs de logiciel. De plus, l'accès aux données est facilité car la dénormalisation sera de mise. Ainsi les requêtes complexes se simplifient.

Finalement, on s'aperçoit que les bases de données NoSQL et les outils du Big Data qui viennent s'y greffer ou les compléter (par exemple: Hadoop) répondent correctement aux défis engendrés par l'explosion des volumes de données. [19]

Nous allons voir dans la partie suivante "Technologies associées" plus en détails ce que sont les "outils" du Big data.

[18] (Lublinsky, 2014)
[19] (CHEMINAT, 2013)

5.2 Technologies associées

5.2.1 CAP

Avant d'aborder les différents types de base de données NoSQL, il est important d'expliquer quel théorème elles suivent. Alors que le modèle relationnel obéit aux formes normales et aux règles ACID, les bases de données NoSQL suivent le théorème CAP. Les principes évoqués, ci-dessous, par Masclet l'expliquent bien.

"En voici les 3 principes :

Consistance (consistency /C) : Tous les clients voient la même vue même lorsqu'il y a des mises-à-jour, en fait il s'agit du 'Atomic' des propriétés ACID.

Haute disponibilité (availability /A) : L'ensemble des clients peuvent trouver des données répliquées, même lorsqu'une avarie survient

Tolérant à la partition (partition-tolerance /P) : Le système est tolérant au partitionnement. (Ndlr : on entend ici que les données peuvent être distribuées sur plusieurs machines)

En revanche, seul deux des trois principes peuvent être respectés en même temps, souvent ce sont les deux derniers principes qui sont choisis (c'est le cas d 'Amazon par exemple) car le besoin en disponibilité et en partitionnement est souvent plus important que la consistance." (Masclet, 2010)

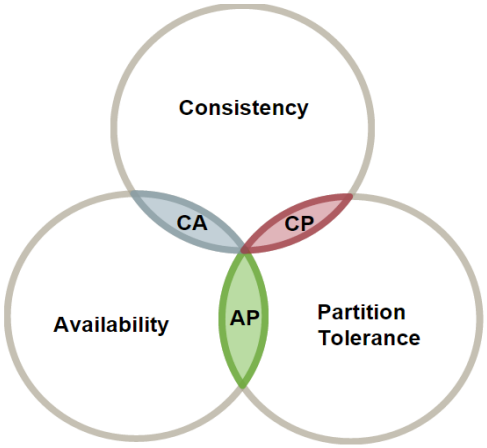


Figure 1.4 Cap Theorem (Erb, n.d.)

Comme mentionné plus haut, les bases de données NoSQL sont tenues de respecter deux principes sur trois. Elles choisiront un des trois cas en fonction des besoins à combler:

- Cohérence et haute disponibilité
- Haute disponibilité et partition tolerance
- Cohérence et partition tolerance

5.2.2 MapReduce

Nous avons vu précédemment que MapReduce permet une distribution automatique sur un grand nombre de nœuds de tâches à effectuer. C'est un patron d'architecture qui est adapté au traitement de volumes conséquents de données.

Il est composé de deux primitives:

- **Map :** Le nœud parent décompose un problème en sous problème sous forme de paires clés/valeurs et les délègues à d'autres nœuds qui pourront en faire de même. [20]
- **Reduce:** Une fois la fonction Map réalisée, les nœuds les plus bas font remonter les résultats des traitements aux nœuds parents.[21]

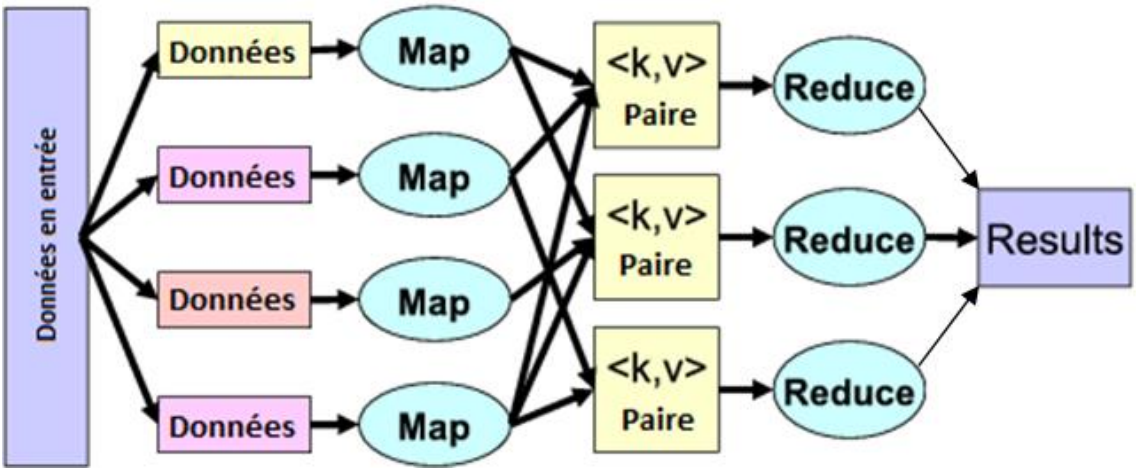


Figure 1.5 le MapReduce (Wikipedia, 2014e)

Comme nous le verrons plus loin, Hadoop et la majorité des bases de données NoSQL utilisent le MapReduce.

[20] (Wikipedia, 2014e)
[21] (Wikipedia, 2014e)

5.2.3 Hadoop

Hadoop est un framework libre en java. Il est adopté par un grand nombre d'entreprises qui ont développé des applications distribuées et scalables[22]. Il est capable de travailler avec un grand nombre de nœuds et un volume conséquent de données. Il se base sur MapReduce pour distribuer automatiquement aux nœuds les opérations à réaliser. Son principal atout, c'est sa capacité à gérer une montée en charge horizontale sans autre contrainte pour le responsable de l'architecture que l'ajout de nœuds supplémentaires au cluster. Il offre une abstraction complète du stockage sur les nœuds. Par conséquent, il le voit comme un unique disque de mémoire. De plus, il favorise la haute disponibilité des données par un système de réplications automatiques de celles-ci sur les nœuds[23].

Le paragraphe, ci-dessous, de Baron explique bien les caractéristiques de Hadoop.

"D'une part, il propose un système de stockage distribué via son système de fichier HDFS (Hadoop Distributed File System) et ce dernier offre la possibilité de stocker la donnée en la dupliquant, un cluster Hadoop n'a donc pas besoin d'être configuré avec un système RAID qui devient inutile. D'autre part, Hadoop fournit un système d'analyse des données appelé MapReduce. Ce dernier officie sur le système de fichiers HDFS pour réaliser des traitements sur des gros volumes de données.(BARON, 2014)

De plus, le passage ci-dessous, de Damais explique bien le fonctionnement de Hadoop.

"Le principe de fonctionnement d'Hadoop est assez simple. L'infrastructure applique le principe bien connu des grilles de calcul, consistant à répartir l'exécution d'un traitement sur plusieurs nœuds, ou grappes de serveurs. Prenons l'exemple d'une liste d'utilisateurs, qui serait celle d'un service sur Internet - par exemple celle des utilisateurs de Yahoo! ou de Facebook (qui utilise également la solution Open Source). Avec, en ligne de mire, l'objectif de réaliser un comptage du nombre d'utilisateurs inscrits à ces réseaux sociaux par exemple.

Dans une logique d'architecture Hadoop, cette liste est découpée en plusieurs parties, chaque partie étant stockée sur une grappe de serveurs différente. Au lieu d'adosser le traitement à une grappe unique, comme c'est le cas pour une architecture plus

[22] (Wikipedia, 2014d)
[23] (Shvachko, n.d.)

traditionnelle, cette distribution de l'information permet ainsi de répartir ce traitement sur l'ensemble des nœuds de calcul sur lesquels la liste est répartie." (Damais, 2012)

En résumé, Hadoop gère la distribution et la réplication automatique de son stockage à l'aide du HDFS et la distribution automatique des traitements aux clusters grâce au MapReduce.

De plus, on préfère l'utiliser avec du matériel serveur bon marché afin de limiter les coûts. La parallélisassions automatique des traitements sur ceux-ci et la montée en charge horizontale aisée en font une technologie rentable financièrement. Il n'est pas nécessaire d'avoir un matériel serveur très haut de gamme (et donc cher) pour peupler un cluster Hadoop. [24]

Il peut être utilisé dans plusieurs domaines allant du stockage et traitement d'importants volumes de données au reporting décisionnel en passant par l'analyse de log et du trafic d'un site web, l'archivage et l'indexation de données semi/non-structurées.[25]

Par conséquent, Hadoop est:

- Flexible car il est sans schéma. Il est capable de stocker et traiter sans contraintes n'importe quels types de données.
- "Scalable"
- Assure la disponibilité de l'information.
- "Cost effective"

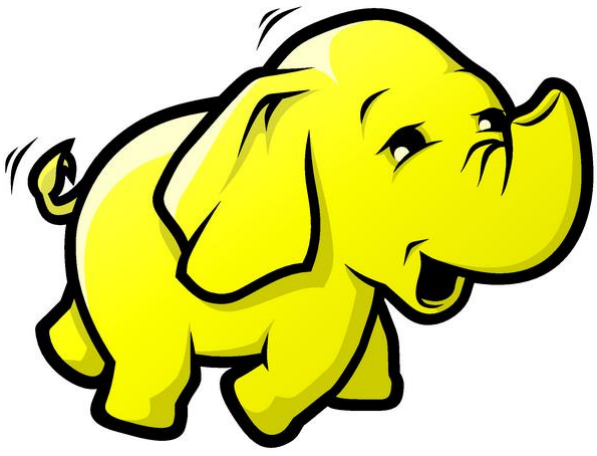


Figure 1.6 mascotte de Hadoop(Google, n.d.)

[24] (IBM, n.d.)
[25] (Damais, 2012)

Surfant sur le succès de Hadoop plusieurs sous projets furent créés.

- **Hbase** : c'est une base de données NoSQL qui s'installe sur le système de fichier HDFS. Elle est de type "entrepôt clé-valeur".
- **Pig :** il fournit un langage de haut niveau (un DSL) qui est parfaitement adapté à l'analyse de gros volumes de données[26]
- **Sqoop :** ce projet a pour but de rendre possible le dialogue entre un SGBDR et Hadoop. Il est ainsi possible d'importer ou d'exporter des données vers une base de données relationnelle.
- **Mahoot:** est projet d'informatique décisionnel. Il vise à permettre une classification et un apprentissage automatique des données sur un cluster Hadoop de nœuds grâce à l'utilisation d'algorithmes.

[26] (BARON, 2014)

5.2.4 Types de bases de données NoSQL

Il existe plusieurs types de bases de données NoSQL. Cependant, elles ont toutes un point commun, c'est l'abandon des contraintes proches des données. Ces bases stockent soit des paires clé-valeur soit des documents JSON[27]. Elles ne possèdent pas de schémas et ainsi de contraintes sur les objets. Le développeur est libre de stocker et organiser les données comme il veut en base. C'est lui qui définit au travers de son application la structure et les règles des éléments qui y seront manipulés et sauvés.

Les types de base de données NoSQL sont:

- Les entrepôts clé-valeur
- Les bases orientées documents
- Les bases orientées colonnes
- Les bases orientées graphes

Dans les parties suivantes, il sera expliqué les différents types de base de données NoSQL. Puis, des exemples succincts d'implémentations pour chacun d'entre eux seront cités.

[27] (Bruchez, 2013)

5.2.4.1 Les entrepôts clé-valeur

Le passage, ci-dessous, de Chaghal explique bien ce qu'est une base de données NoSQL de type entrepôt clé-valeur.

"Ce modèle peut être assimilé à une hashmap distribuée. Les données sont, donc, simplement représentées par un couple clé/valeur. La valeur peut être une simple chaîne de caractères, un objet sérialisé… Cette absence de structure ou de typage ont un impact important sur le requêtage. En effet, toute l'intelligence portée auparavant par les requêtes SQL devra être portée par l'applicatif qui interroge la BD. Néanmoins, la communication avec la BD se résumera aux opérations PUT, GET et DELETE. Les solutions les plus connues sont Redis, Riak et Voldemort créé par LinkedIn." (Chaghal, 2011)

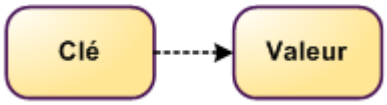


Figure 1.7 une valeur identifiée par un clé (Figuière, 2010d)

Ce modèle est le plus simple des différents types de base NoSQL. Il présente un avantage conséquent, il est facilement scalable horizontalement. Lors de l'ajout d'un nouveau nœud, il suffira de redéfinir les intervalles des clés dont les serveurs du cluster sont responsables.

Comme mentionné dans le passage, ci-dessous, ce type de base de données présente d'excellentes performances en lecture/écriture.

"La représentation en clé-valeur est la plus simple et est très adaptée aux caches ou aux accès rapides aux informations. Elle considère la valeur stockée comme un bloc de données opaque, la base de données étant agnostique de son contenu. Ce postulat permet en général d'atteindre des performances bien supérieures dans la mesure où les lectures et écritures sont réduites à un accès disque simple."(Figuière, 2010d)

En revanche, comme mentionné ci-dessus, il aura un impact sur le requétage. On ne peut effectuer une requête que sur la clef, ce qui implique que la valeur est retournée

dans son entier. Il n'est malheureusement pas possible de récupérer une partie de celle-ci.[28]

Par exemple: on stocke un objet "Personne" avec un nom et un prénom. On ne peut pas récupérer uniquement le prénom de celle-ci, l'objet est retourné dans son ensemble.

Les opérations se résument aux commandes Put, Get et Delete qu'offrent le protocole HTTP REST. Ce protocole est sans état. Il ne garde aucune mémoire de la requête précédente. Le passage, ci-dessous, explique ce qu'est REST.

"REST est un style d'architecture qui repose sur le protocole HTTP : On accède à une ressource (par son URI unique) pour procéder à diverses opérations (GET lecture / POST écriture / PUT modification / DELETE suppression), opérations supportées nativement par HTTP." (Gerald, 2011)

Ce modèle est principalement adopté comme dépôt de données ou système de stockage en cache mémoire (comme Redis).

Pour résumé ses points forts sont donc :

- Montée en charge horizontale performante et très facile
- Opérations de lecture et écriture très performantes.
- Parfaitement adapté pour les caches mémoire

En revanche son point faible c'est:

- Qu'on ne peut agir que sur la valeur dans son entier. On ne peut pas réaliser d'opérations CRUD sur une partie de la valeur.

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

[28] (Girolamo, 2013)

5.2.4.1.1 Base de données NoSQL Redis

Figure 1.8 Logo de redis (Martignole, 2013)

Redis est une base de données de type clé-valeur. Le passage, de Bruchez ci-dessous, décrit succinctement bien Redis.

> *"Redis est un excellent choix pour maintenir des données en mémoire pour un accès en temps réel très rapide. C'est une forme de remplacement d'un cache tel que memcached pour offrir une plus grande richesse fonctionnelle et une manipulation de structure de données plus riche."* (Net, 2013c)

Elle supporte une chaîne binary-safe à l'intérieur de laquelle il est possible de stocker des valeurs numériques et binaires. Depuis ce type de base, elle offre des listes, des ensembles triés ou pas et des tables de hachage. Ces différentes structures sont optimisées en mémoire. Il n'y a pas de principe de cohérence à proprement parler dans Redis. Les données étant stockées en mémoire, cela confère à cette base un accès d'une extrême rapidité. En cas de coupure de courant deux systèmes peuvent lui permettre de récupérer les données.

- RDB
- AOF

RDB va écrire un fichier sur le disque contenant toute la base de données en mémoire. Le transport de cette base en est ainsi facilité. Cependant, cela n'est pas adapté pour enregistrer une transaction. Il faudra faire un enregistrement de la base pour sauver la transaction désirée. On peut ainsi comprendre que ce n'est pas préconisé pour faire autre chose que des backups complets périodiques.

AOF fonctionne à la manière des "redos log d'Oracle". C'est un journal dans lequel les transactions sont sauvées dans un temps répété et déterminé. [29]

[29] (HEINRICH, 2012a)

5.2.4.1.2 Base de données NoSQL Riak

Figure 1.9 logo de riak (Zeeman, 2013)

Riak est une base de données NoSQL de type entrepôt clé-valeur. Les passages, ci-dessous, décrivent succinctement bien Redis.

"Riak est un entrepôt de paires clé-valeur bien conçu, solide, performant et qui monte très bien en charge. C'est un excellent choix pour un moteur qui doit à la fois être distribué et offrir une latence faible, sur des systèmes qui doivent être prêts à monter en charge très rapidement et de façon automatique." (Journaldunet, 2013d)

***"Types de données manipulées** : paires clé-valeur. La valeur est normalement opaque, mais elle peut être indexée si c'est du JSON.*

***Mode de distribution** : décentralisé par hachage consistant, à la manière de Dynamo d'Amazon. Pas de serveur maître et chaque nœud est indépendant. Les connexions utilisateurs peuvent s'effectuer sur n'importe quel nœud, qui se chargera de rediriger le client vers le nœud contenant les données souhaitées.*

***Protocole** : Protobuf et REST. Protobuf est bien sûr à privilégier pour de meilleures performances. L'interface REST est intéressante pour les opérations d'administration ou un accès aux données par des applications mobiles."*(Journaldunet, 2013d)

5.2.4.2 Les bases orientées colonnes

Les bases orientées colonnes fonctionnent par familles de colonnes. Elles représentent l'équivalent de sous-tables. Chaque ligne d'une "table" comporte exactement le nombre de colonnes qu'elle a besoin pour stocker ses valeurs. Par conséquent, plus de NULL stockés dans des enregistrements vides comme c'était le cas dans les SGBDR (à noter qu'un NULL occupe quand même de l'espace mémoire). Nous parlons ici de tables possédant des millions de colonnes. Par conséquent, le gain en espace mémoire est significatif. Ce type de base de données est hautement dénormalisée et s'apparente au système de stockage en table dans les SGBDR. En outre, les données ne sont plus stockées sur deux dimensions (ligne X colonne) mais sur une, la colonne.

Le passage, ci-dessous, explique bien ce concept.

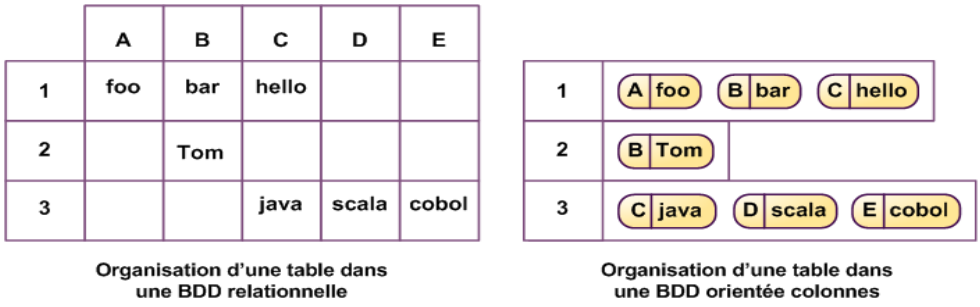


Figure 2.0 les bases orientées colonnes (Figuière, 2010b)

"En fait, les bases de données orientées colonnes sont pensées pour accueillir un grand nombre de colonnes (jusqu'à plusieurs millions) pour chaque ligne. Dès lors on comprend qu'il ne s'agit plus simplement d'y stocker les champs d'une entité, mais également des relations one-to-many.

Les requêtes possibles sont simples. Il est possible de faire des requêtes par clé, ou ensemble de clés et d'y adjoindre un prédicat sur un intervalle de colonnes. Voici quelques exemples de requêtes possibles :

- *Toutes les colonnes de la ligne dont la clé est 12*
- *Toutes les colonnes dont le nom est comprises entre "aaa" et "abb" pour la ligne dont la clé est 156*
- *La colonne "aaa" pour les lignes 26 à 31*

On dispose donc d'un système de requêtes minimaliste qui a permis de grandement simplifier le design de ces bases de données, au profit de la performance." (Figuière, 2010b)

Les colonnes étant stockées de façon triées, cela permet d'obtenir un intervalle de colonnes. On réduit ainsi le temps et les accès aléatoires sur le disque.

A noter qu'on retrouve l'identifiant de l'enregistrement auxquelles appartient la donnée pour chaque colonne (voir figure ci-dessous). On peut ainsi imaginer la simplicité des requêtes en lecture.

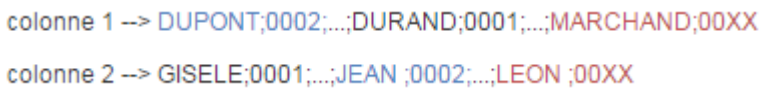


Figure 2.1 Stockage par colonnes triées. (Becquet, 2011)

De plus, grâce à son grand nombre de colonnes, ce type de bases de données est parfaitement adapté aux relations de type one-to-many. De surcroît, les requêtes simplistes destinent ces bases aux applications et services qui se contenteront d'accès simplifiés au profit d'une performance et d'une scalabilité augmentée. Elles sont optimisées pour l'écriture (elles écrivent plus rapidement qu'elles ne lisent). Par conséquent, il faut favoriser leur utilisation quand on doit surtout stocker des données.[30]

Pour conclure, Michael Morello résume bien dans le passage ci-dessous, les avantages et inconvénients de ce type de base (des passages non indispensables ont été supprimés).

"Avantages :

- *L'ajout de données se fait sur une seule dimension (Ndlr: la colonne) ce qui est techniquement plus simple et plus rapide ...*
- *On devient à la mode en étant "scalable" (utilisez votre meilleur accent anglais pour faire bien) : comme le développement des données ne se fait que sur une seule dimension leurs partitionnements est plus simple à réaliser et on peut les distribuer sur plusieurs serveurs …*
- *Ajouter une colonne devient trivial car il s'agit au final seulement d'ajouter un nouveau tuple…*

[30] (Wordpress (Auteur inconnu), 2013)

Inconvénient :

- *…Autrement dit : il y a une seul clé (Ndlr : clé primaire), seul champ véritablement indexé, les recherches sur les autres colonnes seront moins efficaces…*
- *Si ce type de base de données est très efficace en écriture elle l'est moins en lecture, non seulement on se retrouve avec une seule clé mais on voit aussi assez vite que si l'on veut être performant et pouvoir distribuer les données il va falloir mettre un peu d'ordre en les triant et user d'outil tels que les filtres de M. Bloom afin d'optimiser les recherches.*
- *Une mise à jour n'écrase pas directement l'ancienne valeur, il faut donc prévoir à un moment ou un autre un processus appelé "compaction" pour réellement supprimer les données… "* (Morello, 2012)

HBase et Cassandra sont les principales bases de données NoSQL.

5.2.4.2.1 Base de données NoSQL HBase

Figure 2.2 logo de HBase (Apach, 2014)

HBase est une base de données NoSQL de type colonne propriété de l'entreprise Apache qui monte très bien en charge et permet de gérer un important volume de données. Le passage, ci-dessous, explique succinctement bien HBase.

"HBase est un choix intéressant uniquement si vous prévoyez de gérer un volume de données très important. Basé sur Hadoop, il permet de distribuer les données en utilisant le système de fichiers distribué HDFS (Hadoop Distributed File System) d'Hadoop. Cela n'a donc pas de sens de vouloir utiliser HBase sur un système non distribué. De plus, sa mise en œuvre est relativement complexe."(Net, 2013a)

Elle garantit une transaction ACID sur une ligne (donc sur une famille de colonnes). Par conséquent, une commande qui réalise des changements sur plusieurs lignes n'est pas atomique. C'est une suite de mises à jour de chaque ligne qui vont retourner individuellement un message de succès ou d'erreur.

HBase offre une excellente tolérance au partitionnement de données. Facebook l'utilise pour stocker tous ses messages.

*"**Mode de distribution:** basé sur Hadoop et HDFS, avec maître centralisé. Un cluster HBase est constitué d'un maître (HMaster) qui maintient les métadonnées du cluster et gère des serveurs de région (RegionServer). Chaque région contient une partie des données. "* (Net, 2013b)

5.2.4.2.2 Base de données NoSQL Cassandra

Figure 2.3 logo de Cassandra (Cassandra, 2014)

Cassandra est une base de données NoSQL de type colonne propriété de l'entreprise Apache appropriée pour gérer de manière décentralisée d'importants volumes de données. C'est un produit qui a fait ces preuves. Il est utilisé par beaucoup d'entreprise. Les passages suivant décrivent bien Cassandra.

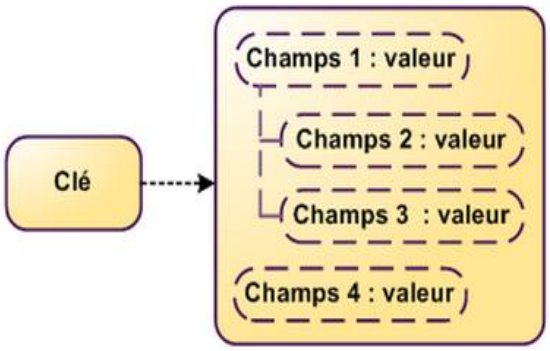
*"**À choisir pour** : choix intéressant pour de grands volumes de données et les besoins de bases distribuées, hautement disponibles et décentralisées, sur de multiples datacenters. Comme pour HBase, Cassandra n'est pas un choix intéressant pour des volumes de données moyens ou des systèmes non distribués."*

***Maintien de la cohérence :** la cohérence en écriture et en lecture sont paramétrables. Par exemple, on peut forcer une écriture, ou une lecture, à être validée sur plusieurs réplicas avant d'être considérée comme effectuée. À partir de Cassandra 1.1, l'ACIDité est garantie par ligne.*

***Mode de distribution :** décentralisé. Chaque nœud est indépendant et il n'y a pas besoin de serveur maître. Les connexions utilisateurs peuvent se faire sur n'importe quel nœud, qui se chargera de rediriger le client vers le nœud qui contient les données souhaitées. La répartition, inspirée de Dynamo d'Amazon, est réalisée par hachage consistant, ce qui évite de coûteux redéploiements de données en cas d'ajout de nœuds."* (Journaldunet, 2013a)

5.2.4.3 Les bases orientées documents

Les bases de données orientées documents se rapprochent du modèle clé-valeur. La valeur étant cette fois caractérisée par un document dans un format hiérarchique semi-structuré dont la structure est libre. Il est identifié par une clé unique. Ce type de modèle permet de stocker des données non-planes. Ce qui signifie des relations avec des jointures de type one to many. C'est particulièrement approprié pour les applications web.



Par exemple : un document qui contient les informations du vendeur et tous ses clients. Ce type de base pousse à la dénormalisation.

Figure 2.4 modèle orienté document (Figuière, 2010d)

Le texte, ci-dessous, explique bien le passage du modèle clé-valeur à celui de document avec les nouvelles fonctionnalités de ce dernier.

"Du clé-valeur au document

Pour passer de la représentation clé-valeur à la représentation en document, il s'agit principalement de rendre la base de données consciente de la structure de la valeur qu'elle stocke. Elle sera ainsi à même d'offrir un certain nombre de fonctionnalités qui ne peuvent être accessibles avec une base de données clé-valeur :

- *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"* (Figuière, 2010c)

Fort de son expérience dans le domaine, Michaël Figuière cite quelques cas d'utilisations particulièrement adaptés au modèle document.

- *Persistance de profils utilisateurs, de regroupement de contenus destinés à une même page, de données de sessions.*
- *Cache d'informations sous une forme structurée.*
- *Stockage de volumes très importants de données pour lesquelles la modélisation relationnelle aurait entraîné une limitation des possibilités de partitionnement et de réplication."* (Figuière, 2010c)

De plus, un des avantages conséquents de ce modèle, est ça capacité à gérer correctement le versioning. Le passage de Rudi Bruchez explique bien cette "atout" pour ce type de base.

"Un autre avantage des bases orientées documents est le versioning. Dans un moteur relationnel, si vous devez changer la structure des tables pour faire évoluer votre application, vous devrez changer votre schéma et faire des choses horribles comme ajouter un numéro de version dans le nom même de la table. Par exemple, si vous avez créé une table "TarifIntervention" en 2011, et que votre direction estime en 2012 que la gestion des tarifs doit changer, vous allez probablement créer une table "TarifIntervention2", parce que vous devez maintenir en parallèle l'ancienne et la nouvelle gestion des tarifs. La gestion d'une telle base de données deviendra de plus en plus complexe au fil du temps. Avec un moteur de bases de données orienté documents, vous pouvez simplement insérer des nouveaux documents comportant les modifications de structure, et maintenir dans le document un numéro de version, qui sera testé dans l'application cliente pour une prise en charge adaptée." (Bruchez, 2013)

En revanche, le désavantage de ce type de modèle, est la quantité importante de données dupliquées dans plusieurs documents.[31] Par conséquent, la gestion de la cohérence peut se relever très complexe.

Par exemple : plusieurs documents qui contiennent les informations de vendeurs et de leurs clients. Si une information de ces derniers change dans un document, la répercussion devra se faire sur tous les documents contenant un réplica de celle-ci. Autrement la cohérence n'est plus garantie. Si on peut vivre avec beaucoup de

[31] (Girolamo, 2013)

données non "consistent" c'est un bon choix. En outre, cela demandera un effort conséquent pour gérer cette problématique.

En résumé, les avantages de ce type de base sont:

- Gérer très bien le versioning
- Permettre de stocker des données non-planes
- Approprié pour les applications web
- Permet de structurer la valeur à stocker
- Ajouter, modifier, lire ou supprimer de seulement certains champs dans un document.
- Indexer de champs de documents permettant ainsi un accès rapide sans avoir recours uniquement à la clé
- Approprié comme cache d'informations sous une forme structurée.
- Stocker d'importants volumes de données distribuées et répliquées

Le désavantage est:

- La forte duplication des données et la difficulté à gérer la non-cohérence de ces dernières.

Les principales bases de données NoSQL orientées documents sont CouchDB et MongoDB.

5.2.4.3.1 Base de données NoSQL MongoDB

Figure 2.5 logo de mongoDB(MongoDB, 2014a)

MongoDB est une base de données NoSQL de type document. Elle monte bien en charge horizontalement et affiche de bonnes performances. Elle est très simple d'utilisation pour un client et intègre bien ses pilotes dans les langages de programmation. Elle s'adapte aux besoins, permettant ainsi une montée en charge horizontale efficace grâce à son sharding et une réplication automatique des données. Le passage, ci-dessous, explique bien les caractéristiques de MongoDB.

"À choisir pour : *tout type de besoin de stockage de documents, c'est-à-dire de données structurées : sérialisation d'objets, pages web, formulaires de saisie, informations d'applications. La représentation en documents permet d'utiliser MongoDB pour des structures proches de ce qui est stocké dans une base relationnelle, ou des structures hiérarchiques, avec une plus grande souplesse dans le schéma de données. Un système d'indexation secondaire permet d'effectuer des recherches dans les données et de récupérer des documents par d'autres points d'entrée que la clé.*

Maintien de la cohérence : *cohérence finale à travers les réplicas. Les écritures ont une option (write concern) pour indiquer qu'elles sont validées si elles ont écrit sur un certain nombre de réplicas ou sur une majorité (N/2+1). La durabilité est optionnelle à travers l'activation d'un journal (Write-ahead log). Une mise à jour est atomique par document, il n'y a pas de transaction multidocument possible.*

Mode de distribution : *centralisé. Autosharding et réplication automatique. Les métadonnées des shards sont centralisées sur un serveur de configuration, qui peut lui-même être répliqué pour fournir de la redondance. La connexion client se fait aussi sur un serveur maître, qui redirige la requête vers le bon serveur de shard. Ce serveur maître peut lui aussi être redondant."* (Journaldunet, 2013c)

5.2.4.3.2 Base de données NoSQL CouchDB

Figure 2.6 logo de CouchDB (CouchDB, 2014)

CouchDB est une base de données NoSQL de type document. Elle est parfaitement adaptée au web. Elle possède une si grande richesse fonctionnelle qu'elle peut devenir un vrai serveur applicatif. Elle est simple de mise en œuvre et ces fonctionnalités de réplication et d'audit des changements sont fiables. Par conséquent, on la choisira si on n'a des besoins poussé en richesse fonctionnelle pour une orientation application web. Le passage, ci-dessous, décrit bien les caractéristiques de CouchDB.

> ***"À choisir pour** : CouchDB est la base de données du Web. Son orientation documents et son interface REST excellent pour construire des projets web ou à destination des terminaux mobiles. CouchDB est à préférer pour les besoins où la richesse fonctionnelle l'emporte sur les grands volumes et la disponibilité. Pour les besoins de grandes performances, voir du côté de Couchbase Server, le successeur de CouchDB.*
>
> ***Types de données manipulées** : document JSON.*
>
> ***Maintien de la cohérence** : cohérence finale (eventual consistency). À la base, CouchDB n'étant pas distribué, la cohérence locale est gérée par un verrouillage optimiste : CouchDB maintient automatiquement un numéro de version sur chaque document. Pour modifier un document existant, il faut indiquer son dernier numéro de révision, sinon la mise à jour est refusée.*
>
> ***Mode de distribution** : pas de distribution native. Un système de réplication des données permet d'échanger les modifications d'une base avec un autre serveur, mais cela doit être géré manuellement. Pour une version nativement distribuée, voir Couchbase Server."* (Journaldunet, 2013b)

5.2.4.4 Les bases orientées graphes

Les bases de données NoSQL orientées graphes cherchent avant tout à répondre à des problèmes complexes voire impossibles pour des SGBDR. Elles offrent une réponse adéquate à un modèle de données **très volumineux et fortement connecté**. [32]

Par exemple: nous cherchons à connaître l'ensemble des personnes qui travaillent chez Google. Si nous nous plaçons dans le cas d'une modélisation relationnelle, on obtient l'exécution représentée ci-après, qui nécessite normalement 3 lookups d'index, optimisés en fonction des indexes et des clés étrangères déclarées. [33]

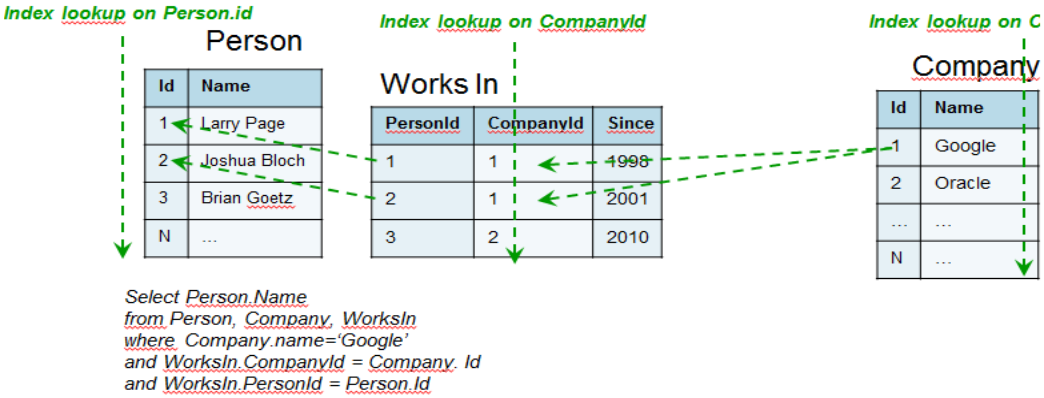


Figure 2.7 modélisation relationnel du cas.(Domenjoud, 2012)

En revanche, si on réalise cette modélisation dans un graphe, la requête en lecture n'aura besoin que d'un seul lookup d'index, puis un parcours par pointeurs physiques des liens (relation) dans le graphe. [34]

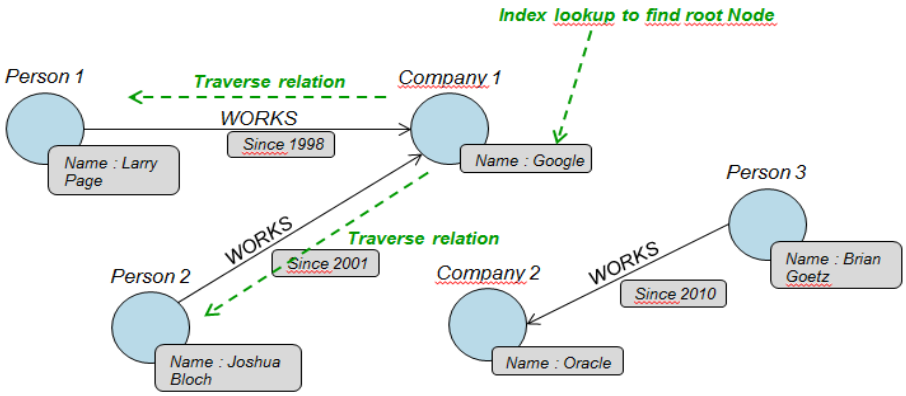


Figure 2.8 modélisation en graphe du cas (Domenjoud, 2012)

[32] (Figuière, 2010a)
[33] (Domenjoud, 2012)
[34] (Domenjoud, 2012)

Dans un cas similaire, le passage ci-dessous, explique que les bases NoSQL de type graphe sont plus performantes que leurs rivaux relationnels pour des volumes de données importants.

"La différence de performances sera d'autant plus importante lorsque la quantité de données stockées augmente, car hormis le lookup d'index pour trouver le(s) noeud(s) de départ du parcours de graphe, la requête se fera en temps à peu près constant dans un graphe, au contraire de la base de données relationnelle dans laquelle chaque parcours d'index pour récupérer une clé étrangère coûtera o(log2 N) si l'index est stocké dans un B-Tree (avec N le nombre d'enregistrements dans la table parcourue)."(Domenjoud, 2012)

De plus, la modélisation d'un problème avec un graphe est très intuitive et correspond à la réalité métier. Ce type de base est très flexible car elle n'impose pas de schéma de données figées comme c'est le cas avec un SGBDR qui respecte les formes normales. De surcroît, elle permet d'exécuter des algorithmes de graphes régulièrement utilisés dans un modèle de données fortement connecté.

Par exemple: connaître le plus court chemin entre deux nœuds ou ceux ayant une position centralisée dans le graphe.

Le principal cas d'utilisation des bases de données NoSQL de type graphe, c'est la modélisation des réseaux sociaux. Mais d'autres systèmes **fortement connectés** peuvent se relever pertinents dans le choix d'un modèle NoSQL de type graphe.

- *"la modélisation d'un ensemble de connaissances sur les personnes, et les organisations d'un secteur de marché ou d'un écosystème de manière plus générale.*
- *la représentation de données métier particulières telles que celles du cinéma (films, acteurs, réalisateurs, …), de l'édition (livres, auteurs, éditeur, …) ou encore la description de l'ensemble des pièces d'une machine industrielle et de la manière dont elles sont liées entre elles."* (Figuière, 2010a)

En résumé elles sont appropriées pour:

- pouvoir traiter efficacement un modèle de données fortement connectées
- pouvoir gérer facilement et avec flexibilité un modèle de données complexes
- avoir une grande performance pour les lectures locales, par parcours de graphe.[35]

Sur un marché restreint, Neo4J est une des principales bases de données NoSQL de type graphe.[36]

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

[35] (Domenjoud, 2012)
[36] (Figuière, 2010a)

5.2.5 Architecture avec maître

5.2.5.1 Réplication

La relation maître-esclave est très attachée aux systèmes transactionnels supportés par les SGBDR. D'abord on écrit sur le maître et celui-ci-réplique les données sur les esclaves de façon asynchrone. L'esclave prend la place du maître si celui-ci tombe. Le maître va s'assurer de la cohérence des données sur les esclaves. C'est lui le chef d'orchestre. Si l'intention est la forte cohérence des données, il sera préférable de lire les données sur le maître. Il est aussi possible de lire les données sur l'esclave (réplicas). Cependant, dans ce cas, il est possible que les données ne soient pas à jour.

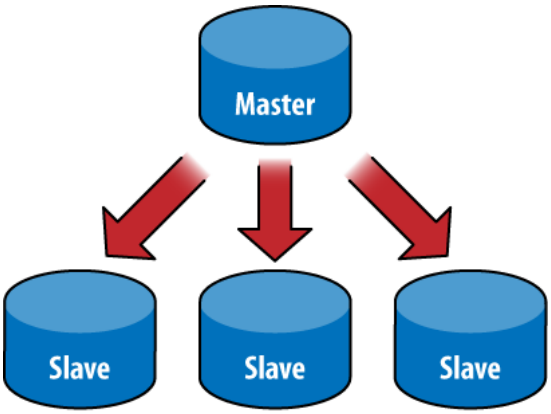


Figure 2.9 réplication master/slave (Dirolf, 2010)

En outre, lire sur les réplicas permet de répartir la charge des requêtes en lecture sur d'autres serveurs. Donc, si nous cherchons une haute disponibilité dans un temps de réponse acceptable, il faudra de préférence lire les répliquas.

Les bases de données NoSQL de ce type d'architecture, fonctionne sur ce même principe que les SGBDR. Redis, MongoDB et Neo4J fonctionnent en architecture maître-esclave. A noter que MongoDB peut aussi être utilisé en mode single (toute la base de données sur un seul serveur) ou réplicas set.[37]

[37] (Bruchez, 2013)

Le mode replicas set

Ce dernier mode fonctionne de façon sensiblement similaire au mode master slave. Le maître (nœud primaire) s'occupe de répliquer ses données sur les esclaves (les réplicas set). On lit les données sur le nœud primaire afin de s'assurer d'avoir une donnée cohérente. Cependant, si la haute disponibilité et la répartition de charge est de mise, il est possible de lire les données sur les réplicas set.

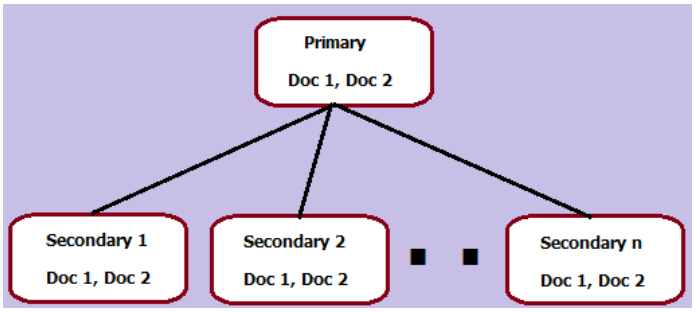


Figure 3.0 Réplicas set (Tiwari, 2013)

La différence avec le mode master/slave, c'est que le mode replicas set offre un mécanisme automatique de tolérance de pannes. Dans le cas du crash d'un nœud primaire, celui-ci sera automatiquement remplacé par un autre. Le nœud remplaçant deviendra primaire si tous les réplicas votent pour lui. Dans le cas contraire, le processus de vote continue jusqu'à élection d'un nœud.[38]

[38] (Ellingwood, 2013)

5.2.5.2 Sharding

Le sharding est une technique de distribution automatique des données sur les nœuds (les shards) à partir d'une clé. La répartition des données est gérée automatiquement par le système. Par conséquent, il n'est à priori pas nécessaire de le gérer manuellement[39]. Il est cependant essentiel de définir la clé (shared key) qui sera utilisée pour la distribution des données sur les shards. La clé sera utilisée par le système pour diviser les données à répartir sur les shards. Cette technique n'est pas née avec les bases NoSQL, les SGBDR utilisaient déjà celle-ci. Elle est utilisée par plusieurs bases NoSQL (c'est le cas notamment de MongoDB). C'est une surcouche basée sur le mode master/slave ou réplicas set. [40]

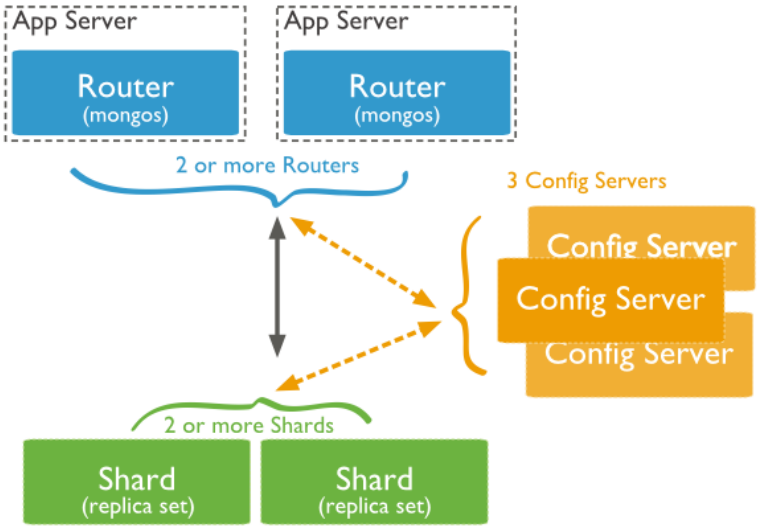


Figure 3.1 sharding dans MongoDB (MongoDB, 2014b)

Dans MongoDB les routeurs utilisent les meta-data des config server pour répartir les données dans les shards. Ces meta-data contiennent les règles de partitionnement des données.

[39] (Bruchez, 2013)
[40] (HEINRICH, 2012b)

5.2.6 Architecture sans maître

Dans ce type d'architecture, il n'y a pas de maître responsable qui endosse le rôle de chef d'orchestre de la réplication ou répartition des données sur les esclaves. Par conséquent, chaque nœud aura pour mission de répliquer et distribuer les données sur d'autres nœuds.

5.2.6.1 Répartition des données

Le hachage consistant permet d'attribuer des données à des nœuds par distribution automatique. Celle-ci se fait sur la base d'attribution d'une valeur aux données qui sont le résultat d'une fonction de hash. Ces dernières seront distribuées aux nœuds qui comprennent la rangée de valeurs acceptées. Le passage, ci-dessous, de Bruchez explique bien le hachage consistant.

"L'idée du hachage consistant est d'attribuer des rangées de valeurs pour chaque nœuds. Ces nœud ce voit attribuer une valeur de hachage, qui définit une rangée de valeur qui est hébergée. L'algorithme de hachage calcul un hash code à partir de la clé, trouve le nœud dont la valeur est immédiatement supérieur et y stocke la donnée. Un nœud contient donc toutes les clés inférieures à sa valeur de rangée et supérieur à la valeur du nœud précédent. Lorsqu'un nœud est ajouté à l'anneau il prend une valeur de rangée, et donc scinde une rangée existante. L'attribution des nœuds ne sera perturbée que pour une seule rangée. Donc un nombre peut important de données sera déplacé". (Bruchez, 2013)

Le système distribué est vu comme un anneau ou chaque nœud possède une place sur celui-ci par son hash.

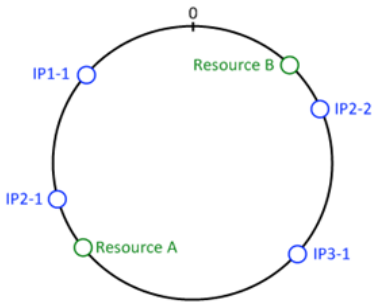


Figure 3.2 consistent hashing circle [41]

[41] (Mallassi, 2009)

Une question légitime se pose alors. Comment retrouve-t-on une donnée dans une architecture sans maître?

Ce système supporte le routage par table de hachage distribué. Cela implique que chaque nœud possède une table de hachage qui permettra de trouver un nœud par son numéro de hash. Elle est mise à jour par protocole de bavardage entre les nœuds.

Bortzmeyer explique bien dans le passage, ci-dessous, le fonctionnement du protocole de bavardage.

"Il existe une classe de protocoles réseaux nommés protocoles de bavardage (gossip protocol) qui résoud élegamment un problème classique dans les réseaux : informer tout le monde sans utiliser de mass media centralisé. Ils utilisent pour cela un principe simple : tout pair du réseau transmet l'information aux pairs qu'il connait (un sous-ensemble du total) et ceux-ci, à leur tour, transmettent aux pairs qu'ils connaissent, et ainsi de suite (on appelle parfois cette procédure l'indondation - flooding). Au bout d'un moment, la totalité du réseau est au courant, sans qu'on ait eu besoin d'un système centralisé." (Bortzmeyer, 2009)

Les nœuds distribuent leurs informations aléatoirement à d'autres nœuds dans un temps définit par une planification relâchée (cela permet de ne pas surcharger le réseau par la communication entre les nœuds).

5.2.7 Réplication

Pour ce qui est de la réplication sur un système sans maître, elle sera à paramétrer. Si on prend l'exemple de Cassandra, cette base réplique les données sur les nœuds grâce à une stratégie de réplication définie par l'utilisateur (par exemple, selon un paramètre de proximité géographique). Cassandra utilise une table de hachage (comme plus haut) dans laquelle elle stock les informations des nœuds sur lesquelles l'information est dupliquée.[42]

[42] (Bruchez, 2013)

5.2.8 Memcached

Dans ce contexte de forte consultation de données, il est important de citer cette technologie qui permet d'accélérer considérablement l'accès à l'information. Comme l'explique le passage ci-dessous, le principe est de mettre en mémoire dynamique les objets et données souvent consultés afin de diminuer leurs temps d'accès. A l'époque, déjà mis en place en complément des services basés sur un modèle relationnel, il est aujourd'hui toujours utilisé dans le monde NoSQL (c'est notamment le cas de Couchbase server qui l'utilise nativement).

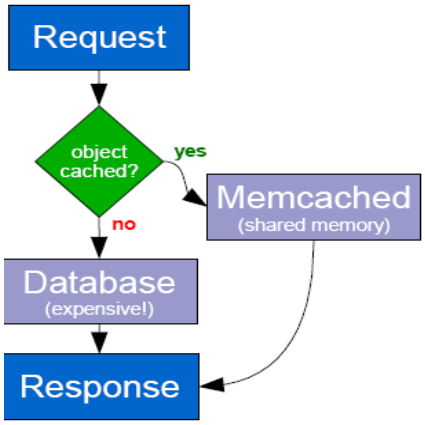


Figure 3.3 Algorithme de Memcached (cerberus, n.d.)

"Memcached est un système d'usage général servant à gérer la mémoire cache distribuée. Il est souvent utilisé pour augmenter la vitesse de réponse des sites web créés à partir de bases de données. Il gère les données et les objets en RAM de façon à réduire le nombre de fois qu'une même donnée stockée dans un périphérique externe est lue. Il tourne sous Unix, Windows et MacOS et est distribué selon les termes d'une licence libre dite permissive." (Wikipedia, 2014f)

5.2.9 WAL

Dans ce monde ou les données numériques prennent toujours plus d'importances, il est important de citer une autre technologie permettant de s'assurer qu'une base de données soit à jour au niveau transactionnel après un crash. Il s'agit du fichier journal de transaction nommé WAL. De plus, il permet de réduire considérablement le nombre d'écritures sur le disque. Il suffit d'écrire le journal WAL en base pour que toutes les transactions passent d'un coup. [43]

Ce système n'est pas apparu avec le NoSQL, il existait déjà avant. Cependant, plusieurs bases de données NoSQL l'intègrent (c'est le cas notamment de HBase et Oracle NoSQL database). [44]

[43] (PostGreSQL, n.d.)
[44] (Georg., 2012)

"Write-Ahead Logging (WAL) est une approche conventionnelle pour l'écriture d'un journal de transactions. Sa description détaillée peut être trouvée dans la plupart (si ce n'est tous) des livres sur le traitement transactionnel. Brièvement, le concept central des WAL est d'effectuer les changements des fichiers de données (où résident les tables et les index) uniquement après que ces changements ont été écrits dans un journal, c'est-à-dire quand l'enregistrement du journal décrivant les changements a été écrit vers le stockage permanent. Si nous suivons cette procédure, nous n'avons pas besoin d'écrire les pages de données vers le disque à chaque validation de transaction car nous savons que, dans l'éventualité d'une défaillance, nous serons capables de récupérer la base de données en utilisant le journal : chaque changement qui n'a pas été appliqué aux pages de données peut être ré-exécuté depuis les enregistrements du journal (ceci est une récupération roll-forward, aussi connue sous le nom de REDO)." (PostGreSQL, n.d.)

5.3 NoSQL sur mobile

Depuis l'avènement des smartphones, les applications sur mobile ont cru exponentiellement. Ces systèmes embarqués communicants, ont aujourd'hui des capacités mémoires et matérielles permettant d'y stocker une quantité "notable" de données.

Par exemple: un Samsung Galaxy S5 bénéficie nativement de 16 GB de mémoire fixe. En ajoutant une carte micro sd de 64 GB nous avons approximativement 80 GB de mémoire à disposition (il faudra cependant déduire ce qui est utilisé par l'OS et les autres applications du mobile).

Le mode off line des logiciels est une nécessité sur mobile. Il serait trop coûteux en communication d'imposer à l'utilisateur, des logiciels qui échangent beaucoup de données à travers le réseau. C'est pourquoi ces applications utilisent des bases de données sur le système embarqué.

Nous allons présenter ci-après quelques bases de données ou technologies permettant de stocker de l'information sur smartphone.

5.3.1 SQLite

Aujourd'hui, la plupart des applications mobiles natives utilisant une base de données, se reposent sur un SGBDR nommé SQLite. Elle est extrêmement légère (moins de 300KB) et n'utilise qu'un seul fichier pour le stockage des données[45]. Cette dernière propriété offre ainsi une simplicité accrue pour les backups, restauration et le déplacement de la base. Il s'agit ni plus ni moins qu'une base de données SQL transactionnelle en respect des règles ACID et des formes normales. Nous ne rappellerons pas ici les avantages et désavantages du modèle relationnel. Cependant, il est intéressant de constater que SQLite ne nécessite pas de serveur pour tourner. La base s'exécute dans le même processus que l'application qui utilise ses données (ce qui peut ralentir sensiblement cette dernière)[46]. SQLite est aussi adopté sur pc par des logiciels grands publics comme Firefox et Skype.

[45] (Wikipedia, 2014h)
[46] (Espiau, 2014)

Comme le mentionne le passage ci-dessous, cette solution pose plusieurs problèmes.

> - *lorsqu'un nombre important de clients accèdent à une même base, si un des clients commence la lecture d'une partie, la totalité de la base est bloquée en écriture pour les autres clients, et inversement si un des clients démarre l'écriture dans une partie, la totalité de la base est bloquée en lecture pour les autres clients. Ces derniers sont alors mis en attente durant ce laps de temps, ce qui peut être contre-performant ;*
> - *il est très compliqué de répartir la charge sur plusieurs machines : l'utilisation d'un système de fichiers sur le réseau engendre des latences et un trafic considérable, du fait que la base doit être rechargée chez le client à chaque ouverture. Ce problème est encore plus important avec des sites web où la base est rouverte pour chaque page demandée. De plus, il peut se produire des erreurs dans la synchronisation des verrous sur les fichiers qui pourraient autoriser l'écriture simultanée à deux clients différents sur une même partie de la base et ainsi aboutir à la corruption de la base de données ;*
> - *avec des bases de grande envergure, il est impossible de diviser la base en plusieurs parties ou fichiers dans le but de distribuer la charge sur plusieurs machines. Certains systèmes de fichiers, comme le FAT 32, ont une taille maximale de fichier qui pourrait rapidement limiter l'expansion d'une base.*(Wikipedia, 2014h)

C'est pourquoi SQLite sera préféré pour une application transactionnelle sur mobile qui ne nécessite pas de partager sa base avec d'autres logiciels.

5.3.2 Web Storage

Web Storage est une technologie permettant de conserver des données dans le navigateur. Elle rend possible le stockage de beaucoup plus d'informations que dans les cookies (quelques MB pour le Web Storage contre quelques KB pour un cookie). C'est un système de type clé-valeur ou la donnée (la valeur) est référencée par une clé.[47] Il n'y a aucun schéma fixe de données. Le développeur d'applications web est libre de stocker ce qu'il veut comme il veut. On constate, par conséquent, des ressemblances avec une base de données NoSQL de type clé/valeur. En revanche, cette technologie est spécialement orientée pour une application utilisant les standards du web. On pourra sauvegarder quelques informations à partir du navigateur, permettant ainsi de retrouver rapidement des données à la reconnexion. De plus, le travail après chargement des données, en mode hors-connexion sur l'application est rendu possible. Les principaux avantages de cette technologie pour les applications web sont:

- *"stocker rapidement des données en cache sans faire intervenir le serveur (par exemple via AJAX),*
- *augmenter la performance ressentie et faciliter les développements,*
- *se passer des cookies et du trafic HTTP supplémentaire qu'ils représentent (un cookie est envoyé à chaque requête effectuée sur un domaine),*
- *exploiter un espace alloué plus important que la limite imposée par les cookies (fixée à 4 Ko),*
- *retrouver des données immédiatement à la reconnexion ou les mémoriser pour éviter la perte s'il y a déconnexion."* (Alsacreations, 2012)

En revanche, bien qu'on puisse stocker plus d'informations que dans un cookie, cette technologie reste limitée en termes de stockage.

"Par défaut, Internet Explorer alloue 10 Mo par espace de stockage, et les autres navigateurs (Firefox, Chrome, Opera, Safari) 5 Mo par domaine." (Alsacreations, 2012)

De plus, on ne peut stocker que des chaînes de caractères. Pour stocker un objet JavaScript plus complexe, il faudra utiliser le format JSON. Deux primitives intégrées à la plupart des navigateurs permettent de manipuler ces objets.

[47] (Alsacreations, 2012)

"JSON.stringify() qui prend en paramètre un objet et renvoie une chaîne de texte linéarisée, et son inverse JSON.parse() qui reforme un objet à partir de la chaîne linéarisée." (Alsacreations, 2012)

Par conséquent, on comprend ainsi pourquoi cette technologie est aussi utilisée sur mobile pour une application web. Tout le monde n'a pas forcément beaucoup de mémoire sur son smartphone (d'où l'utilisation d'une application web). En outre, il est souhaitable d'avoir la possibilité de stocker des informations sur le système embarqué afin de pouvoir travailler en mode hors connexion ou éviter d'avoir à recharger des données lors d'une reconnexion (système de cache).

5.3.3 IndexedDB

IndexedDB est une base de données NoSQL qui stocke des données (sous forme d'objets) dans le navigateur du client. Aucune installation n'est nécessaire. Les navigateurs qui le supportent proposent une API IndexedDB permettant de l'utiliser. C'est une évolution du WebStorage (les avantages de ce dernier y sont repris). Chaque objet (la valeur) est identifié par une clé. Ils sont insérés dans IndexedDB en utilisant la syntaxe JSON.[48] On entrevoit dans ce principe une ressemblance avec les bases de données orientées documents comme MongoDB ou CouchDB. Voici quelques-unes de ses caractéristiques:

- Toutes opérations sur les données se fait au travers d'une transaction ACID.
- Un schéma doit être définit avant de pouvoir stocker des données et réaliser des transactions sur ces dernières. On crée des "object store" dans lequel on stockera des objets de même type. **Par exemple:** des objets de type "Person".
- Une transaction n'est permise que sur le domaine depuis lequel elle est appelée. **Par exemple:** depuis la page http://www.google.ch on ne peut réaliser des transactions que sur les données appartenant à ce domaine.
- Pas de limite de taille sur les données stockées dans le navigateur[49]
- Seuls les derniers navigateurs supportent IndexedDB [50]Les passages ci-dessous, expliquent bien à quels besoins IndexedDB cherchent à répondre.

[48] (Julienw, 2011)
[49] (Julienw, 2011)

"IndexedDB est une API de stockage côté client qui permet de gérer des quantités importantes de données structurées et des recherches de haute performance sur ces données en utilisant des index. Alors que DOM Storage (Ndlr. WebStorage) est utile pour stocker de petites quantités de données, il est moins utile pour stocker de grandes quantités de données structurées. IndexedDB répond à ce besoin. "(Caudralys, 2014)

"Ses fonctions de recherche avancées permettent de créer des applications qui fonctionnent tant connecté que déconnecté. " (Julienw, 2011)

En résumé, IndexedDB et WebStorage sont tous deux des outils permettant de stocker de l'information dans un navigateur. Ils répondent cependant à différents besoins.

Disposant de toujours plus de mémoire sur mobile, les applications web ont tout intérêt à utiliser IndexedDB pour stocker des quantités d'informations supérieures à la limite du WebStorage dans les navigateurs. Le mode hors ligne des logiciels web et la mise en cache des données sont ainsi étendues.

[50] (Mohan, 2014)

5.3.4 Couchbase lite

Couchbase lite est la version mobile de la base de données NoSQL orientée document Couchbase server. Elle étend sa haute disponibilité des données cette fois sur mobile grâce notamment au mode hors-ligne des applications garanti par sa présence native sur le smartphone. Le coté sans schéma de la base de données, la structuration des données sous forme JSON et le support dans les plus grands langages de librairies permettant leurs manipulations dans la base n'est pas sans déplaire au développeur de logiciel. De plus, étant une base NoSQL, Couchbase lite permet de stocker aussi des données semi-structurées et non-structurées sur mobile.[51] Comme mentionné plus haut nous utilisons de plus en plus de données non-structurées. Ce phénomène est bien entendu équivalent sur smartphone car nous avons accès maintenant aux mêmes applications que sur pc.

CouchBase Sync Gateway est un outil fournit par Couchbase lite pour synchroniser avec le serveur la base de données mobile et vice et versa. Il permet de répliquer ou partager les données avec d'autres systèmes mobiles. La collaboration sur des flux de données entre plusieurs terminaux est supportée par cet outil de synchronisation. Le passage ci-dessous, décrit de façon succincte bien Couchbase lite et l'idée dernière cette déclinaison mobile. [52]

"Disposant d'API équivalentes à la version serveur, quant à elle plutôt destinée au traitement d'informations en masse, Couchbase Lite se présente sous la forme d'un serveur de données ultraléger doté d'une passerelle (Couchbase Sync Gateway) pour gérer les synchronisations avec celui d'autres terminaux.

"L'idée est de proposer une solution performante pour prendre en charge les sessions de chat et les flux de collaboration entre plusieurs terminaux simultanément, en orchestrant des communications temps réel", commente-t-on chez Couchbase." (Antoine Crochet- Damais, 2013)

Ainsi, Couchbase lite couvre particulièrement les besoins d'échanges, de partage et de la collaboration sur des données ou flux de données structurées, semi-structurées ou non-structurées entre systèmes embarqués communicants. De surcroit, comme mentionné plus haut elle propose de base ce que peut offrir un système de gestion de données NoSQL de type document.

[51] (CouchBase, 2014b)
[52] (CouchBase, 2014a)

5.4 Problème de "eventual consistency"

Au vue de ce qui précède, on pourrait ainsi penser que le NoSQL présente que des avantages. Cependant, l'abandon quasi systématique de la forte cohérence des données au bénéfice de la performance et de la haute disponibilité est un désavantage de cette technologie. S'assurer d'avoir une donnée cohérente dans un système redondant a souvent un coup important. Il faudra attendre que tous les réplicas contenant les données identiques sur lesquelles opérés soient à jours avant une transaction sur le système. Par conséquent, dans la plupart des cas on accepte que la donnée soit éventuellement cohérente. Ceci bien sûr ne peut être appliqué que si on accepte de travailler avec des données qui ne soient pas toujours à jours.

Par exemple : un réseau social n'a pas besoin d'une forte cohérence des données. Il n'est pas vital que tous les amis de Dupont voient les derniers tweets au même moment.

Ce système adopté par la plupart des technologies NoSQL porte le nom de « eventual consistency ». En outre, c'est pour certain un frein à l'adoption de technologie de type NoSQL. On imagine aisément pourquoi les milieux bancaires ne sont pas séduits par cela. Consulter le cours d'un taux de change pas qui n'est pas à jour pour une transaction valant des millions est tout simplement inacceptable.

De plus, les développeurs d'applications doivent répondre à plusieurs questions importantes avec l'adoption d'une éventuelle cohérence des données.

- Quel sera l'effet sur l'application si la base de données retourne arbitrairement une valeur pas à jour ?
- Quel sera l'effet sur l'application si la base de données détecte des modifications exécutées dans un mauvais ordre ?
- Quel sera l'effet sur l'application si un autre utilisateur modifie les données au même moment que je suis en train de les lire ?
- Quel sera l'effet pour un autre utilisateur qui cherche à lire une donnée dans la base si je modifie celle-ci ?

Par conséquent, ils passent beaucoup de temps pour gérer correctement les effets de l'éventuelle cohérence des données.[53]

[53] (Rosenthal, 2013)

5.5 Critères

Dans cette partie nous allons extraire les critères de choix qui vont conduire à l'adoption du NoSQL comme système de base de données. On l'utilise principalement dans un contexte d'informations massives et/ou en croissances exponentielles sur un système distribué. Comme vu dans les parties relatives au Big Data et au NoSQL, cette technologie de manière générale couvre les besoins suivant pour un important volume de données. On la choisira dans le cas où…:

- on cherche à gérer une montée en charge horizontale d'importants volumes de données en croissances exponentielles.
- on veut une répartition et réplication des données efficaces et automatiques sur les nœuds sans intervention humaine (ou presque).
- on veut paralléliser automatiquement le traitement des requêtes sans intervention humaine (ou presque).
- on veut disposer d'un schéma de données flexible.
- on veut stocker, traiter et lire des données principalement non-structurées et semi-structurées.
- la performance des requêtes en lecture est fondamentale.
- la haute disponibilité des données est une priorité.
- on cherche à faire de l'analyse prédictive sur des flux de données continus.
- la performance des requêtes en lecture et la haute disponibilité des données peuvent être adoptées au détriment de la forte cohérence de ces dernières.
- on cherche à réduire les coûts en investissant dans du matériel moyen de gamme.

Plus spécifiquement, les bases de données NoSQL de type "entrepôt clé-valeur" couvrent les besoins suivant. On les choisira dans le cas où…:

- on cherche des opérations de lecture et écriture les plus performantes.
- on veut réaliser un cache mémoire

Pour des exemples d'implémentations, se référer aux chapitres relatifs aux bases Redis et Riak.

Plus spécifiquement, les bases de données NoSQL de type "orientées colonnes" couvrent les besoins suivant. On les choisira dans le cas où..:

- on veut une performance en écriture sur un système de gestion de données se rapprochant d'un modèle relationnel hautement dénormalisé.
- on ne veut plus avoir à traiter de valeurs "NULL" dans un enregistrement.
- on cherche un système de requête minimaliste performant.
- on cherche à stocker et requêter des relations de type one to many au sein d'une même famille de colonnes (sous-tables).

Pour des exemples d'implémentations, se référer aux chapitres relatifs aux bases HBase et Cassandra.

Plus spécifiquement, les bases de données NoSQL de type "orientées documents" couvrent les besoins suivant. On les choisira dans le cas où:

- on cherche un système de gestion de données adapté au versioning
- on cherche un système de gestion de données adapté aux relations de type one to many.
- on veut un système de gestion de données adapté pour une application web.
- on veut structurer la valeur à stocker
- on veut pouvoir ajouter, modifier, lire ou supprimer individuellement des champs dans un document.
- on veut un cache d'informations sous une forme structurée.

Pour des exemples d'implémentations, se référer aux chapitres relatifs aux bases MongoDB et CouchDB.

Plus spécifiquement, les bases de données NoSQL de type "graphes" (p.ex. Neo4J) couvrent les besoins suivant. On les choisira dans le cas où…:

- on veut traiter efficacement un modèle de donnée fortement connecté
- on veut pouvoir gérer facilement et avec flexibilité un modèle de données complexe
- on veut avoir une grande performance pour les lectures locales, par parcours de graphe.

Pour ce qui est du NoSQL sur mobile, on l'utilisera principalement pour :

- disposer d'un mode off-line sur l'application du smartphone.

Si on se réfère aux applications sur smartphone, on utilisera :

- la technologie Web Storage ou la base de données NoSQL IndexedDB pour des applications web sur mobile qui nécessitent la mise en cache d'un volume de données notables (bien supérieur aux cookies).

Si on recherche les avantages d'une base de données orientée document sur mobile on choisira:

- CouchBase lite dans les cas où on a un besoin d'échange, où de partage et de collaboration sur des données ou un flux de données structurées, semi-structurées ou non-structurées entre systèmes embarqués communicants.

6. Le NewSQL

6.1 Le future du SQL

Dans les parties précédentes, nous avons parlé des nouveaux besoins qui sont survenus depuis l'émergence du Big Data et des bases de données NoSQL qui peuvent être dans certain cas une réponse pertinente. Cependant, il apparait que cette technologie sacrifie la plupart du temps 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[54]. Certaines entreprises ne peuvent simplement pas se passer de transactions ACID (p.ex. banque, fiduciaire etc…). De plus, le développeur sera conduit à "jongler" avec l'éventuelle cohérence des données. Comme nous l'avons vu plus haut, cela oblige celui-ci à se poser plusieurs questions qui vont impliquer un important travail de réflexion pour pouvoir gérer ce problème. De surcroît, chaque système de gestion de base de données NoSQL vient avec son propre langage de requête. On déplore ainsi un manque de standardisation des interfaces entre les applications.[55]

Durant l'année deux-milles onze, un groupe de recherche nommé "451 Research" a analysé avec brio le problème évoqué ci-dessus et défini le nom "NewSQL" pour une base de données regroupant les avantages du NoSQL et des SGBDR[56].

Dès lors, plusieurs implémentations de ce nouveau type d'architecture sont apparues très récemment.

Figure 3.4 le NewSQL et ses implémentations (Eakwattanaphan, 2014)

[54] (Venkatesh, 2012)
[55] (Leblal, 2011)
[56] (Group, 2011)

6.2 Caractéristiques du NewSQL

Le NewSQL est une architecture qui reprend les avantages du NoSQL et comble son principal désavantage. Il palie à l'éventuel cohérence des données de ce dernier grâce au support de transaction ACID via un langage unique de requétage le SQL.

Il est aussi jeune qu'il est plein de promesses. Voici ci-dessous quelques-unes de ces caractéristiques:

- Le SQL comme langage commun de requétage
- Transaction ACID
- Un mécanisme qui évite la pause de verrous lors d'opérations concurrentes de lecture avec les opérations d'écritures. La lecture en temps réel en est ainsi facilitée.
- Une architecture qui a de meilleures performances par nœud que les solutions classiques de type SGBDR.
- Scalabilité horizontale, architecture sans maître et capable de tourner sur un nombre important de nœuds sans souffrir de goulot d'étranglement. [57]
- Architecture distribuée.
- Base de données en mémoire.[58]

Au vu de ce qui précède, on peut raisonnablement "penser" que ce type d'architecture sera celle adoptée demain en entreprise.

En revanche, nous n'avons pas le recul suffisant pour aller plus loin dans cette étude car le NewSQL est extrêmement jeune et n'a pas encore pu suffisamment faire ses preuves et être sujet à étude. Par conséquent, les entreprises sont encore réticentes à l'adoption de cette toute nouvelle architecture. [59] [60]

[57] (Venkatesh, 2012)
[58] (Moniruzzaman, n.d.)
[59] (Henschen, 2014)
[60] (Moniruzzaman, n.d.)

7. Stratégies de choix

Nous avons vu dans les parties précédentes, ce que sont ces technologies, à quels besoins elles répondent et leurs critères de choix. Dans ce chapitre nous allons effectuer la synthèse de ces derniers.

Les principaux éléments permettant d'opposer et choisir ces technologies sont la forte cohérence des données et la scalabilité horizontale.

Le graphique ci-dessous, résume bien ces observations.

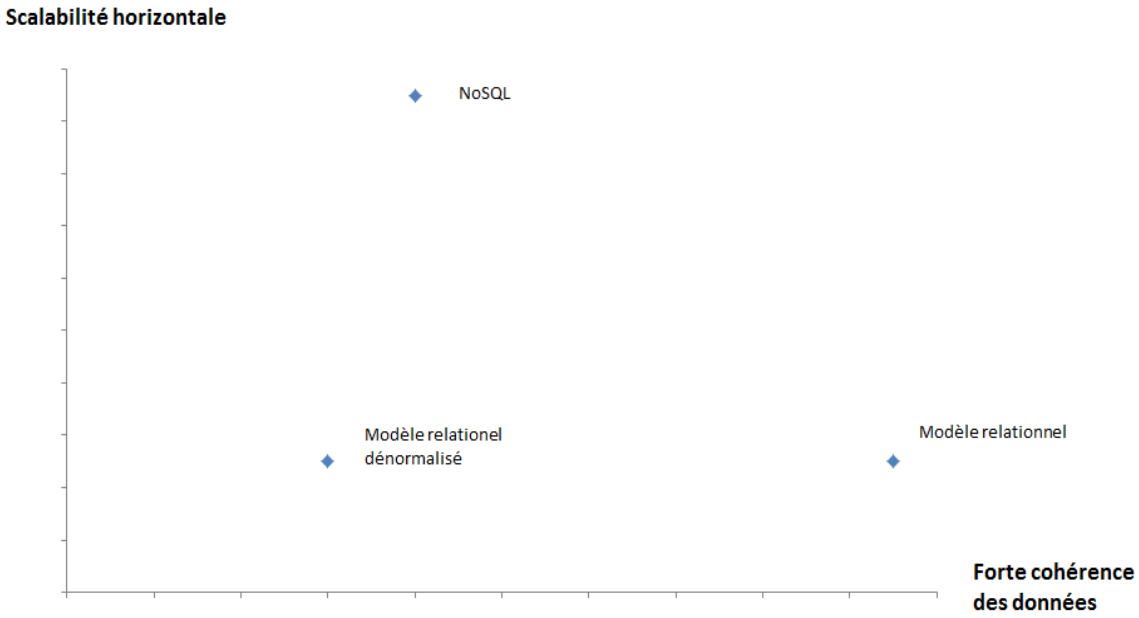


Figure 3.5 notre graphique comparatif

Au vu de ce qui est montré ci-dessus, on peut conclure qu'on choisira le modèle relationnel si on n'a besoin de données fortement cohérentes et si notre système ne nécessite (ra) pas une forte montée en charge horizontale. De plus, on optera pour une dénormalisation de celui-ci pour de meilleures performances avec les requêtes en lecture au détriment de la forte cohérence des données et de la montée en charge horizontale. Le NoSQL sera adopté pour une forte scalabilité horizontale et si l'éventuelle cohérence des données est suffisante pour le système à supporter. Cette dernière explique le positionnement plus à droite sur le graphique par rapport au modèle relationnel dénormalisé qui lui ne la garantit pas.

Au vue de son extrême jeunesse et du manque de retour suffisant des entreprises l'ayant implémenté, le NewSQL n'est pas situé dans ce graphique. Cependant, si nous nous référons à ses "promesses" nous pourrions le placer tout en haut à droite.

Pour plus de détails, nous avons synthétisé l'ensemble des critères de choix d'adoption pour ces technologies dans le tableau comparatif ci-dessous.

	Modèle relationnel	Modèle relationnel dénormalisé	NoSQL
Transactions ACID	X	-	-
Eventuelle cohérence des données	X	X	X
Forte cohérence des données	X	-	-
Redondance des données	-	X	X
Optimiser l'espace disque	X	-	-
Langage unique permettant de manipuler et définir des droits d'accès sur les données	X	X	-
Langage d'interrogation de la base de données facile d'accès pour un non-informaticien	X	X	-
Performance des requêtes en lecture sur un important volume de données	-	X	X
Analyse de données volumineuses	-	X	X

Base de données multidimensionnelle	-	X	X
Réduction de la complexité des requêtes	-	X	X
Scalabilité horizontale sur un important volume de données.	-	-	X
Répartition et réplication des données sans interventions humaines (ou presque) sur un important volume de données	-	-	X
Paralléliser automatiquement le traitement des requêtes sans interventions humaines (ou presque)	-	-	X
Disposer d'un schéma de données flexible.	-	-	X
Stocker, traiter et lire des données principalement non-structurées et semi-structurées.	-	-	X
Pour un important volume de données leurs hautes disponibilités sont une priorité	-	X	X
Analyse prédictive sur des flux de données continus.	-	-	X

Pouvoir réduire les coups en investissant dans du matériel moyen de gamme	-	-	X
Représentation des données sous formes de tables à deux dimensions	X	X	-
Standard de modèle de données le plus utilisé.	X	-	-

A noter que "**X**" = Bon ou très bon "**-**" = faible ou inexistant

De par leur caractère très spécifique, les différents types de base de données NoSQL ainsi que celles sur mobile ne sont pas représentées dans ce tableau.

Pour plus de détails sur les critères de choix, veuillez-vous référer aux chapitres relatifs aux technologies abordées dans ce travail de Bachelor.

Il est important de préciser que ces dernières peuvent et devront parfois être utilisé de concert[61].

Par exemple:

Utiliser un SGBDR pour les données ayant besoins d'une forte cohérence et mettre en place une base de données NoSQL pour gérer la forte montée en charge horizontale pour les autres qui n'en nécessite pas. Cette combinaison peut s'avérer être un choix judicieux.

[61] (Aubry, n.d.)

8. Conclusion

Dans ce travail nous avons extrait l'ensemble des critères permettant de choisir ces technologies. Synthétisés dans la partie "stratégie de choix", ces derniers vont permettre aux responsables d'un projet d'informatisation de répondre aux questions suivantes:

- Est-ce que l'architecture que j'utilise est la plus adaptée ?
- Est-ce que je dois me tourner vers d'autres architectures?
- Comment choisir l'architecture selon le projet?

Comme mentionné plus haut, ces technologies répondent malheureusement actuellement encore à des besoins antagonistes. Chacune d'elle ne peut garantir à la fois la forte cohérence des données, la haute disponibilité, une montée en charge horizontale et une performance des requêtes en lecture sur des volumes très importants de données. Pour y parvenir il faudra les utiliser de concert.

Plein de promesses le NewSQL nous fait miroiter une solution miracle permettant de combiner les avantages du modèle relationnel et du NoSQL. Ainsi, les besoins cités plus haut ne s'opposeront plus. Il faudra s'armer de patience afin de constater si cette nouvelle technologie y répond correctement.

Ce travail m'a beaucoup apporté, il m'a offert une vue d'ensemble des bases de données NoSQL. En outre, cela m'a permis de le comparer au modèle relationnel normalisé ou non et d'en retirer des critères de choix pour ces technologies. Cette connaissance nouvellement acquise me sera certainement précieuse dans ce monde ou l'information est devenue ubiquitaire (omniprésente). Le stockage des données de cette dernière est devenu une priorité pour l'essentiel des entreprises d'aujourd'hui et certainement de demain[62]. Les centres de données fleurissent partout dans le monde et les systèmes de gestion de base de données qui vont avec aussi[63].

[62] (StorageNewsletter, n.d.)
[63] (Coquio, n.d.)

Bibliographie

Alsacreations. (2012). Stockage des données locales : Web Storage. *2012*. Retrieved from http://www.alsacreations.com/article/lire/1402-web-storage-localstorage-sessionstorage.html

Antoine Crochet- Damais. (2013). NoSQL : Couchbase arrive sur iOS et Android. *2013*. Retrieved from http://www.journaldunet.com/solutions/dsi/nosql-couchbase-sur-mobile-0913.shtml

Apach. (2014). Welcome to Apache HBase™. *2014*. Retrieved from http://hbase.apache.org/

Aubry, Y. (n.d.). Relationnel vs. NoSQL : nous devons faire évoluer les modèles de bases de données traditionnels. *2014*. Retrieved from http://www.programmez.com/avis-experts/relationnel-vs-nosql-nous-devons-faire-evoluer-les-modeles-de-bases-de-donnees-traditionnels-20381

BARON, M. (2014). Tutoriel d'introduction à Apache Hadoop. *2014*. Retrieved from http://mbaron.developpez.com/tutoriels/bigdata/hadoop/introduction-hdfs-map-reduce/

Becquet, T. (2011). HANA – Ce que vous n'avez pas osé demander sur les bases de données en colonnes : ça marche comment ? *2011*. Retrieved from http://blog.censio.fr/sap-hana/hana-base-donnees-colonne-definition/

Bellamy, A. (2013). LG : four, lave-linge et réfrigérateur NFC, pilotés par smartphone. *2013*. Retrieved from http://www.lesnumeriques.com/lg-four-lave-linge-refrigerateur-nfc-pilotes-par-smartphone-n31223.html

Borderie, X. (n.d.). Expliquez-moi... Les propriétés ACID d'une base de données. *2006*. Retrieved from http://www.journaldunet.com/developpeur/tutoriel/theo/060615-theo-db-acid.shtml

Bortzmeyer, S. (2009). Les protocoles réseaux de bavardage. *2009*. Retrieved from http://www.bortzmeyer.org/gossip-protocol.html

Brice Davoleau. (n.d.). No Title. Retrieved November 16, 2014, from http://www.cogoobi.com/blog/2009/01/26/caracteristique-de-la-bi-"denormalisons-les-donnees"/

Bruchez, R. (2013). *Les bases de données NoSQL Comprendre et mettre en oeuvre*. (Eyrolles, Ed.).

Cassandra, A. (2014). Welcome to Apache Cassandra. *2014*. Retrieved from http://cassandra.apache.org/

Caudralys. (2014). IndexedDB. *2014*. Retrieved from https://developer.mozilla.org/fr/docs/IndexedDB

cerberus. (n.d.). Using Memcached as an object cache. Retrieved from http://cerberusweb.com/book/latest/admin_guide/performance/memcache.html

Chaghal, A. (2011). NoSQL : 5 minutes pour comprendre. *2011*. Retrieved from http://blog.neoxia.com/nosql-5-minutes-pour-comprendre/

Charrière, J. (n.d.). Principes de base. Retrieved from http://csud.educanet2.ch/3oc-info/2_Bases-donnees/1_Introduction/page2.html

CHEMINAT, J. (2013). Big data: 150 millions pour MongoDB. Retrieved from http://www.ictjournal.ch/fr-CH/News/2013/10/08/Big-data-150-millions-pour-MongoDB.aspx

Clarke, G. (n.d.). 12 simple rules: How Ted Codd transformed the humble database. *2013*. Retrieved from http://www.theregister.co.uk/Print/2013/08/19/ted_codd_90_relational_daddy/

Coquio, F. (n.d.). Les data centers, pierres angulaires de l'économie numérique. *2013*. Retrieved from http://pro.01net.com/editorial/605472/les-data-centers-pierres-angulaires-de-l-economie-numerique/

CouchBase. (2014a). Couchbase Server Architecture and Capabilities. *2014*. Retrieved from http://www.couchbase.com/nosql-databases/about-couchbase-server#DataMobility

CouchBase. (2014b). couchbase-mobile. *2014*. Retrieved from http://www.couchbase.com/nosql-databases/couchbase-mobile

CouchBase. (2014c). Pourquoi NoSQL? *2014*. Retrieved from http://www.couchbase.com/fr/why-nosql/nosql-database

CouchDB. (2014). A Database for the Web. *2014*. Retrieved from http://couchdb.apache.org/

Damais, A. C.-. (2012). Hadoop en 5 questions. *2012*. Retrieved from http://www.journaldunet.com/solutions/systemes-reseaux/definition-d-hadoop.shtml

Diforti, L. (2013). Qu'est-ce que le Big Data ? *Management de la performance*. Retrieved from http://www.redsen-consulting.com/2013/06/big-data/

Dirolf, M. (2010). MongoDB: The Definitive Guide. *2010*. Retrieved from http://librairie.immateriel.fr/fr/read_book/9781449381561/ch09

Domenjoud, M. (2012). Bases de données graphes : un tour d'horizon. *2012*. Retrieved from http://blog.octo.com/bases-de-donnees-graphes-un-tour-dhorizon/

DORY, T. (2012). Quand et pourquoi utiliser une base de données NoSQL ? Retrieved from http://www.marginweb.com/blog/20121110/quand-et-pourquoi-utiliser-une-base-de-donnees-nosql

DURIS, M. E. (n.d.). Les formes normales. *2012*. Retrieved from http://igm.univ-mlv.fr/~dr/XPOSE2011/BDD/fn.html

Eakwattanaphan, A. (2014). what-is-newsql. *2014*. Retrieved from http://www.somkiat.cc/what-is-newsql/

Ellingwood, J. (2013). How To Implement Replication Sets in MongoDB on an Ubuntu VPS. *2013*. Retrieved from https://www.digitalocean.com/community/tutorials/how-to-implement-replication-sets-in-mongodb-on-an-ubuntu-vps

Erb, B. (n.d.). Concurrent Programming for Scalable Web Architectures. Retrieved from http://berb.github.io/diploma-thesis/original/061_challenge.html

Espiau, F. (2014). LES BASES DE DONNÉES. *2014*. Retrieved from http://openclassrooms.com/courses/creez-des-applications-pour-android/les-bases-de-donnees-5

Figuière, M. (2010a). NoSQL Europe : Bases de données graphe et Neo4j. *2010*. Retrieved from http://blog.xebia.fr/2010/05/03/nosql-europe-bases-de-donnees-graphe-et-neo4j/

Figuière, M. (2010b). NoSQL Europe : Bases de données orientées colonnes et Cassandra. *2010*. Retrieved from http://blog.xebia.fr/2010/05/04/nosql-europe-bases-de-donnees-orientees-colonnes-et-cassandra/

Figuière, M. (2010c). NoSQL Europe : Bases de données orientées documents et MongoDB. *2010*. Retrieved from http://blog.xebia.fr/2010/04/30/nosql-europe-bases-de-donnees-orientees-documents-et-mongodb/

Figuière, M. (2010d). NoSQL Europe : Tour d'horizon des bases de données NoSQL. *2010*. Retrieved from http://blog.xebia.fr/2010/04/21/nosql-europe-tour-dhorizon-des-bases-de-donnees-nosql/

Gasnier, L. (2013). Aux origines du Big data…. *2013*. Retrieved from http://www.solucominsight.fr/2013/08/auw-origines-du-big-data/

Georg., L. (2012). NoSQL and Cloud Databases Community. *2012*. Retrieved from http://www.toadworld.com/platforms/nosql/w/wiki/357.hbase-write-ahead-log.aspx

Gerald. (2011). Gerald's Blog. *2011*. Retrieved from http://www.croes.org/gerald/blog/qu-est-ce-que-rest/447/

Girolamo, A. P. (2013). NoSQL Etat de l'art et benchmark. *2013*. Retrieved from http://doc.rero.ch/record/209288/files/Travail_de_Bachelor_-_NoSql_Etat_de_l_art_et_Benchmark.pdf

Google. (n.d.). Hadoop on Google Cloud Platform. *2014*. Retrieved from https://cloud.google.com/hadoop/

Grim, Y. (n.d.). OLAP, les fondamentaux. *Publié le 7 mai 2008*. Retrieved from http://grim.developpez.com/articles/concepts/olap/

Group, 451. (2011). NoSQL, NewSQL and Beyond: The drivers and use cases for database alternatives. *2011*. Retrieved from https://451research.com/report-long?icid=1651

Guillaume Sarlat (inspecteur des finances), S. D. V. et O. M. (2014). Le Big Data, un virage technologique à ne pas rater. Retrieved from

http://www.lemonde.fr/idees/article/2014/10/02/le-big-data-un-virage-technologique-a-ne-pas-rater_4499371_3232.html

HEINRICH, L. (2012a). Architecture NoSQL et réponse au Théorème CAP. *2012*. Retrieved from http://doc.rero.ch/record/31235/files/TDIG_74.pdf?version=1

HEINRICH, L. (2012b). Architecture NoSQL et réponse au Théorème CAP. *2012*. Retrieved from https://doc.rero.ch/record/31235/files/TDIG_74.pdf

Henschen, D. (2014). 16 NoSQL, NewSQL Databases To Watch. *2014*. Retrieved from http://www.informationweek.com/big-data/big-data-analytics/16-nosql-newsql-databases-to-watch/d/d-id/1269559

IBM. (n.d.). What is Hadoop? *2014*. Retrieved from http://www-01.ibm.com/software/data/infosphere/hadoop/

Jean-François PILLOU. (n.d.). Le modèle relationnel. *2015*. Retrieved from http://www.commentcamarche.net/contents/1013-le-modele-relationnel

Journaldunet. (2013a). Cassandra : une base NoSQL adaptée à une architecture décentralisée. *2013*. Retrieved from http://www.journaldunet.com/developpeur/outils/comparatif-des-bases-nosql/cassandra.shtml

Journaldunet. (2013b). CouchDB : une base orientée documents bien adaptée au web. *2013*. Retrieved from http://www.journaldunet.com/developpeur/outils/comparatif-des-bases-nosql/couchdb.shtml

Journaldunet. (2013c). MongoDB : une gestion intelligente de la montée en charge. *2013*. Retrieved from http://www.journaldunet.com/developpeur/outils/comparatif-des-bases-nosql/mongodb.shtml

Journaldunet. (2013d). Riak : un moteur NoSQL distribué pour une latence faible. *2013*. Retrieved from http://www.journaldunet.com/developpeur/outils/comparatif-des-bases-nosql/riak.shtml

Julienw. (2011). Les concepts basiques d'IndexedDB. *2011*. Retrieved from https://developer.mozilla.org/fr/docs/IndexedDB/Basic_Concepts_Behind_IndexedDB

Larousse, N. (2006). *Création de base de données*. Retrieved from http://books.google.ch/books?id=_I0kSc5idxAC&dq=redondance+incohérence+des+données&hl=fr&source=gbs_navlinks_s

Leblal, S. (2011). NewSQL pour combiner le meilleur de SQL et NoSQL. *2011*. Retrieved from http://www.lemondeinformatique.fr/actualites/lire-newsql-pour-combiner-le-meilleur-de-sql-et-nosql-34475-page-2.html

Lévy-Abégnoli, T. (2012). Bases de données : le noSQL menace 25 ans de certitudes. Retrieved from http://www.indexel.net/infrastructure/bases-de-donnees-le-nosql-menace-25-ans-de-certitudes-3570.html

Lévy-Abégnoli, T. (2013). Explosion des volumes de données : de nouvelles architectures s'imposent. *2013*. Retrieved from http://blogs.microsoft.fr/decideurs-it/explosion-des-volumes-de-donnees-de-nouvelles-architectures-simposent-3.html

Lublinsky, B. (2014). Forrester Wave: Evaluating NoSQL Key-Value Databases. *2014*. Retrieved from http://www.infoq.com/news/2014/10/NoSQL

Mallassi, O. (2009). Consistent Hashing ou l'art de distribuer les données. *2009*. Retrieved from http://blog.octo.com/consistent-hashing-ou-l'art-de-distribuer-les-donnees/

Martignole, N. (2013). Redis, découverte d'un moteur clé-valeur simple et puissant. *2013*. Retrieved from http://www.touilleur-express.fr/2013/04/03/redis/

Masclet. (2010). 10 minutes pour comprendre...NoSQL. *2010*. Retrieved from http://davidmasclet.gisgraphy.com/post/2010/06/09/10-minutes-pour-comprendre...NoSQL

Maxicours. (n.d.). Le modèle relationnel. *2015*. Retrieved from http://www.maxicours.com/soutien-scolaire/comptabilite-et-finance-des-entreprises/terminale-stg/207845.html

Mohan, S. (2014). HTML5 : Learn how to use IndexedDB. *2014*. Retrieved from http://blogs.shephertz.com/2014/01/14/html5-learn-how-to-use-indexeddb/

MongoDB. (2014a). BRAND RESOURCES. *2014*. Retrieved from http://www.mongodb.com/brand-resources

MongoDB. (2014b). Sharding Introduction. *2014*. Retrieved from http://docs.mongodb.org/manual/core/sharding-introduction/

Moniruzzaman, A. B. M. (n.d.). NewSQL: Towards Next-Generation Scalable RDBMS for Online Transaction Processing (OLTP) for Big Data Management. Retrieved from http://arxiv.org/ftp/arxiv/papers/1411/1411.7343.pdf

Morello, M. (2012). Orientée colonnes ? *2012*. Retrieved from http://michaelmorello.blogspot.ch/2012/06/orientee-colonnes.html

Moulanier, R. (n.d.). Entrepôts de données et Big data: ce n'est pas la guerre, bien au contraire ! Retrieved from http://www.widoobiz.com/l-entrepreneur-pratique/entrepots-de-donnees-et-big-data-ce-nest-pas-la-guerre-bien-au-contraire/38479

Net, J. du. (2013a). HBase : le NoSQL au service du Big Data. *2013*. Retrieved from http://www.journaldunet.com/developpeur/outils/comparatif-des-bases-nosql/hbase.shtml

Net, J. du. (2013b). HBase : le NoSQL au service du Big Data. *2013*. Retrieved from http://www.journaldunet.com/developpeur/outils/comparatif-des-bases-nosql/hbase.shtml

Net, J. du. (2013c). Redis : stocker les données en mémoire en NoSQL. *2013*. Retrieved from http://www.journaldunet.com/developpeur/outils/comparatif-des-bases-nosql/redis.shtml

Paumard, J. (n.d.). Formes normales. *2013*. Retrieved from http://blog.paumard.org/cours/sql/chap05-jointures-formes-normales.html

PILLOU, J.-F. (n.d.). Le langage SQL. *2015*. Retrieved from http://www.commentcamarche.net/contents/1062-le-langage-sql

PostGreSQL. (n.d.). Write-Ahead Logging (WAL). Retrieved from http://docs.postgresql.fr/8.1/wal-intro.html

Rosenthal, D. (2013). Next gen NoSQL: The demise of eventual consistency? *2013*. Retrieved from https://gigaom.com/2013/11/02/next-gen-nosql-the-demise-of-eventual-consistency/

S.Louis. (n.d.). Normalisation : Formes normales. *2006*. Retrieved from http://www.ht.refer.org/coursenligne/base_donnees/normalisation.htm

Serries, G. (2014). Pratique - Big Data : 4 virages majeurs que doivent prendre les entrepôts de données. *2014*. Retrieved from http://www.zdnet.fr/actualites/pratique-big-data-4-virages-majeurs-que-doivent-prendre-les-entrepots-de-donnees-39809883.htm

Shvachko, K. V. (n.d.). Apache Hadoop. *Novembre 2013*. Retrieved from http://fr.slideshare.net/wandisco/hadoop-scalability

Sohm, J. C. (n.d.). Les bases de données relationnelles. *2002*. Retrieved from http://cerig.pagora.grenoble-inp.fr/tutoriel/bases-de-donnees/chap06.htm

StorageNewsletter. (n.d.). Stockage/gestion de données en tête des priorités informatiques pour 53% des entreprises françaises. *2012*. Retrieved from http://www.storagenewsletter.com/rubriques/market-reportsresearch/etude-emc-france/

Sybase. (2012). Dénormalisation de tables et de colonnes. Retrieved from http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.infocenter.dc31014.1610/doc/html/rad1232020345198.html

Tiwari, J. (2013). Setup a MongoDB Replica Set. Retrieved from http://jayatiatblogs.blogspot.ch/2013/01/setup-mongodb-replica-set.html

Torres, D., Gonzalez, L., & SaraTassini. (n.d.). LES BASES DE DONNES. Retrieved from http://tecfaetu.unige.ch/staf/staf-h/tassini/staf2x/Heidi/last_bd.htm

Tranchant, M. (n.d.). No Title. *juin 2011*. Retrieved from http://business-intelligence.developpez.com/tutoriels/quest-ce-que-la-bi/

Venkatesh, P. (2012). NewSQL — The New Way to Handle Big Data. *2012*. Retrieved from http://www.opensourceforu.com/2012/01/newsql-handle-big-data/

Vinette, L. (n.d.). PROGRÈS DE LA COLLECTE DES DONNÉES AUPRÈS DES ENTREPRISES. *2005*. Retrieved from http://www.bfs.admin.ch/bfs/portal/en/index/institutionen/statistikaemter_in/03/02.parsys.0057.downloadList.00571.DownloadFile.tmp/statcanplancomptable.pdf

WARIN, C. (2009). Héritage dans une base de données Access. Retrieved from http://warin.developpez.com/tutoriels/access/heritage-dans-base-donnees-access/

Wikipedia. (n.d.-a). Forme normale (bases de données relationnelles). *2014*. Retrieved from http://fr.wikipedia.org/wiki/Forme_normale_(bases_de_données_relationnelles)

Wikipedia. (n.d.-b). NoSQL. *2014*. Retrieved from http://fr.wikipedia.org/wiki/NoSQL

Wikipedia. (n.d.-c). Propriétés ACID. *2014*. Retrieved from http://fr.wikipedia.org/wiki/Propriétés_ACID

Wikipedia. (2014a). Base de données relationnelle. *2014*. Retrieved from http://fr.wikipedia.org/wiki/Base_de_données_relationnelle

Wikipedia. (2014b). Étoile (modèle de données). Retrieved from http://fr.wikipedia.org/wiki/%C3%89toile_%28mod%C3%A8le_de_donn%C3%A9es%29

Wikipedia. (2014c). Forme normale (bases de données relationnelles).

Wikipedia. (2014d). Hadoop. *2014*. Retrieved from http://fr.wikipedia.org/wiki/Hadoop

Wikipedia. (2014e). MapReduce. *2014*. Retrieved from http://fr.wikipedia.org/wiki/MapReduce

Wikipedia. (2014f). Memcached. *2014*. Retrieved from http://fr.wikipedia.org/wiki/Memcached

Wikipedia. (2014g). Parallélisme (informatique). *2014*. Retrieved from http://fr.wikipedia.org/wiki/Parallélisme_(informatique)

Wikipedia. (2014h). SQLite. *2014*. Retrieved from http://fr.wikipedia.org/wiki/SQLite

Wikipedia. (2014i). Traitement analytique en ligne. *28.08.2014*.

Wikipedia. (2015). Structured Query Language. *Wikipedia*. Retrieved from http://fr.wikipedia.org/wiki/Structured_Query_Language

Wordpress (Auteur inconnu). (2013). Orienté colonne. *2013*. Retrieved from https://basesdedonnees.wordpress.com/nosql/oriente-colonne/

Zeeman, T. (2013). EFFICIENTLY STORING YOUR DOMAIN MODEL IN RIAK. *2013*. Retrieved from http://blog.trifork.com/2013/06/20/efficiently-storing-your-domain-model-in-riak/