École Nationale Supérieure d'Électronique, Informatique, Télécommunications, Mathématique et Mécanique de Bordeaux

Département informatique

1 avenue du Dr Albert Schweitzer
B.P. 99 33402 Talence Cedex

PROJET DE FIN D'ÉTUDE / STAGE MASTER 2 RECHERCHE

Mise en place d'une solution non relationnelle de gestion de données

1er août 2012

Maître de stage :
Eric de MARIGNAN

Tuteur à BORDEAUX1 **:**
Mohamed MOSBAH
Sofian MAABOUT

Tuteur à l'ENSEIRB **:**
Denis LAPOIRE

Étudiant :
BARRO Lissy Maxime

Élève ingénieur
Dernière année - Option GL
ENSEIRB-MATMECA 2011/2012

Notice analytique

Maître de stage :
Eric de MARIGNAN

Tuteur à BORDEAUX1 :
Mohamed MOSBAH
Sofian MAABOUT

Tuteur à l'ENSEIRB :
Denis LAPOIRE

Étudiant :
BARRO Lissy Maxime

Élève ingénieur
Dernière année - Option GL
ENSEIRB-MATMECA 2011/2012

Titre :	Mise en place d'une solution non relationnelle de gestion de données
Résumé :	
Mots clés :	
Caractéristiques :	1 volume - x pages - x annexes
Type de travail :	Projet de fin d'étude / stage master 2 recherche 5 mois / 1er février - 30 juin
Date de fin de rédaction :	1er août 2012

Résumé

résumé

Abstract

abstract

Terminologie

Terme	Signification
SGBD	Système de Gestion de Base de Données
SGBDR	Système de Gestion de Base de Données Relationnel
BDD	Base De Données
BDDR	Base De Données Relationnelle
Mémoire cache	Mémoire qui copie temporairement des données provenant d'une autre source de données afin de diminuer le temps d'accès. Elle est plus rapidement accessible que le disque dur.
SNS	Social Networking Services

TABLE DES MATIÈRES

TABLE DES FIGURES

REMERCIEMENTS

Merci à tous!!

Première partie

Présentation du cadre de mon stage

CHAPITRE 1

PRÉSENTATION DE L'ORGANISME D'ACCEUIL : EVOLLIS

1.1 La fiche d'identité

EVOLLIS est une SAS - Société par Actions Simplifiée - au capital de 245000 € crée par M. Xavier PINSE en mars 2011. Sise à Bordeaux, EVOLLIS compte un effectif de 6 personnes. Le fonctionnement d'EVOLLIS est essentiellement orienté vers de la sous-traitance. EVOLLIS propose un nouveau produit, uZ'it, qui est une nouvelle solution financière. Grâce à sa solution, EVOLLIS permet aux particuliers d'accéder à des produits haut de gamme et dernier cri, pour quelques euros par mois sans craindre leur obsolescence.

1.2 Présentation du produit uZ'it

Le produit uZ'it est basé sur le principe de location avec option d'achat qui répond à un nouveau mode de consommation. Il est « packagée » et se compose de 3 services intégrés :

Le financement du produit en location : une véritable innovation commerciale, une alternative au crédit ou le client ne paye que la valeur d'usage et surtout la possibilité d'accéder à un produit d'une gamme supérieure grâce aux mensualités réduites.

La couverture complète durant toute la durée d'utilisation : à la solution de financement par location, s'ajoute des services associés comme la garantie totale du produit et le service Après-Vente. Le consommateur est ainsi est ainsi totalement couvert durant toute la durée du contrat.

La flexibilité en fin de contrat : au terme du contrat, le consommateur est libre de choisir entre la restitution du produit, l'option d'achat ou l'acquisition d'un nouveau bien «dernier cri».

1.3 Les partenaires d'EVOLLIS

EVOLLIS ne fonctionne pas tout seul, il se met en intermédiaire de plusieurs partenaires. Il combine ainsi les services de ses différents partenaires pour en faire un seul produit. Les partenaires d'EVOLLIS sont :

- ✦ Les distributeurs : en partenariat avec EVOLLIS, ils vont proposer le produit uZ'it aux clients finaux.
- ✦ Les organismes de financement qui financent les contrats de location.
- ✦ Les structures de dématérialisations qui permettent de vérifier la validité des pièces justificatives fournies par le client final.

- ✦ Les recycleurs qui interviennent à la récupération des produits à la fin du contrat de location.

CHAPITRE 2

PRÉSENTATION DU PROJET ZÉNOSQL

2.1 Le contexte

Dans le but de coordonnées les interactions entre ses partenaires, EVOLLIS échange avec ceux-ci des flux d'informations qui constituent de gros volumes de données hétérogènes. Aussi EVOLLIS propose l'offre uZ'it via une vignette image apparaissant sur la fiche produit du distributeur. La plateforme d'Evollis fera donc l'objet d'un accès concourant aux données et d'un fort trafic. Le but principale d'EVOLLIS est de pouvoir répondre efficacement aux différentes requêtes de génération de vignettes produits.

FIGURE 2.1 – Exemple de vignette produit avec un loyer de 38.43 €

EVOLLIS génère les vignettes à la volée. Le loyer est calculé à partir de données stockées en base. À chaque génération de vignette correspondent plusieurs accès à la base de données et éventuellement un recalcul du loyer si les données transmises par le distributeur ne correspondent pas à celles présentes en base. La génération des vignettes peut beaucoup impacter sur le temps de réponse de la plateforme. Pour ce faire, EVOLLIS met en place un système de cache pour éviter de régénérer les vignettes produits dont les données n'ont pas changé en base. Le mécanisme est illustré par le schéma ci-dessous.

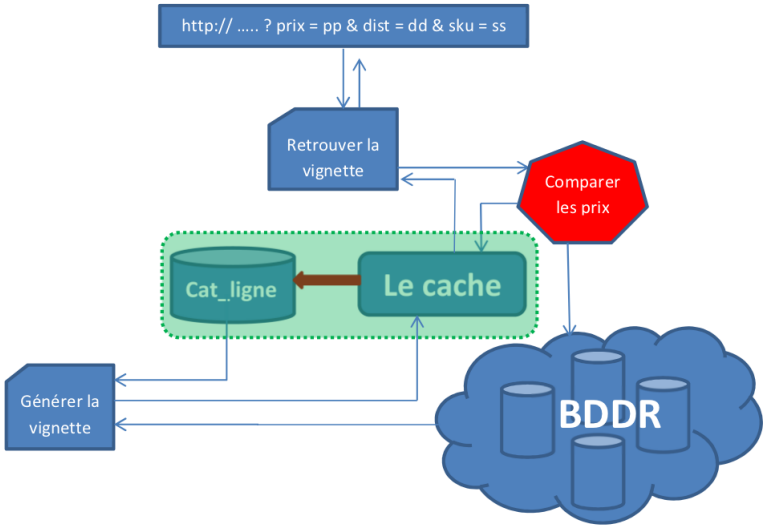

FIGURE 2.2 – Mécanisme de génération d'une vignette produit

Avec l'apparition du web 2.0 dans les années 2000, le nombre d'internautes a considérablement augmenté. Les sites de commerce rencontrent deux problèmes majeurs qui sont d'une part la taille et l'hétérogénéité des données stockées en base et d'autre part le temps de réponse. Dans le cadre du stage EVOLLIS est intéressé par une solution qui optimiserait la gestion des vignettes. À l'appel du Web Service de génération de vignette, les deux entités table en base « cat_ligne » et cache entrent en jeu conformément au mécanisme de génération de vignette défini plus haut. Les solutions non relationnelles de gestions des données connues sous la dénomination « NoSQL » proposent de nouvelles alternatives d'organisations physiques des données. L'idée ici du stage étant de voir comment tenir compte de celles-ci afin de fusionner la table en base « cat_ligne » et le cache en une seule entité pour meilleure gestion des vignettes comme l'illustre le schéma ci-dessous.

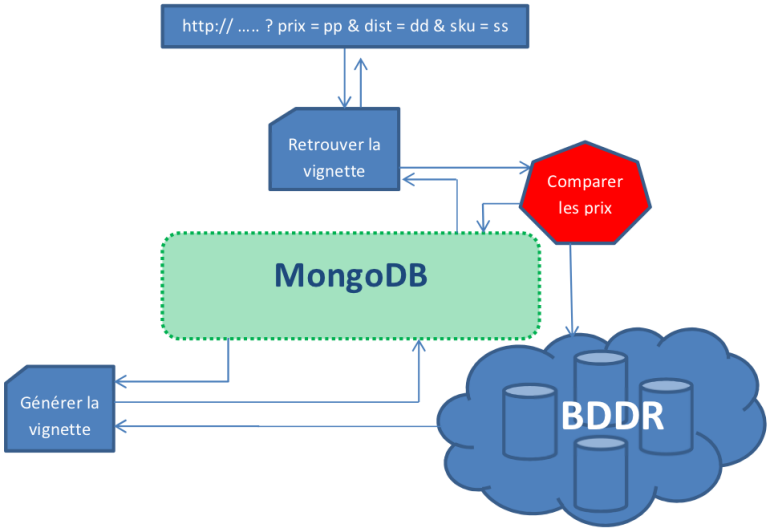

FIGURE 2.3 – Mécanisme de génération d'une vignette produit

2.1.1 Étude du gain de la fusion

Nous allons dans un premier temps

2.2 L'environnement de travail

2.3 L'équipe de travail

Deuxième partie

Déroulement de mon stage

CHAPITRE 3

ÉTAT DE L'ART

Avant de rentrer dans le vif du sujet, il est fondamental dans un premier temps de s'imprégner des différentes notions qui le détermine. Ainsi dans l'état de l'art vais-je aborder la notion clé du contexte de mon stage qui est le NoSQL. Il s'agit en effet d'une nouvelle famille de gestionnaire de BDD. Il en existe plusieurs réparties par catégories. Je prendrai le soin de parler de ces différentes catégories avec des exemples à l'appui. Je termine l'étude sur la notion NoSQL en définissant clairement les caractéristiques des gestionnaires NoSQL qui sont en quelque sorte un ensemble de propriétés qu'ils ont en commun. D'un premier abord, le NoSQL est considéré comme une alternative au SQL. Je ferai donc un rapprochement de ces deux familles en mettant en évidence ce qui les distingue. Ensuite je termine sur l'étude en abordant la problématique d'échange entre les deux familles SQL et NoSQL.

3.1 Le NoSQL

Le NoSQL signifiant littéralement « Not only SQL » est une dénomination désignant une nouvelle catégorie de gestionnaires de bases de données massives non relationnelles. Le NoSQL est une nouvelle mouvance dans la gestion des données qui se départit du relationnel pour rechercher plus de performance et de scalabilité[1]. Dans une base de données relationnelle, les données sont structurées dans des tables à deux dimensions selon un modèle permettant de dresser une relation entre elles, basée sur la théorie des ensembles et la logique mathématique[2]. Les BDDR classiques ont toujours cherché à implémenter les propriétés ACID [1][3] qui sont nécessaires à la gestion des transactions.

Pour certains cas d'utilisations comme la gestion des réseaux sociaux, un gestionnaire n'a pas forcement besoin des propriétés ACID. Le MySQL en est la preuve par sa popularité. Ce dernier est un SGBDR qui n'est toujours pas transactionnel en septembre 2010 avec sa version 5.5.6[4]. Cependant il est la BDD Open Source la plus utilisée avec plus de 65,000 téléchargements par jour[5]. Les utilisateurs de BDD n'ont pas forcement pour intention de gérer des transactions. Pour cette raison, les gestionnaires NoSQL optent pour la simplicité, la performance et la scalabilité au détriment des propriétés ACID et du modèle relationnel pour mieux parer la montée en charge dans des situations où les propriétés transactionnelles ne sont pas exigées. Celles-ci sont à l'origine d'*overhead*, temps consacré par un système à se gérer lui-même plutôt que d'effectuer le travail proprement dit. D'après Michael Stonebraker, pionnier des SGBD et selon des travaux menés dans le laboratoire de MIT, « 96% [des cycles machines de MySQL est] de l'*overhead* » partagé entre la gestion de buffer, les verrous au niveau enregistrement, l'écrit-

1. Pour plus de détails, voir annexe A

ure des logs et le multi-threading. Donc seulement « 4% [...] consacré au travail en lui-même »[6].

Les SGBD non-relationnels existent depuis le début de l'histoire des bases de données dans les années 60 notamment avec les systèmes de gestion de fichiers plus ou moins sophistiqués[2]. Ils sont de nos jours rependus sur les mainframes et les logiciels d'annuaire. Il sont plus anciens que les SGBD relationnels qui ont quant à eux fait leur apparition à partir de 1970[2]. Les SGBD non-relationnels ont connu une nouvelle jeunesse avec la mouvance NoSQL. La conférence meet-up de 2009 à San-Francisco est considérée comme l'inauguration de la communauté des développeurs de logiciels NoSQL. Ce retour aux non-relationnels est essentiellement motivé par les nouvelles demandes de performance et de scalabilité face à la montée en charge des sites web de grande audience apparus à partir des années 2000. La plupart des logiciels NoSQL sont ainsi destinés à être utilisés dans les dispositifs en répartition de charge des services Internet.

Les leaders de la communauté NoSQL sont issus des start-up internet pour lesquels la simplicité et le gratuité des SGBD sont un critère de choix important quite à risquer l'intégrité des données. Il est important de noter que la communauté NoSQL a effectivement mis en place des produits capables de manipuler de très grandes quantités de données qui se mesurent en centaines de `Téraoctets` et offrent une meilleure scalabilité mais force est de remarquer que les solutions NoSQL sont seulement adaptées pour certains besoins comme ceux des applications `Web 2.0` qui ne nécessitent pas de manoeuvres critiques comme les transactions dans la gestion des données. En effet le Web 2.0 désigne l'ensemble des fonctionnalités nouvelles et usages nouveaux du World Wide Web permettant aux internautes ayant peu de connaissances techniques d'être actif sur la toile, à l'image des réseaux sociaux.

Il y a une autre mouvance plus récente, le NewSQL, qui contrairement au NoSQL tente de conserver la structure classique relationnelle tout en faisant appel à différents procédés dans le but de conserver la rapidité, même sur de larges volumes[7]. Ces derniers ne font pas l'objet du présent document.

3.1.1 Les différents types de bases de données NoSQL

Dans la mouvance NoSQL, les données sont représentées de diverses manières. Les gestionnaires NoSQL sont ainsi classés par catégorie en fonction de la représentation des données. Ci-dessous les 4 principales catégories :

Clé - valeur : Représentation la plus simple, limitée à un mapping entre un ensemble de clé et un ensemble de valeurs. À chaque clé est associée une seule valeur dont elle ne connaît pas la structure. Ce postulat s'applique aux cas d'utilisation où les lectures et écritures sont réduites à un accès disque simple, c'est à dire que le système se charge uniquement de stocker et de restituer sans effectuer des opérations, ni sur la structuration des valeurs stockées, ni sur les relations entre elles. Cette catégorie trouve sa légitimité dans le constat que les applications accèdent à la base de données pour les mêmes informations. Les bases « clé-valeur » fonctionnent généralement en parallèle avec une autre base considérée comme la principale. Elles vont servir à stocker les résultats des requêtes récurrentes pour éviter de répéter les mêmes opérations dans la base principale[8].

Document : Ajoute au modèle clé-valeur une valeur à structure non plane qui nécessiterait un ensemble de jointures en logique relationnelle. La valeur est sous la forme d'un document contenant des données organisées de manière hiérarchique, à l'image de ce que permettent XML ou JSON[2]. La principale différence avec le modèle clé-valeur réside dans le fait que les valeurs

2. Java Script Object Notation

stockée sont désormais structurées. Cette propriété permet un accès beaucoup plus élaboré, non limité à une lecture ou écriture par clé, à la base de données. Cependant des opérations d'*hoverhead* supplémentaires notamment pour vérifier la structuration des valeurs stockées, diminuent sa performance par rapport au modèle clé-valeur en terme de lecture et écriture des données.

Colonne : Autre évolution du modèle clé-valeur, il permet de disposer d'un très grand nombre de valeurs sur une même ligne, permettant ainsi de stocker les relations de type one-to-many. Les lignes peuvent avoir des types de colonnes différents et également des nombres de colonnes différents. Il y a également une hiérarchie entre les colonnes. Cette configuration permet d'effectuer des requêtes par clé ou des opérations d'ensemble par enregistrement. Ceci n'est pas réalisable avec le modèle clé-valeur ou le modèle document dans la mesure où il y a une seule valeur par clé.

Graphe : très adapté à la modélisation, au stockage et à la manipulation des relations non-triviales ou variables entre les données. À l'image de la gestion des liens d'amitié sur les réseaux sociaux comme Facebook. Ces informations sont difficilement modélisables dans une base de données relationnelle.

Le site http://nosql-database.org recense 122 solutions NoSQL réparties en 9 catégories dont les principales sont celles citées ci-dessus. Pour une illustration des différentes représentations sur un exemple concret, se référer à l'annexe B.

3.1.2 Des exemples de NoSQL

Comme je l'ai mentionné à la section précédente, le site http://nosql-database.org recense 122 solutions NoSQL. J'en présenterai seulement 5. Un exemple par catégorie énumérée précédemment. Je m'intéresserai particulièrement aux solutions « Open Source » très utilisées. Je parlerai aussi de la solution propriétaire BigTable qui est considérée comme la première de la mouvance NoSQL. Dans l'ordre, Membase une solution orientée clé-valeur, MongoDB une solution orientée document, cassandra une solution orientée colonne, Néo4j une solution orientée graphe et BigTable une solution propriétaire orientée colonne.

Membase : Membase Solution NoSQL orientée clé-valeur, Open Source et diffusée sous la licence Apache 2.0. Écrit en C++/Erlang, il est soutenu par l'entreprise du même nom Membase[1]. Il a été développé par les leaders du projet Memcached qui est un autre système de stockage clé-valeur. Ce dernier permet la gestion de mémoire cache distribuée, utilise la RAM et n'est donc pas persistant. Memcached fut à l'origine développé par Brad Fitzpatrick pour le LiveJournal en 2003 et est aujourd'hui utilisé part de nombreux sites tels Wikipedia, Flickr, Bebo, Twitter, Typepad, Yellowbot, Youtube, Digg, WordPress.com, Craigslist, Mixi[9]. Membase rajoute au système Memcached la propriété de persistance sur le disque, la réplication des données pour assurer la résistance aux pannes et la scalabilité horizontale. Il est compatible avec les applications memcached existantes.

MongoDB : Solution NoSQL orientée document, Open Source et diffusée sous la licence GPL. Écrit en C++, il est soutenu par 10gen[1]. Son développement a débuté en Octobre 2007 et sa prémière version public est sortie en Février 2009[10]. Sur son site http://www.10gen.com/what-is-mongodb, 10gen écrit que le but de MongoDB est de combiner le modèle clé-valeur, rapide et scalable, et le modèle relationnel qui permet des opérations complexes comme les jointures et l'indexation. MongoDB stocke les données sous le format binaire BSON de JSON[3] dont la structure reste libre et dynamique. En effet, aucun schéma de BDD à respecter n'est

3. Ces notions sont expliquées plus en détail à la section 4.2 du présent document

prédéfini[11]. Le site web «ChinaVisual», la plus grande média en ligne chinoise , a annoncé en début d'année 2009 sa migration de MySQL vers MongoDB[12] alors que 10gen venait juste de publier la prémière version stable. Aujourd'hui, MongoDB est utilisé par de grands acteurs de la toile comme Buddy Media, Craigslist, Disney, Forbes, foursquare, Intuit, MTV Networks, Shutterfly, Traackr, Wordnik[13].

Cassandra : Solution NoSQL orientée colonnes de la fondation Apache, Open Source et diffusée sous la licence Apache 2.0. Comme toutes les solutions NoSQL, Cassandra met en avant le clustering. Il est écrit en Java[1]. Initialement développé par Facebook, le code source est devenu « Open-Source » sur Google code en juillet 2008 : http://code.google.com/p/the-cassandra-project/ En mars 2009, Cassandra rentre dans l'incubateur de projets de la fondation Apache, une passerelle de validation de projets désireux d'intégrer la fondation. En Février 2010, Cassandra dévient un projet à part entière de la fondation Facebook scale Cassandra sur plus de 150 machines. D'autres grands sites tels que Twitter et SoftwareProjects utilisent Cassandra. SoftwareProjects utilise 20 nœuds Cassandra à travers 3 entrepôts de données pour alimenter leur plate-forme e-commerce électronique pour 3000 entreprises. Il utilise Cassandra pour stocker des données d'achat en temps réel et fournir des statistiques aux divers clients. Cassandra est leur magasin de données principal[14].

Neo4j : Solution NoSQL orientée gaphe de l'entreprise Neo Technology, une entreprise suédoise[15]. Il est écrit en java[16]. La Première version, Neo4j 1.0 est sortie en Février 2010[17]. Sur le site http://neo4j.org/, j'ai pu lire que Neo4j est Open Source sous licence AGPLv3 et offre des performances 1000 fois supérieures aux BDDR classiques. En effet Neo4j permet à une application de bénéficier de toute l'expressivité des graphes pour modéliser toutes les dépendances non triviales entre les données. Il permet aussi de bénéficier d'algorithmes très optimisées de recherche et d'ajout d'éléments dans un graphe. L'utilisation de graphe permet surtout de répondre efficacement à la question d'existence de relation entre les éléments : la connexité, la complétude, l'accessibilité ... Neo4j est utilisé par Adobe, Viadeo, Deutsche Telecom, box, etc[18]. Il est important de noter que Neo4j répond aux enjeux traditionnels d'une BDD, principalement l'ACIDité des transactions. Il est considéré comme NoSQL juste parce qu'il n'utilise pas le schéma relationnel classique.

BigTable : Solution NoSQL orientée colonnes propriétaire, développé et exploité par Google. Cette BDD est proposée via la plate-forme d'application Google App Engine. Son développement a commencé en 2004 et BigTable est aujourd'hui utilisé par les applications Google telles que Google Earth, Blogger.com, Google Code hosting, YouTube, Gmail. BigTable a servi d'inspiration à des projets Open Source de solution NoSQL tels que HBase, Cassandra ou Hypertable[1].

N.B : certains SGBD comme Neo4j sont considérés comme NoSQL pour la seule et simple qu'il n'utilise pas le modèle relationnel de données représentées en tables. Neo4j implémente les propriétés ACID. Pour la suite, le terme NoSQL désignera tous les SGBD qui ont délibérément fait fi de l'ACIDité pour plus de performance et de scalabilité.

3.1.3 Les caractéristiques d'une base de données NoSQL

Le seul dénominateur commun entre les solutions NoSQL est la non utilisation du modèle relationnel classique[19]. Chaque solution a sa propre représentation des données et surtout son propre langage de requête[20]. Cependant toute les solutions NoSQL reposent sur les mêmes principes. Ci-dessous, 6 principales propriétés recherchées dans la mouvance NoSQL :

1. Open Source : la majeur partie des solutions NoSQL est Open Source
2. Scalabilité horizontale : fonctionner sur plusieurs machines peu coûteuses plutôt que sur

une seule machine puissante mais coûteuse[19]. En d'autres termes, pour gagner en performance, il suffit de rajouter une machine plutôt que d'augmenter la puissance d'une machine. Cette propriété est surtout implémentée pour les opérations simples. Par "opération simple", il faut comprendre la recherche par clé, lecture et écriture d'un enregistrement ou d'un petit groupe d'enregistrements. À l'opposé d'opérations complexes comme les jointures dans le modèle relationnel[1].

3. Réplication et distribution des données : les données sont répliquées et distribuées entre plusieurs serveurs pour améliorer l'accès et éviter les pertes de données en cas de panne. Une grande partie des solutions NoSQL effectue une mise à jour asynchrone. Cette notion est abordée plus en détail plus bas.
4. Un protocole simplifié : les solutions NoSQL mettent en avant la facilité d'usage, notamment une installation et configuration faciles et un langage de requête bas niveau.
5. Une gestion de concurrence faible : pour gagner en performance, les solutions NoSQL se focalisent moins sur l'intégrité des données et par conséquent sur la gestion des accès concourant aux données. Les solution NoSQL ne gèrent pas l'ACIDité des transactions. Cette est abordée plus en détail plus bas.
6. Souplesse et dynamisme : les BDD NoSQL se veulent libres de schémas et dynamiques. Ajout et suppression de colonnes sans arrêter le serveur et adaptation à n'importe quel type de données. Elles se démarquent du modèle relationnel stockent uniquement les données qui peuvent être distillées en tables.

Après avoir survolé les valeurs défendues par la mouvance NoSQL, je vais maintenant aborder la notion d'ACIDité en rapport avec le NoSQL. Je l'ai tantôt dit, les NoSQL n'ont pas pour vocation d'être ACID. En rappel, les propriétés ACID garantissent que la BDD garde toujours un état cohérent [4]. Cependant, à défaut d'être ACID, les solutions NoSQL revendiquent d'être BASE[1], acronyme de Basically Available, Soft state, Eventually consistency. Avec un BDD BASE, il n'y a pas la garantie que les données soient à jour sur tous les nœuds du cluster d'où le « Basically Available ». Cependant, après un temps assez long où il n'y a pas de nouvelles mises à jour d'où le « Soft state », la BDD pourra atteindre son état cohérent d'où l'« Eventually consistency ».

Le NoSQL a donc abandonné la « consistance forte » pour de la « consistance éventuelle ». Il a pris pour alibi le théorème CAP[21] qui postule que dans un environnement distribué, une BDD ne peut être à la fois consistante, disponible et résistante au morcellement. Elle ne peut avoir que 2 de ces 3 propriétés. Ce théorème est détaillé à l'annexe C du présent document. Ce théorème laisse entendre donc que le NoSQL, en renonçant à la « consistance forte », pourra préserver la disponibilité et la résistance au morcellement dans un environnement distribué.

Ce choix est tout à fait justifié au regard des valeurs défendues par la mouvance. Les solutions NoSQL entendent fonctionner dans un environnement distribué et utilisent la méthode de réplication des données pour résister aux pannes logicielles et machines. Elles ont pour vocation d'apporter plus de performance que les SGBDR dans l'exécution des opérations simples. Elles ont donc intérêt a choisir parmis les trois propriétés, la disponibilité et la résistance au morcellement.

4. voir annexe A pour plus de détails

3.2 SQL vs NoSQL

Il serait tout à fait légitime de penser que le NoSQL soit là pour remplacer les BDDR classiques. Mais ceci est difficilement imaginable de nos jours. L'intérêt pour le NoSQL s'est considérablement accru à l'issu des annonces d'adoption de ces technologies par les grands acteurs d'Internet tels Google et Facebook qui se sont multipliées. Ces acteurs n'ont pas pour autant abandonné BDDR. Google et Facebook utilisent MySQL[22].

Ci-dessous quelques traitements auxquels le modèle relationnel classique ne répond pas et qui pourrait motiver le recours aux technologies NoSQL :

Environnement distribué : pour faire face à des volumes importants de données, il est possible de les repartir sur différentes machines physiques. Pour avoir plusieurs points d'accès aux données, il est nécessaire de les dupliquer sur différents serveurs. Toutes ces opérations constituent en la mise en place d'un environnement distribué. Les SGBDR classiques montrent des limites dans un environnement distribué. Ces SGBD ne sont pas destinés à fonctionner dans un environnement à données reparties du fait de l'opération de jointure qui est difficilement réalisable entre des tables réparties sur des systèmes différents[1]. Les SGBDR classiques effectuent également des opérations impliquant de la distribution des données mais ces opérations ne sont pas du type « *shared nothing* ». Les coeurs et les processus impliqués dans ces opérations se partagent mutuellement la RAM et l'espace disque mémoire.

Données à structure dynamique et libre : les BDDR classiques prévoient un schéma statique à l'avance. Le schéma est organisé en tables de données où les lignes contiennent les mêmes types et nombre de colonnes. Celles-ci n'offrent donc pas un environnement dynamique pour les enregistrements. Aussi les SGBDR fonctionnent avec des données structurées organisables en tables. Ce n'est pas le cas avec des données non structurées, telles les données de traitement de texte et les images [19].

Réécritures fréquentes : en effet, les SGBDR classiques en générale appliquent la consistance forte et ce grâce aux propriétés ACID. Ces opérations sont à l'origine d'*overhead* et sont appliquées même pour les opérations simples d'écriture dans la base. Ceci diminuera considérablement la performance en cas de réécritures fréquentes des données même pour les opérations simples. Les BDDR mettent en avant un système d'indexation très évolué. L'utilisation d'index est plutôt conseillée pour les systèmes où l'accès en écriture est beaucoup plus important que celui en écriture. Tout ceci laisse penser que le modèle relationnel classique prévoit plus de lectures que d'écritures.

Extensibilité de la base : comme signaler à la section 3.1.3, la scalabilité horizontale qui est l'une des caractéristiques principales des solutions NoSQL, offre la possibilité d'ajouter des nœuds au cluster pour gagner en performance. Les SGBDR classiques n'ont pas cette propriété de scalabilité horizontale.

L'extensibilité requise, la grande quantité de données et les mises à jour massives rendent le modèle relationnel inefficace, ce qui a obligé à trouver un nouveau modèle. Cependant il est important de prendre en considération quelques aspects au risque d'une mauvaise utilisation. « L'intérêt d'une base de données NoSQL pour un projet ne dépend pas du volume de données qu'elle aura à manipuler. Le choix de son utilisation doit être basé sur la préférence d'un mode de représentation et non sur une forte volumétrie »[23]. Il ne s'agit donc pas d'une solution miracle pour tout type de stockage de données. La tentative de reproduire dans une base de

données NoSQL une représentation ou un comportement habituellement offert par la représentation en tables n'aboutira pas à une solution plus efficace car l'architecture des SGBDR y est exclusivement destinée.

À présent, je vais attirer l'attention sur une différence fondamentale entre ces deux familles de gestionnaires de données. Elles n'ont pas la même conception de la gestion des données. Le SQL impose un format de stockage le plus explicite possible à l'utilisateur pour permettre au gestionnaire d'avoir la maîtrise totale des données stockées afin d'en assurer l'intégrité. Le gestionnaire SQL ne se contente pas uniquement de répondre aux requêtes. Il effectue d'autres opérations qui consistent en l'entretien de la base à l'image de MySQL dont 96% des cycles machines sont de l'*overhead*. Le NoSQL vise à éviter les opérations d'*overhead* afin de consacrer toutes les ressources machines aux seuls traitements des requêtes pour être plus rapide. Cependant la cohérence dans la BDD nécessite beaucoup d'apport extérieur pour pallier à la très grande souplesse du système. Une illustration de cette différence philosophique entre les deux familles de gestionnaires est la suivante :

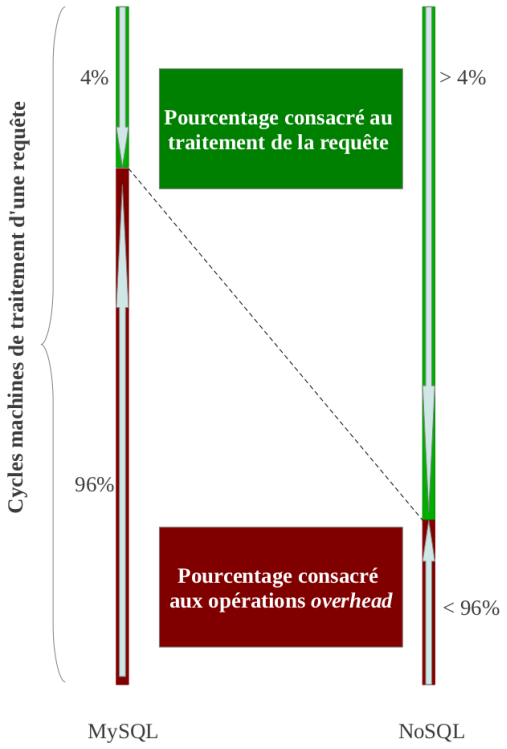

FIGURE 3.1 – Répartitions des cycles machines SQL et NoSQL

Toujours en quête de performance et comme mentionné à la section 3.1.3, le NoSQL a renoncé à de la « consistance forte » pour de la « consistance éventuelle ». Il justifie ce choix par le théorème de CAP pendant que les SGBDR défendent systématiquement toutes les propriétés ACID dont la « consistance forte ». Ceci n'est pas sans compromis, notamment pour la reprise sur erreurs. Michael Stonebraker explique clairement sur 8 cas d'erreur, les enjeux du choix des propriétés CAP[21]. Je me limiterai seulement à 3 cas d'erreur. Le but étant de mettre en relief les limites d'une « consistance éventuelle » et d'exhiber deux situations qui rendent impossible la propriété A de CAP défendue par la mouvance NoSQL.

Dans l'illustration qu'il a utilisé, Michael Stonebraker a considéré un ensemble de clusters interconnectés via un réseau WAN. Les nœuds à l'intérieur d'un cluster utilisent le LAN pour échanger. Ci-dessous trois cas d'erreur possibles :

Cas 1 Erreur au niveau de la couche application. Une application effectue une ou plusieurs mises à jour incorrectes. De telles erreurs ne sont généralement pas détectées dans les minutes qui suivent afin de revenir sur une version précédente de la base avant les mises à jour incorrectes.

Cas 2 Les erreurs reproductibles. Par exemple, une transaction fait planter le serveur principal. La même transaction fait tomber tous les autres serveurs copies qui vont servir de relais. Ces erreurs sont connues sous le nom de « Bohr bugs ».

Cas 3 Une catastrophe. Un cluster local est entièrement détruit.

Les deux premiers cas d'erreur conduisent la base dans un état incohérent. Ils affectent la disponibilité des données. Dans de telles situations, il est impossible de garantir la disponibilité des données même après avoir renoncé à de la consistance forte. Le troisième cas d'erreur montre les limites de la consistance éventuelle. Les données ne pourront être récupérées que si la transaction a été propagée sur un autre cluster avant la catastrophe. Le temps de latence imposé par la consistance éventuelle entre les mises à jour est donc problématique.

Pour finir, un paramètre à prendre en compte et pour ainsi reprendre la pensée de Ted Dziuba, est que contrairement aux solutions NoSQL, les SGBDR sont matures et ont par conséquent "une liste de limitations et de bugs connus"[24]. L'utilisateur d'une SGBDR classique est plus ou moins rassuré parce qu'il sait à quoi s'en tenir.

3.3 Échange entre SQL et NoSQL

Pour profiter des performances d'un NoSQL, il est essentiel de définir à l'avance la nature et la représentation faite des données. L'utilité et le choix du NoSQL doit principalement réposer de la représentation des données. Adopter le NoSQL à cause de la volumétrie ne garantit pas la performance et surtout la base perd l'intégrité qu'offre le modèle rélationnel.

Les « NoSQL » offrent une autre façon de représenter les données pour faciliter les accès et mises à jour des données afin de gagner en performance. Le NoSQL n'est pas là pour remplacer systématiquement le SQL. Pour la gestion des données organisée en tables, la structure des SGBDR y est exclusivement dédiée. Cependant certaines solutions NoSQL, à l'image de MongoDB, ont exhibé une charte de correspondance entre leurs composants et les composants BDDR[5]. MongoDB a également mis en place une charte de traduction requête SQL / requête MongoDB.

SQL d'une part, NoSQL d'autre part, l'un n'exclut pas l'autre. Tout dépend de l'usage qu'on compte en faire ou du profit qu'on souhaite en tirer. Pour profiter à la fois du NoSQL et du SQL, il est très commun de les faire cohabiter et de stocker les données dans l'un ou dans l'autre en fonction des caractéristiques de celles-ci. C'est ce que fait d'ailleurs Google qui même après avoir mis en place sa solution propriétaire BigTable utilise MySQL notamment pour son programme publicitaire AdWords, sa principale source de revenue, et qui utilise d'énormes quantités de données. Cependant imaginer un protocole d'échange entre SQL et NoSQL peut s'avérer très fastidieux dans la mesure où les deux conçoivent les données très différemment.

Il existe une solution Open Source permettant des échanges SQL/NoSqL : Sqoop ou « SQL To Hadoop ». Hadoop est un assemblage de plusieurs sous-projet avec à la base un système de fichier HDFS[6] et HBASE pour le stockage. Développé en java, sous la direction de la fondation Apache et destiné aux applications distribuées, sa structure de stockage HBASE est un xNoSQL orienté colonnes. Sqoop charge une table ou toute la base en système de fichiers HDFS en fonction des options d'importation spécifiées et génère aussi des classes java correspondantes aux tables de la bases, ainsi que des objets correspondants aux lignes pour permettre d'interagir avec les données importées.

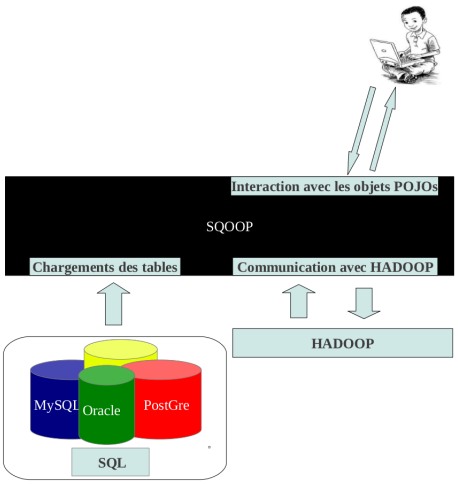

FIGURE 3.2 – Fonctionnement de SQOOP

Mise à part Sqoop qui est une solution pour Hadoop d'échange avec le SQL, le framework spring

5. Database, Table, Index, colonne, clé primaire ...
6. Hadoop Distributed File System

intègre des modules Data access[25] permettant de faire persister des objets POJO [7] dans des BDD NoSQL. En exemple, le module Spring Data MongoDB permet de faire persister les données dans MongoDB. Avec spring il est alors possible d'effectuer une échange en deux temps :

1. D'abord transformer les données dans la BDDR en des objets POJO par le biais de module déjà existant comme hibernate, toplink, iBatis...
2. Puis faire persister les objets POJO dans la base NoSQL par le biais du module Data access dédié.

Cependant il faut changer de module Data access à chaque fois qu'il faut changer de BDD NoSQL. Il est difficile de mettre en place une standardisation des interfaces dans la mesure où chaque BDD NoSQL vient avec son propre langage de requête.

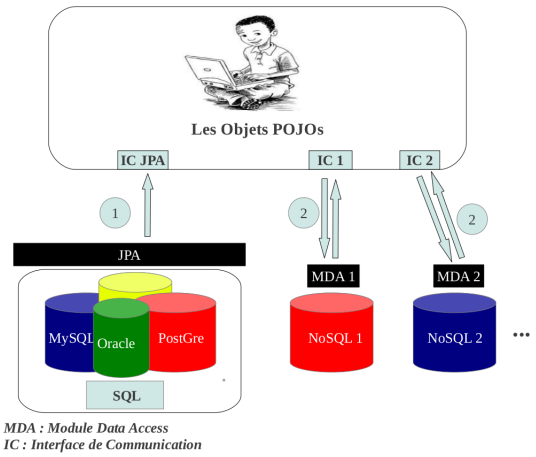

FIGURE 3.3 – Possibilité d'échange SQL/NoSQL avec spring

7. Acronyme de Plain Old Java Object désignant les objets java simples n'héritant d'aucune classe et n'implémentant aucune interface

CHAPITRE 4

CHOIX DES SOLUTIONS À ÉTUDIER

Comme je l'ai mentionné dans le coxtexte à la section **??**, Evollis utilise actuellement une BDDR classique, PGSQL. Il utilise également un système de cache, le Memcached.Un peu plus au dessus du present document à la section **??**, j'ai aussi parlé de Membase qui peut être vue comme une amélioration de memcached, en ce sens qu'il est compatible avec toutes les applications memcached avec des fonctionnalités supplémentaires telles que la persistance, le réplication et le clustering. Avec le système clé-valeur simple, le système n'est pas conscient de la structure de la valeur. Evollis veut tester une autre solution NoSQL qui est MongoDB. MongoDB utilise la techno clé-valeur mais rend le système conscient de la structure de la valeur stockée dans le but de permettre un champ opérationnelle plus large sur la base de données.

En raisons du nombre de solutions NoSQL existants, il fallait opérer un choix. Je m'intéresserai donc uniquement aux deux solutions membase et MongoDB pour les mêmes raisons mentionnées précédemment.

4.1 Le NoSQL Couchbase

Les systèmes de gestion clé/valeur comme memcached propose une mise en cache rapide des données. Il est utilisé par de nombreux sites tels Facebook et LiveJournal. t=The catch is that this application has got two significant shortcomings. First it is a memory-only database. That means if the power goes out, all the data is lost. Second, you can't actually search for datas using memcached ; you can only request specific keys. Firs and foremost, memcached is written to solve a specific problem. Memcached is not intended to be the permanent data store, but only to provide a caching layer to your existing database.

4.2 Le NoSQL mongoDB

Norme de stockage : Le BSON un dérivé binaire du JSON qui offre un bon rapport entre la place occupée et la rapidité de parsing notamment en intégrant la taille de chaque entrée pour pouvoir passer à la suivante si la clé ne correspond pas.

Toute cette partie est profondement inspirée de ... Avec quelques détails supplémentaires dont les sources sont citées au fur et à mesure.

replicaset et shards sont deux notions fondamentales dans la distribution des données de MongoDB.

Limitations dans l'indexations : On ne peut pas indexé des données (par un exemple un document ou des champs dans un document) depassant 800 bytes.
On peut indexer tant que la RAM le permet.
On a pas bésoin de réorganiser les données en table en table ou de changer leur formet avant de les stocker en base. Quand on accède aux données, aucun traitement de remise en forme est necessaire. Pas besoin d'un code intermediaire. MongoDB s'occupe de tout.
Les grandes idées : * L'organisation physique : collection, document, base de données, format de stockage BSON. *

CHAPITRE 5

MISE EN ŒUVRE DE LA SOLUTION MONGODB

5.1 Le modèle de données

5.2 L'architecture de l'application

5.3 Tests de charge sur MongoDB

5.4 Résultats

CHAPITRE 6

ACTIVITÉS TRANSVERSES AU SEIN D'EVOLLIS

INDEX

BIBLIOGRAPHIE

[1] Rick Cattell. Scalable sql and nosql data stores. *SIGMOD Record*, December 2010 (Vol. 39, No. 4).

[2] Cours de bases de données relationnelles. *www.cmi.univ-mrs.fr/~campioni/documents/BD/BD-relationnelles.pdf*.

[3] Daniel Bartholomew. Sql vs. nosql. *http://www.linuxjournal.com/article/10770*, 1er septembre 2010. Article apparu dans Linux Journal, Magazine mensuel spécialisé dans GNU/Linux et les logiciels libres.

[4] Brad Arrington. Mysql is not acid compliant. *http://www.remotedbaexperts.com/Blog/2010/11/mysql-is-not-acid-compliant/*, 22 novembre 2010. Exemple d'exécution de MySQL avec des résultats prouvant qu'il n'est pas ACID.

[5] Site officiel de MySQL. *http://www.mysql.com/why-mysql/marketshare/*. Visité le 29 février 2012.

[6] Michael Stonebraker. Newsql vs nosql for new oltp, michael stonebraker, voltdb. *http://www.slideshare.net/Dataversity/newsql-vs-nosql-for-new-oltp-michael-stonebraker-voltdb*, 24 Août 2011. Conférence de Michael Stonebraker, pionnier des SGBD et directeur technique chez VoltDB, sur sa solution NewSQL.

[7] Scriptol. Nosql. *Scriptol http://www.scriptol.fr/programmation/nosql.php*, 2011.

[8] Michaël Figuière. Nosql europe : Bases de données clé-valeur et riak. *http://blog.xebia.fr/2010/04/26/nosql-europe-bases-de-donnees-cle-valeur-et-riak/*, 26 avril 2010.

[9] Site officiel de Memcached. *http://memcached.org/about*, visité le 25 février 2012.

[10] Blog officiel de MongoDB. *http://blog.mongodb.org/post/434865639/state-of-mongodb-march-2010*, mars 2010. visité le 25 février 2012.

[11] Michaël Figuière. Nosql europe : Bases de données orientées documents et mongodb. *http://blog.xebia.fr/2010/04/30/nosql-europe-bases-de-donnees-orientees-documents-et-mongodb/*, 30 avril 2010.

[12] SHEN Shu GU Yunhua and ZHENG Guansheng. Application of nosql database in web crawling. *International Journal of Digital Content Technology and its Applications*, Volume 5, Number 6, June 2011.

[13] Site officiel de 10gen. *http://www.10gen.com/customers*, visité le 25 février 2012. Donne des explications sur les motivations de ses différents clients.

[14] Wiki officiel de la fondation Apache. *http://wiki.apache.org/cassandra/CassandraUsers*, 18 décembre 2010. visité le 25 février 2012.

[15] Michaël Figuière. Nosql europe : Bases de données orientées graphe et neo4j. *http://blog.xebia.fr/2010/05/03/nosql-europe-bases-de-donnees-graphe-et-neo4j/*, 3 mai 2010.

[16] Gavin Terrill. Neo4j - an embedded, network database. http://www.infoq.com/news/2008/06/neo4j/, 16 février 2010.

[17] The top 10 ways to get to know neo4j. *Neo4j Blog* http://blog.neo4j.org/2010/02/top-10-ways-to-get-to-know-neo4j.html, 16 février 2010.

[18] Site officiel de l'entreprise neotechnology. http://neotechnology.com/customers/. visité le 25 février 2012.

[19] Neal Leavitt. Will nosql databases live up to their promise ? *IEEE Computer Society 0018-9162*, 26 janvier 2010.

[20] Serge Leblal avec IDG NS. Newsql pour combiner le meilleur de sql et nosql. http://www.lemondeinformatique.fr/actualites/lire-newsql-pour-combiner-le-meilleur-de-sql-et-nosql-34475-page-3.html, 25 août 2011. visité le 1er mars 2012.

[21] Michael Stonebraker. Errors in database systems, eventual consistency, and the cap theorem. http://cacm.acm.org/blogs/blog-cacm/83396-errors-in-database-systems-eventual-consistency-and-the-cap-theorem/fulltext, 5 avril 2010.

[22] Site officiel de MySQL. http://www.mysql.com/customers/. Visité le 05 mars 2012.

[23] Michaël Figuière. Nosql europe : Tour d'horizon des bases de données nosql. http://blog.xebia.fr/2010/04/21/nosql-europe-tour-dhorizon-des-bases-de-donnees-nosql/, 21 avril 2010.

[24] Ted Dziuba. I can't wait for nosql to die. http://teddziuba.com/2010/03/i-cant-wait-for-nosql-to-die.html, Jeudi 4 mars 2010.

[25] Site du framework spring. http://www.springsource.org/features/data-access, visité le 7 mars 2012.

[26] Marwan KHELIF. Nosql : premiers pas avec cassandra. http://www.mkhelif.fr/2010/10/01/nosql-premiers-pas-avec-cassandra.html, 2010.

[27] Robert Rees. Nosql, no problem : An introduction to nosql databases. http://www.thoughtworks.com/articles/nosql-comparison.

[28] Dries Buytaert. Nosql and sql. http://buytaert.net/nosql-and-sql, 4 Decembre 2009 - 13 :05.

Troisième partie

Annexes

ANNEXE A

LES PROPRIÉTÉS ACID

Les propriétés ACID permettent à un SGBD d'effectuer des transactions. Par transaction, il faut comprendre une suite d'opérations qui font passer la BDD d'un état antérieur à un état postérieur. Les états intermédiaires entre les états avant la transaction et après la transaction ne sont pas visibles. Ses propriétés sont les suivantes :

Atomicité : la suite d'opérations constituant une transaction est indivisible. La transaction est entièrement effectuée ou pas du tout. Il y a annulation de toute la transaction lorsqu'une des opérations échoue. S'il est question de modifier une série de valeurs et qu'une modification échoue alors toutes les valeurs déjà modifiées reprennent leurs anciennes valeurs.

Cohérence : quelque soit l'opération effectuée, la base doit garder un état cohérent. Toute transaction qui viole par exemple une règle d'intégrité échoue. Après la fusion de deux tables, les entrées doivent toutes avoir des identités différentes. Si ce n'est pas le cas alors la fusion n'est pas effectuée.

Isolation : chaque transaction est isolée de sorte à ce qu'elle est seule peut voir les modifications pendant son exécution. Toute transaction enclenchée en parallèle d'une autre voit la version des données antérieure à celle-ci. Il existe 4 niveaux d'isolation définis dans le standard ANSI/ISO SQL :

1. Uncommited read ou lectures des données non validées. Ce niveau est le niveau d'isolation le plus léger. Avec un tel niveau d'isolation le système se comporte comme s'il n'y en avait pas. Les modifications apportées par une transaction non validée sont visibles.
2. Commited read ou lecture des seules données validées. Les modifications lors d'une transaction ne sont visibles que lorsqu'elle termine. Cependant lors d'une transaction une donnée peut changer sans que la transaction en cours en soit responsable. Ceci peut arriver, par exemple, lorsqu'une transaction en parallèle termine et que ses modifications sont validées.
3. Repeatable read ou lecture répétée. Ce niveau fonctionne comme le niveau précédent à la seule différence que durant son exécution, une transaction ne voit pas les mises à jour effectuées par d'éventuelles transactions qui se sont exécutées en parallèle. D'où « repeatable read » pour souligner que pendant une transaction, une donnée aura toujours la même valeur en lecture si elle n'est pas modifiée par la transaction elle-même. Cependant tout nouveau rajout validé de données au système par une transaction qui a terminé est visible par toute autre transaction en cours.
4. Serializable ou sérialisable. Ce niveau est le niveau d'isolation le plus poussé. Le système se comporte comme s'il n'y avait qu'une transaction à la fois. Pendant son exécution une

transaction ne voit ni les mises à jour, ni les rajouts de données au système des autres transactions. D'où « serializable » pour mettre en relief le caractère d'exécution en série des transactions plutôt qu'en parallèle.

Durabilité : dès lors qu'une transaction est validée, aucune défaillance du système ne pourra conduire à l'annulation de celle-ci. Les modifications liées à une transaction validée perdurent et ne sont jamais remises en cause.

ANNEXE B

ILLUSTRATION DES REPRÉSENTATIONS NOSQL

ANNEXE C

LE THÉORÈME DE BREWER OU LE THÉORÈME CAP

Jusqu'en 2002, le théorème n'était qu'une conjecture de énoncée par Brewer.

Théorème C.0.1 (de Brewer) *Dans un environnement distribué, un système ne peut être doté qu'au plus de deux des trois propriétés suivantes :*

- *Cohérence (Consistency)*
- *Disponibilité (Availability)*
- *Résistance au morcellement (Partition Tolerance)*

Cohérence : toute opération effectuée sur un nœud est automatiquement visible sur tous les autres nœuds du système. Si j'écris sur un noeud et lis sur un autre, je dois Disponibilité : Résistance au morcellement :

MySQL n'est pas transactionnel et ça n'a pas empêcher sa popularité.
A faire :

Trouver un nom au projet.
RDV avec Labri mercredi 15/02.
Pour le choix de solution :
Recherche de solution Opensource
Analyse de la communauté :
Dimension géographique de la communauté : France/Europe/monde
Nombre de membres
Activité de la communauté
Qualité de l'activité
Base installée de l'opensource
Stabilité de l'opensource (nb de versions/an ; compatibilité ascendante des versions)

Besoins :
Voir s'il est utile d'acheter des publications
Voir s'il est possible de commander un serveur sans OS : Dell/LDLC/PC21 : ...
Etablir un budget publications et serveurs