NOSQL

SMILE

I.T IS OPEN

PREAMBULE

SMILE

Smile est une **société d'ingénieurs experts** dans la mise en œuvre de **solutions open source** et l'intégration de systèmes appuyés sur l'open source. Smile est membre de l'**APRIL**, l'association pour la promotion et la défense du logiciel libre, du **PLOSS** – le réseau des entreprises du Logiciel Libre en Ile-de-France et du **CNLL – le conseil national du logiciel libre**.

Smile compte plus de 1200 collaborateurs dans le monde ce qui en fait le premier intégrateur français et européen de solutions open source.

Depuis 2000, environ, **Smile mène une action active de veille technologique** qui lui permet de découvrir les produits les plus prometteurs de l'open source, de les qualifier et de les évaluer, de manière à proposer à ses clients les produits les plus aboutis, les plus robustes et les plus pérennes.

Cette démarche a donné lieu à **toute une gamme de *livres blancs*** couvrant différents domaines d'application. La gestion de contenus (2004), les portails (2005), la business intelligence (2006), la virtualisation (2007), et la gestion électronique de documents (2008), les PGIs/ERPs (2008), les VPN open source (2009), les Firewall et Contrôle de flux (2009), les Middleware orientés messages (2009), l'ecommerce et les Réseaux Sociaux d'Entreprise (2010) et plus récemment, le Guide de l'open source (2011). Chacun de **ces ouvrages présente une sélection des meilleures solutions open source** dans le domaine considéré, leurs qualités respectives, ainsi que des retours d'expérience opérationnels.

Au fur et à mesure que des solutions open source solides gagnent de nouveaux domaines, Smile sera présent pour proposer à ses clients d'en bénéficier sans risque. Smile apparaît dans le paysage informatique français comme **le prestataire intégrateur de choix** pour **accompagner** les plus grandes entreprises dans l'adoption des meilleures solutions open source.

Ces dernières années, Smile a également étendu la gamme des services proposés. Depuis 2005, un département consulting accompagne nos clients, tant dans les phases d'avant-projet, en recherche de solutions, qu'en accompagnement de projet. Depuis 2000, Smile dispose d'un studio graphique, devenu en 2007 Smile Digital – agence interactive, proposant outre la création graphique, une expertise e-marketing, éditoriale, et interfaces riches. Smile dispose aussi d'une agence spécialisée dans la TMA (support et l'exploitation des applications) et d'un centre de formation complet, Smile Training. **Enfin, Smile est implanté à Paris, Lille, Lyon, Grenoble, Nantes, Bordeaux, Poitiers, Aix-en-provence et Montpellier. Et présent également en Espagne, en Suisse, au Benelux, en Ukraine et au Maroc**.

QUELQUES REFERENCES DE SMILE

SMILE est fier d'avoir contribué, au fil des années, aux plus grandes réalisations Web françaises et européennes. Vous trouvez ci-dessous quelques clients nous ayant adressé leur confiance.

Sites Internet

EMI Music, Salon de l'Agriculture, Mazars, Areva, Société Générale, Gîtes de France, Patrice Pichet, Groupama, Eco-Emballage, CFnews, CEA, Prisma Pub, Véolia, NRJ, JCDecaux, 01 Informatique, Spie, PSA, Boiron, Larousse, Dassault-Systèmes, Action Contre la Faim, BNP Paribas, Air Pays de Loire, Forum des Images, IFP, BHV, ZeMedical, Gallimard, Cheval Mag, Afssaps, Benetaux, Carrefour, AG2R La Mondiale, Groupe Bayard, Association de la Prévention Routière, Secours Catholique, Canson, Veolia, Bouygues Telecom, CNIL…

Portails, Intranets et Systèmes d'Information

HEC, Bouygues Telecom, Prisma, Veolia, Arjowiggins, INA, Primagaz, Croix Rouge, Eurosport, Invivo, Faceo, Château de Versailles, Eurosport, Ipsos, VSC Technologies, Sanef, Explorimmo, Bureau Veritas, Région Centre, Dassault Systèmes, Fondation d'Auteuil, INRA, Gaz Electricité de Grenoble, Ville de Niort, Ministère de la Culture, PagesJaunes Annonces…

E-Commerce

Krys, La Halle, Gibert Joseph, De Dietrich, Adenclassifieds, Macif, Furet du Nord, Gîtes de France, Camif Collectivité, GPdis, Longchamp, Projectif, ETS, Bain & Spa, Yves Rocher, Bouygues Immobilier, Nestlé, Stanhome, AVF Périmédical, CCI, Pompiers de France, Commissariat à l'Energie Atomique, Snowleader, Darjeeling…

ERP et Décisionnel

Veolia, La Poste, Christian Louboutin, Eveha, Sun'R, Home Ciné Solutions, Pub Audit, Effia, France 24, Publicis, iCasque, Nomadvantage, Gets, Nouvelles Frontières, Anevia, Jus de Fruits de Mooréa, Espace Loggia, Bureau Veritas, Skyrock, Lafarge, Cadremploi, Meilleurmobile.com, Groupe Vinci, IEDOM (Banque de France), Carrefour, Jardiland, Trésorerie Générale du Maroc, Ville de Genève, ESCP, Sofia, Faiveley Transport, INRA, Deloitte, Yves Rocher, ETS, DGAC, Generalitat de Catalunya, Gilbert Joseph, Perouse Médical…

Gestion documentaire

Primagaz, UCFF, Apave, Géoservices, Renault F1 Team, INRIA, CIDJ, SNCD, Ecureuil Gestion, CS informatique, Serimax, Véolia Propreté, NetasQ, Corep, Packetis, Alstom Power Services, Mazars…

Infrastructure et Hébergement

Agence Nationale pour les Chèques Vacances, Pierre Audoin Consultants, Rexel, Motor Presse, OSEO, Sport24, Eco-Emballage, Institut Mutualiste Montsouris, ETS, Ionis, Osmoz, SIDEL, Atel Hotels, Cadremploi, SETRAG, Institut Français du Pétrole, Mutualité Française…

Consulter nos références, en ligne, à l'adresse : http://www.smile.fr/clients.

CE LIVRE BLANC

Avec l'essor des grandes plateformes web, le volume de données à gérer par les applications a explosé, et les systèmes de gestion de données traditionnels, relationnels et transactionnels, basés sur le langage SQL, ont montré leurs limites.

Depuis quelques années, de nouvelles approches du stockage et de la gestion des données sont apparues, qui permettent de s'astreindre de certaines contraintes, en particulier de scalabilité, inhérentes au paradigme relationnel. Regroupées derrière le vocable **NoSQL**, ces technologies auraient très bien pu être nommées "NoRel" comme le suggère lui-même l'inventeur du terme NoSQL, Carl Strozzi.

Plus que des technologies de remplacement intégral des outils relationnels, le NoSQL correspond le plus souvent à des approches qui complètent les outils existants, pour en combler les faiblesses. Ainsi, on observe de plus en plus souvent la traduction "Not Only SQL" au lieu de "No SQL".

Ces technologies, portées par des acteurs majeurs du Web comme Google, Facebook, Twitter, ont rapidement acquis une légitimité réelle dans le domaine de la manipulation de volumétries de données très importantes et ont rapidement gagné tant en maturité qu'en notoriété.

Ce document dresse un tour d'horizon des différents paradigmes et technologies NoSQL disponibles. Il décrit également, pour les plus emblématiques de ces solutions, leurs caractéristiques, leurs forces et leurs faiblesses.

Bonne lecture !

N'hésitez pas à nous transmettre vos avis et évaluations sur les produits présentés dans ce livre blanc. Une seule adresse : contact@smile.fr

SOMMAIRE

ETAT DE L'ART

LES SGBD RELATIONNELS

Des acteurs bien installés

Les technologies de bases de données relationnelles et transactionnelles, qu'on résumera ici par "technologies SQL", règnent en maîtres pour le stockage et la manipulation de données depuis plus de 20 ans. Cette situation de leadership technologique peut facilement être expliquée par plusieurs facteurs.

Tout d'abord, SQL est un langage standardisé. Bien que chaque fournisseur de base de données ait implémenté quelques variantes, il n'en demeure pas moins que ce socle commun est bien connu par les développeurs. Pour une entreprise, investir dans une technologie SQL s'avère donc moins coûteux sur le plan de la formation que toute autre technologie. Cela explique sans doute l'échec des bases de données objets que l'on a vu fleurir autour des années 2000 et qui n'ont jamais véritablement percé, malgré une approche novatrice.

Ensuite, SQL embarque de nombreuses fonctionnalités importantes, adaptées aux besoins des entreprises, en particulier la gestion de l'intégrité des données et l'implémentation de transactions, indispensables pour bon nombre d'applications de gestion. Dans un monde où l'informatique de gestion a représenté une part très importante des besoins durant ces dernières décennies, ces fonctionnalités sont devenues attendues et indispensables dans une base de données.

Enfin, les outils permettant l'exploitation de ce type de bases de données sont matures. Que l'on se place au niveau des outils de développements (IDE graphiques, intégration dans les langages et les frameworks, …) ou d'exploitation (outils de sauvegarde, monitoring, …), les solutions existent et sont prêtes depuis déjà un certain temps. On peut donc considérer SQL comme un standard de fait pour toute problématique ayant trait au stockage et à la manipulation de données.

Pourtant, un certain nombre de limitations importantes sont apparues au fil des années. Les premiers acteurs à buter sur ces limites furent les fournisseurs de services en ligne les plus populaires, tels que Yahoo, Google ou plus récemment les acteurs du web social comme Facebook, Twitter ou LinkedIn. Le constat était simple : **les SGBD relationnels ne sont pas adaptés aux environnements distribués requis par les volumes gigantesques** de données et par les trafics tout aussi gigantesques générés par ces opérateurs.

Les besoins majeurs identifiés par ces acteurs sont les suivants :

- Capacité à distribuer les traitements sur un nombre de machines important afin d'être en mesure d'absorber des charges très importantes. On parlera de **scaling des traitements**.
- Capacité à répartir les données entre un nombre important de machines afin d'être en mesure de stocker de très grands volumes de données. On parlera plutôt de **scaling des données**.
- La distribution de données sur plusieurs datacenters afin d'assurer une continuité de service en cas d'indisponibilité de service sur un datacenter. Cette approche doit par ailleurs permettre que les données soient au plus proche des personnes qui les utilisent.
- Une architecture qui fonctionne sur du matériel peu spécialisé et donc facilement remplaçable en cas de panne.

Dans des contextes plus modestes, il est apparu que les solutions développées par ces opérateurs SaaS, disposaient de caractéristiques intéressantes en ce qui concerne la modélisation des données, permettant notamment de s'émanciper d'une certaine rigidité des modèles relationnels standards.

Limites des SGBD relationnels dans les architectures distribuées

Malgré l'appellation NoSQL, ce n'est pas tant le langage SQL en lui même qui est inadapté mais les grands principes sur lesquels il a été construit : le modèle relationnel et transactionnel. En effet, les bases de données relationnelles mettent à la disposition des développeurs un certains nombre d'opérations de relations entre les tables :

- un système de jointure entre les tables permettant de construire des requêtes complexes faisant intervenir plusieurs entités (les tables en l'occurrence)
- un système d'intégrité référentielle permettant de s'assurer que les liens entre les entités sont valides

La mise en œuvre de tels mécanismes a un coût considérable dés lors que l'on se trouve dans le contexte d'un système distribué. Sur la plupart des SGBD relationnels, il convient de s'assurer en permanence que les données liées entre elles sont placées sur le même nœud du serveur. Lorsque le nombre de relations au sein d'une base augmente, il devient de plus en plus difficile de placer les données sur des nœuds différents du système.

Nous verrons que les systèmes NoSQL adressent de différentes manières ce problème selon les besoins.

Outre leur modèle relationnel, la plupart des moteurs de SGBDs relationnels sont transactionnels ce qui leur impose le respect des contraintes **A**tomicity **C**onsistency **I**solation **D**urability, communément appelé par son acronyme **ACID** :

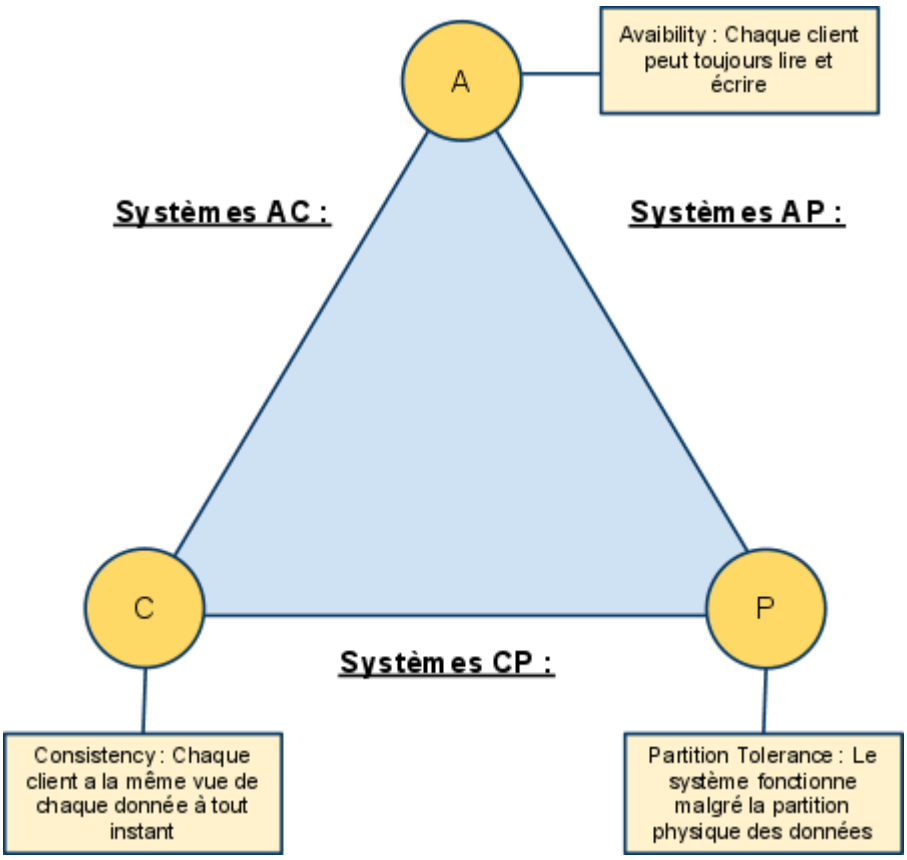
- A comme **Atomicity** : Cela signifie que les mises à jour de la base de données doivent être atomiques, c'est-à-dire qu'elles doivent être totalement réalisées ou pas du tout. Par exemple, sur 5000 lignes devant être modifiées au sein d'une même transaction, si la modification d'une seule échoue, alors la transaction entière doit être annulée. C'est primordial, car chaque ligne modifiée peut dépendre du contexte de modification d'une autre, et toute rupture de ce contexte pourrait engendrer une incohérence des données de la base.
- C comme **Consistency** : Cela signifie que les modifications apportées à la base doivent être valides, en accord avec l'ensemble de la base et de ses contraintes d'intégrité. Si un changement enfreint l'intégrité des données, alors soit le système doit modifier les données dépendantes, comme dans le cas classique d'une suppression en cascade, soit la transaction doit être interdite.
- I comme **Isolation** : Cela signifie que les transactions lancées au même moment ne doivent jamais interférer entre elles, ni même agir selon le fonctionnement de chacune. Par exemple, si une requête est lancée alors qu'une transaction est en cours, le résultat de celle-ci ne peut montrer que l'état original ou final d'une donnée, mais pas l'état intermédiaire. De fait, les transactions doivent s'enchaîner les unes à la suite des autres, et non de manière concurrentielle.
- D comme **Durability** : Cela signifie que toutes les transactions sont lancées de manière définitive. Une base ne doit pas afficher le succès d'une transaction, pour ensuite remettre les données modifiées dans leur état initial. Pour ce faire, toute transaction est sauvegardée dans un fichier journal, de sorte que si un problème survient empêchant sa validation complète, la transaction pourra être correctement terminée lors de la disponibilité du système.

Dans un contexte centralisé, les contraintes ACID sont plutôt aisées à garantir. Dans le cas de systèmes distribués, il est en revanche nécessaire de distribuer les traitements de données entre différents serveurs. Il devient alors difficile de maintenir les contraintes ACID à l'échelle du système distribué entier tout en maintenant des performances correctes. Comme on le verra, la plupart des SGBD de la mouvance NoSQL ont donc été construits en faisant fi des contraintes ACID quitte à ne pas proposer de fonctionnalités transactionnelles.

Il existe par ailleurs, une limitation plus théorique à ce que peuvent réaliser les bases de données relationnelles distribuées qui est décrite par le théorème de CAP (**C**onsistency-**A**vailibility-**P**artition tolerance). Celui-ci énonce trois grandes propriétés pour les systèmes distribués :

- C comme **Coherence** ou Cohérence : tous les nœuds du système voient exactement les mêmes données au même moment
- A comme **Availability** ou Disponibilité : la perte de nœuds n'empêche pas les survivants de continuer à fonctionner correctement
- P comme **Partition tolerance** ou Résistance au partitionnement : aucune panne moins importante qu'une coupure totale du réseau ne doit empêcher le système de répondre correctement (ou encore : en cas de morcellement en sous-réseaux, chacun doit pouvoir fonctionner de manière autonome)

Le théorème de **CAP** stipule qu'il est impossible d'obtenir ces trois propriétés en même temps dans un système distribué et qu'il faut donc en choisir deux parmi les trois :



Les bases de données relationnelles implémentent les propriétés de Cohérence et de Disponibilité (système AC). Les bases de données NoSQL sont généralement des systèmes CP (Cohérent et Résistant au partitionnement) ou AP (Disponible et Résistant au partitionnement).

Les limites des systèmes de SGBD relationnels face aux usages

Le stockage distribué n'est pas la seule contrainte qui pèse à ce jour sur les systèmes relationnels. Au fur et à mesure du temps, les structures de données manipulées par les systèmes sont devenues de plus en plus complexes avec en contrepartie des moteurs de stockage évoluant peu.

Le principal point faible des modèles relationnels est l'absence de gestion d'objets hétérogènes ainsi que le besoin de déclarer au préalable l'ensemble des champs représentant un objet. Pour répondre au besoin des objets hétérogènes non pré-déclarés, on a vu apparaître des modélisations complexes sur le plan algorithmique comme le modèle **E**ntity-**A**ttribute-**V**alue permettant de séparer un objet et ses champs. Ces modèles, quoique répondant au besoin de flexibilité des applications modernes, présentent un inconvénient majeur qui est leur très faible niveau de performance. La majeure partie des outils développés dans le cadre de la mouvance NoSQL permettent l'utilisation d'objets hétérogènes apportant comparativement une bien plus grande flexibilité dans les modèles de données ainsi qu'une simplification de la modélisation.

VERS L'ECLOSION DE DIFFERENTS PARADIGMES

Pourquoi la solution n'est elle pas unique ?

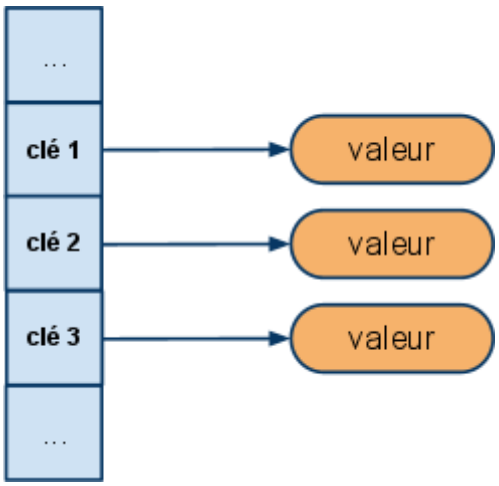
Nous l'avons vu, les SGDB relationnels disposent d'atouts indéniables expliquant leur popularité et leur large utilisation, mais aussi de limites importantes qui ont conduit à de nouvelles approches.

Tout comme les SGBD relationnels ne sont pas la réponse unique aux problématiques de base de données, on ne doit pas considérer qu'un seul autre outil sera en mesure d'apporter une solution au caractère universel. Rappelons à ce titre que la mouvance NoSQL ne prône d'ailleurs pas l'abandon total de SQL mais se traduit plutôt par "Not Only SQL" et non "No SQL". A ce titre, il existe dans la mouvance NoSQL une diversité importante d'approches. que nous classons **en quatre grandes catégories : Paradigme clé / valeur, Bases orientées colonnes, Bases documentaires et Bases orientées graphes**.

Types de base NoSQL : Paradigme clé / valeur

Il s'agit de la catégorie de base de données la plus basique. Dans ce modèle chaque objet est identifié par **une clé unique** qui constitue la seule manière de le requêter.

La structure de l'objet est libre et le plus souvent laissé à la charge du développeur de l'application (XML, JSON, ...), la base ne gérant généralement que des chaînes d'octets.



Dans ce modèle on ne dispose généralement que des quatre opérations **C**reate **R**ead **U**pdate **D**elete (CRUD):

- **Create** : créer un nouvel objet avec sa clé → create(key, value)
- **Read** : lit un objet à partir de sa clé → read(key)
- **Update** : met à jour la valeur d'un objet à partir de sa clé → update(key, value)
- **Delete**: supprime un objet à partir de sa clé → delete(key)

Les bases de ce type disposent pour la plupart d'une interface HTTP REST permettant de procéder très simplement à des requêtes, et ceci depuis n'importe quel langage de développement.

L'approche volontairement **très limitée** de ces systèmes sur le plan fonctionnel est radicale et leur permet d'afficher **des performances exceptionnellement élevées en lecture et en écriture ainsi qu'une scalabilité horizontale considérable**. Le besoin de scalabilité verticale est fortement réduit au niveau des bases par le caractère très simple des opérations effectuées.

Du fait de leur modèle de requête très simple et réduit aux opérations CRUD, ces systèmes sont principalement utilisés comme dépôt de données dès lors que les besoins en termes de requêtes sont très simples. On les retrouve très souvent comme système de stockage de cache ou de sessions distribuées, notamment là où l'intégrité relationnelle des données est non significative.

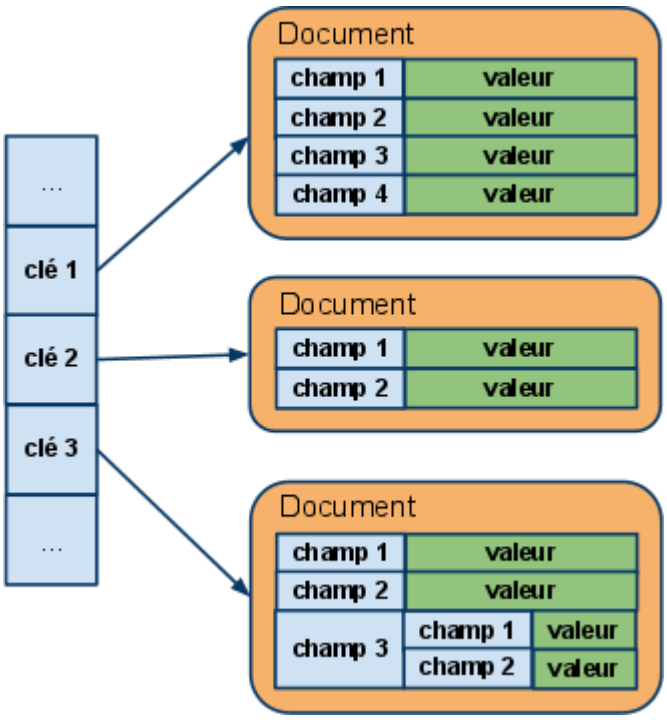
Types de base NoSQL : Bases documentaires

Les bases de données documentaires sont constituées de collections de documents. Un document est composé de champs et des valeurs associées, ces dernières pouvant être requêtées.

Par ailleurs, les valeurs peuvent être, soit d'un type simple (entier, chaine de caractère, date, ...), soit elles-mêmes composées de plusieurs couples clé/valeur.

Bien que les documents soient structurés, ces bases sont dites "schemaless". A ce titre il n'est pas nécessaire de définir au préalable les champs utilisés dans un document. Les documents peuvent être très hétérogènes au sein de la base.

Le stockage structuré des documents leur confère des fonctionnalités dont ne disposent pas les bases clés/valeurs simples dont la plus évidente est la capacité à effectuer des requêtes sur le contenu des objets.



On conserve généralement une interface d'accès HTTP REST permettant d'effectuer simplement des requêtes sur la base mais celle-ci est plus complexe que l'interface CRUD des bases clés/valeurs. Les formats de données JSON et XML sont le plus souvent utilisés afin d'assurer le transfert des données sérialisées entre l'application et la base.

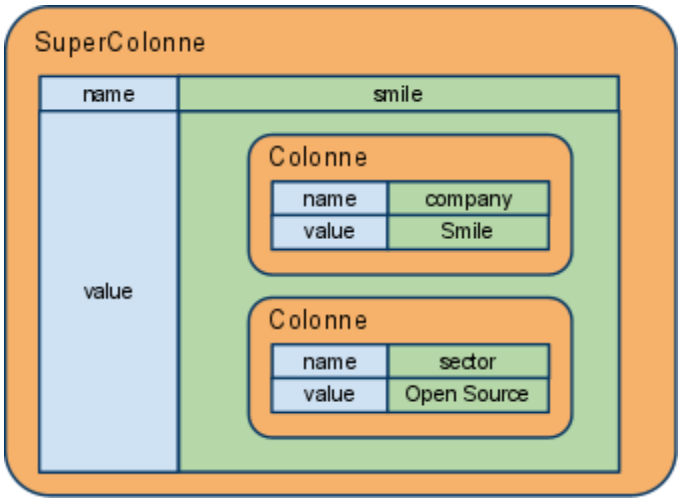
Du fait de leur conception plus simple que les bases de données relationnelles, ces bases conservent des performances élevées. Elles disposent par ailleurs de fonctionnalités très intéressantes en terme de flexibilité du modèle documentaire, ce que les bases relationnelles ne peuvent offrir. Cette dernière fonctionnalité leur confère une popularité importante, notamment dans le milieu du développement de CMS (Content Management System, outils de gestion de contenus) pour lesquels ce modèle est très adapté.

Types de base NoSQL : Bases orientées colonnes

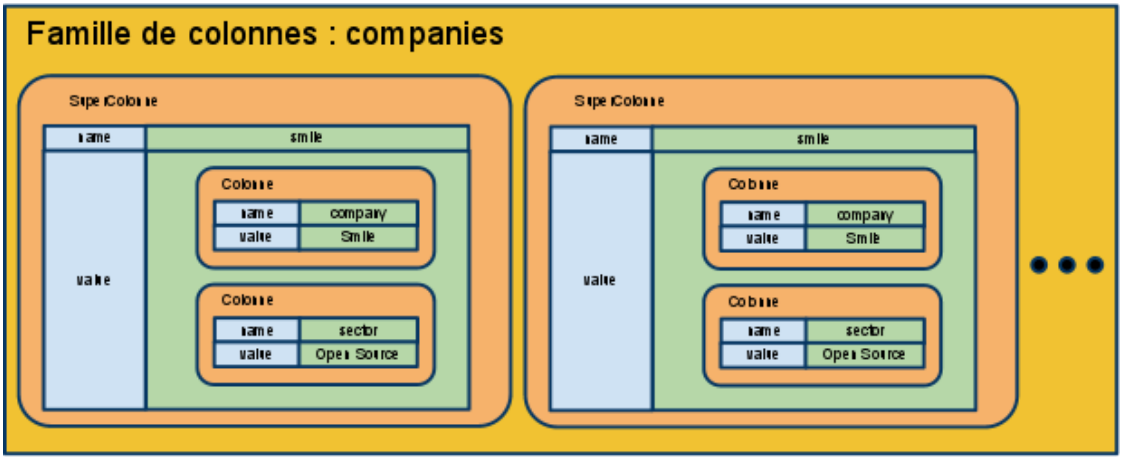
Les bases orientés colonnes sont résolument les plus complexes à appréhender parmi la mouvance NoSQL. Bien que l'on obtienne au final un schéma relativement proche des bases documentaires, l'organisation sous-jacente des données permet à ces bases d'exceller dans les traitements d'analyse de données et dans les traitements massifs (notamment via des opérations de type Map-Reduce).

Bien que l'on trouve des variations en fonction de la base considérée, les concepts essentiels sont les suivants :

- Colonne : il s'agit de l'entité de base représentant un champ de donnée. Chaque colonne est définie par un couple clé / valeur. Une colonne contenant d'autres colonnes est nommée super-colonne.



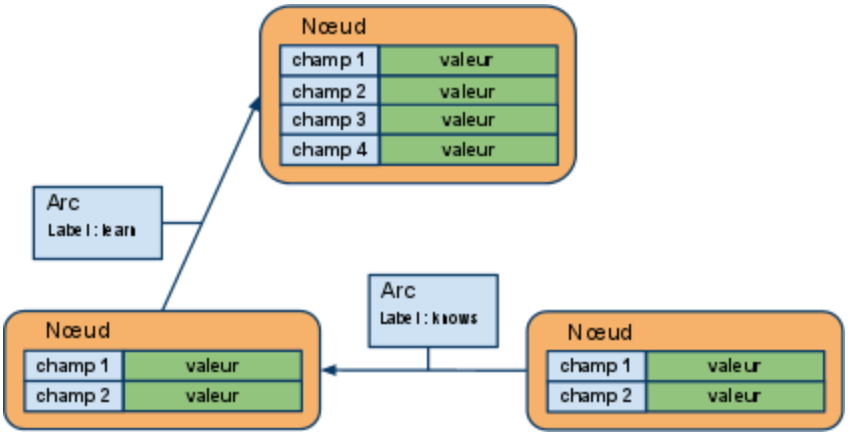
- Famille de colonnes : il s'agit d'un conteneur permettant de regrouper plusieurs colonnes ou super-colonnes. Les colonnes sont regroupées par ligne et chaque ligne est identifiée par un identifiant unique. Elles sont généralement assimilées aux tables dans le modèle relationnel et sont identifiées par un nom unique.



Les super-colonnes situées dans les familles de colonnes sont souvent utilisées comme les lignes d'une table de jointure dans le modèle relationnel. Comparativement au modèle relationnel, les bases orientée colonnes offrent plus de flexibilité. Il est effet possible d'ajouter une colonne ou une super colonne à n'importe quelle ligne d'une famille de colonnes, colonnes ou super-colonne à tout instant.

Types de base NoSQL : Paradigme graphe

Ce paradigme est le moins connu de ceux de la mouvance NoSQL. Ce modèle s'appuie principalement sur deux concepts : d'une part l'utilisation d'un moteur de stockage pour les objets (qui se présentent sous la forme d'une base documentaire, chaque entité de cette base étant nommée nœud). D'autre part, à ce modèle, vient s'ajouter un mécanisme permettant de décrire les arcs (relations entre les objets), ceux-ci étant orientés et disposant de propriétés (nom, date, …).



Par essence ces bases de données sont nettement plus efficaces que leur pendant relationnel pour traiter les problématiques liées aux réseaux (cartographie, relations entre personnes, ...). En effet, lorsqu'on utilise le modèle relationnel, cela nécessite un grand nombre d'opérations complexes pour obtenir des résultats.

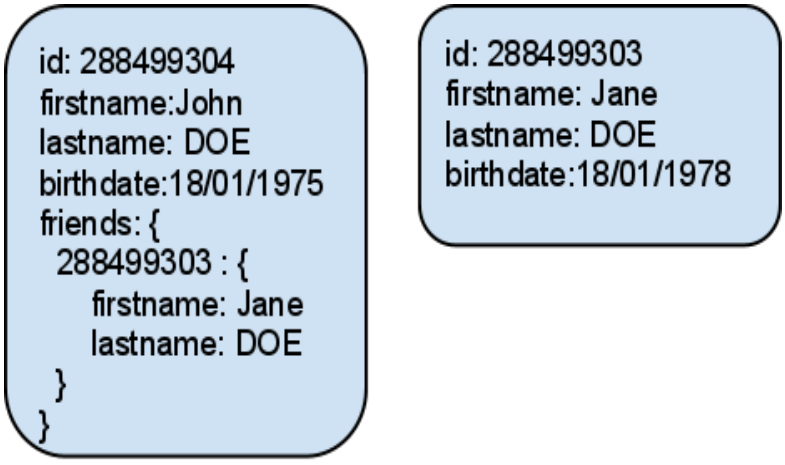
De nouveaux paradigmes entraînant de nouvelles pratiques

L'apparition de ces nouveaux paradigmes a entraîné des changements majeurs dans la manière d'aborder la conception des applications et leur rapport à la base.

Dénormalisation des données :

La première rupture est inhérente au modèle de données non relationnel qui ne dispose pas de méthodes efficaces permettant d'opérer des opérations de jointure. Dans le modèle relationnel, la pratique courante est d'utiliser des données sous forme normalisée et de procéder à des jointures entre les tables au besoin. Sur le sujet de la normalisation, les théoriciens de la mouvance NoSQL prônent une approche pragmatique guidée par l'utilisation qui est faite des données :

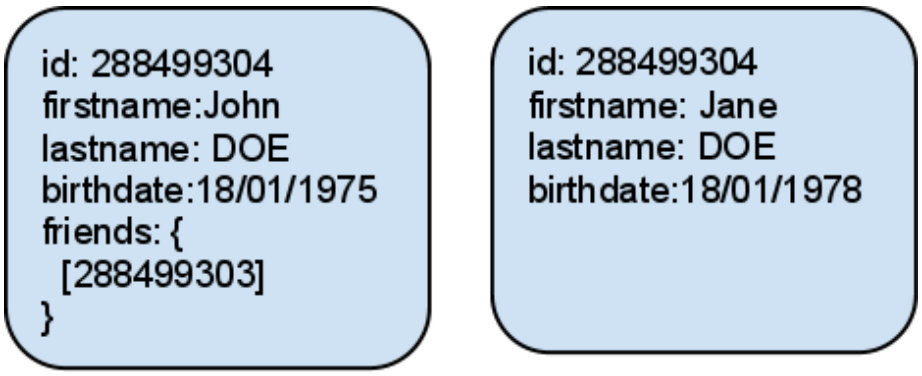
- Dans certains cas les données seront embarquées dans la donnée parente (la manière de procéder dépend du paradigme utilisé). C'est notamment le cas lorsque le volume de la donnée est faible et que celle-ci est peu mise à jour. Se pose alors la problématique de la mise à jour de données.



Avantages : Lecture rapide

Inconvénients : Lors de la mise à jour d'un documents ces références doivent être mise à jour

- Dans les autres cas, on stockera simplement les identifiants permettant d'accéder aux données comme on le ferait dans le modèle relationnel. La différence réside dans le fait que c'est le plus souvent à l'application de procéder par la suite aux jointures.



Avantages : - Ecriture rapide

Inconvénients : - Lecture moins rapide
- Reste à gérer le cas de la suppression du lien lors de la suppression de l'objet lié (inhérent au modèle non relationnel)

MapReduce :

MapReduce est une technique de programmation distribuée très utilisée dans le milieu NoSQL et qui vise à produire des requêtes distribuées. Cette technique se décompose en deux grandes étapes :

- **Etape de mapping** : Chaque item d'une liste d'objets clé/valeur est passé à la fonction map qui va retourner un nouvel élément clé-valeur. Exemples de fonction map :
  - A un couple (UserId, User), on assigne le couple (Role, User). A l'issue de l'étape de mapping, on obtient une liste contenant les utilisateurs groupés par rôle
  - A un couple (UserId, User) on assigne le couple (UserId, User) uniquement si l'email de l'utilisateur se termine par ".fr"
  - ...

- **Reduce** : La fonction reduce est appelée sur le résultat de l'étape de mapping et permet d'appliquer une opération sur la liste. Exemples de fonction reduce :
  - Moyenne des valeurs contenues dans la liste
  - Comptabilisation des différentes clés de la liste

- Comptabilisation du nombre d'entrées par clé dans la liste

L'étape de mapping peut être parallélisée en traitant l'application sur différents nœuds du système pour chaque couple clé-valeur. L'étape de réduction n'est pas parallélisée et ne peut être exécutée avant la fin de l'étape de mapping.

Les bases de données NoSQL proposent diverses implémentations de la technique MapReduce permettant le plus souvent de développer les méthodes map et reduce en Javascript ou en Java.

PANORAMA DES SOLUTIONS NOSQL EXISTANTES

On dénombre actuellement une douzaine de solutions existantes, qu'elles soient de type :

- base clé/valeur, comme Redis, Riak, Voldemort
- base documentaire, comme MongoDB, CouchDB, Terrastore
- base orientée colonne, comme Cassandra, Amazon SimpleDB, Google BigTable, HBase
- base orientée graphe, comme Neo4j, OrientDB

Dans la mouvance NoSQL, les grands noms de la base de données propriétaire brillent par leur absence, les solutions citées ci-dessus étant soient open source, soit proposées en SaaS.

Les produits issus du monde SaaS et proposés sur ce mode uniquement, principalement portés par Google et Amazon, font partie intégrante de l'offre **PaaS** (Platform as a Service) mise à disposition des développeurs dans le cadre des frameworks de développements clouds.

Les produits open source constituent la majorité de l'offre NoSQL, et l'on assiste à une réelle montée en de ces solutions dans ce secteur, souvent portée par des contributeurs importants comme Facebook, Twitter, LinkedIn, … Un certain nombre de produits ont été développés pour les usages internes de ces grands acteurs de l'Internet, puis libérés dans des états d'avancement très important, puisque déjà en production pour ces grands sites web. Notons que la fondation Apache joue ici encore un rôle primordial dans ces projets.

Nous listons ci-après les solutions, leur licence et une première analyse.

SOLUTION DE TYPE "BASE CLE / VALEUR"

Outil	Distribution	Commentaire
Redis http://redis.io/	Licence BSD	Redis dispose de fonctionnalités légèrement plus avancées que la majeure partie des autres systèmes de type clé/valeur, permettant notamment la manipulation des chaînes ou le stockage de collections. Il ne dispose en revanche pas de réel mécanisme de partitionnement mais simplement de réplication maitre/esclave. Ce projet est sponsorisé par VMWare ce qui lui assure une certaine pérennité.
Riak http://wiki.basho.com/	Apache License 2.0	Riak est l'implémentation open source de Amazon Dynamo. Le principaux avantages de Riak sont : système complètement distribués, support Map/Reduce (Javascript/Erlang) permettant de procéder à des traitements distribués à grandes échelles.
Voldemort http://project-voldemort.com/	Apache License 2.0	Un système prometteur développé par LinkedIn pour leur usage interne puis libéré en open source. Un travail important a été réalisé sur l'optimisation de la communication entre les différents nœuds du réseau.

SOLUTIONS DE TYPE "BASE DOCUMENTAIRE"

Outil	Distribution	Commentaire
MongoDB http://www.mongodb.org/	Licence GNU AGPL v3.0	MongoDB est l'une des bases documentaires open source les plus populaires. Stable depuis longtemps, elle dispose d'une couverture fonctionnelle importante ainsi que d'un support professionnel.
CouchDB http://couchdb.apache.org/	Apache License 2.0	CouchDB est le projet de base documentaire officiel de la fondation Apache. L'approche est radicalement orientée sur les méta-données, le versionning et le contenu des documents.
Terrastore http://code.google.com/p/terrastore/	Apache License 2.0	Terrastore est une base documentaire qui est assez intéressante dans la mesure où celle-ci est extensible (développement en Java) via un système d'événements. Le langage de requête est lui aussi extensible par ce même biais.

SOLUTIONS DE TYPE "BASE ORIENTEE COLONNES"

Outil	Distribution	Commentaire
Cassandra http://cassandra.apache.org/	Apache License 2.0	Cassandra est la base de données orientée Colonnes open source la plus populaire. Son développement est mené sous l'égide de la fondation Apache et elle est utilisée par des acteurs de premier plan du Web comme Facebook.
Amazon SimpleDB http://aws.amazon.com/fr/simpledb/	SaaS	Brique essentielle de Amazon AWS, la base de données SimpleDB permet de développer des applications dans le cloud Amazon.
Google BigTable	SaaS	BigTable est la base de données de Google App Engine, l'environnement de développement SaaS de Google. Seules les APIs de App Engine permettent son exploitation.
HBase http://hbase.apache.org/	Apache License 2.0	HBase est une implémentation open source de Google BigTable basée sur Hadoop. Il s'agit, avec Cassandra, du deuxième projet d'envergure menée par la fondation Apache en ce qui concerne les bases orientées colonnes.

SOLUTIONS DE TYPE "BASE ORIENTEE GRAPHES"

Outil	Distribution	Commentaire
Neo4j http://neo4j.org/	Licence GNU GPL v3.0 / Version commerciale	Cette base de données orientée Graphes est disponible en version stable depuis janvier 2010 et dispose d'un support commercial.
OrientDB http://www.orientechnologies.com/	Apache License 2.0	OrientDB est une base de données orientée graphe qui présente la particularité d'implémenter, entre autres, une partie du langage SQL comme possibilité de requête.

OUTILS GENERALEMENT ASSOCIES

Bien que cela ne soit pas le cœur de ce livre blanc, on observe souvent la présence d'outils complémentaires communs dans le cadre de projets NoSQL. En effet, les besoins suivants bien que ne faisant pas partie du spectre des problématiques traités par les bases de données NoSQL sont récurrents dans les projets distribués :

Système de fichiers distribués :

Ce besoin est très souvent associé à des bases de données volumineuses. Moins coûteux que des architectures de type SAN, il existe des systèmes de fichiers distribués dont les plus connus sont **Lustre**[1] et **MogileFS**[2].

Recherche distribuée :

La recherche est souvent un élément clé sur un système distribué. On utilisera le plus souvent le moteur de recherche open source **SolR**[3] qui présente la capacité de construire simplement un index de recherche distribué.

Processus applicatif distribué :

Lorsque les volumes de données deviennent importants, il peut apparaître que les traitement non assurés par la base de données ne soient plus réalisables par un seul et même nœud applicatif. Il existe un certain nombre d'outils permettant d'obtenir des systèmes de traitement de données distribués dont les plus connus sont les environnements **Hadoop**[4] (Java) et **Node.JS**[5] (JavaScript). Il existe par ailleurs des langages de programmation spécifiquement conçus pour ce type de problématique comme **Erlang**.

[1] Luste FS : http://wiki.lustre.org/index.php/Main_Page

[2] MogileFS : http://danga.com/mogilefs/

[3] SolR : http://lucene.apache.org/solr/

[4] Hadoop : http://hadoop.apache.org/

[5] Node.JS : http://nodejs.org/

Gestion de configuration distribué :

La gestion de configuration est une problématique difficile en informatique qui se doit d'être adressée par des outils spécifiques dès lors que l'on se place dans un environnement distribué. L'un des outils le plus souvent utilisé est **Apache ZooKeeper**[6].

[6] Apache ZooKeeper : http://zookeeper.apache.org/

SELECTION DES MEILLEURES SOLUTIONS OPEN SOURCE

Dans la liste précédente, nous avons sélectionné les meilleures solutions open source, c'est-à-dire celles qui présentent à la fois la maturité la plus grande, le potentiel le plus important, ou encore la meilleure intégration aux outils de développement actuels.

Cette sélection nous amène à une liste réduite à trois produits clés :

- Cassandra,
- MongoDB,
- CouchDB

CASSANDRA

La base de données Cassandra est née dans les locaux de Facebook et a été développée pour prendre en charge les très gros volumes de données générés par la boite de réception de Facebook. Le projet fut finalement placé sous l'égide de la fondation Apache qui assure maintenant la conduite des développements.

Cassandra a été construit comme une base de données orientée colonnes et respecte donc les principes présentés plus haut pour ce paradigme.

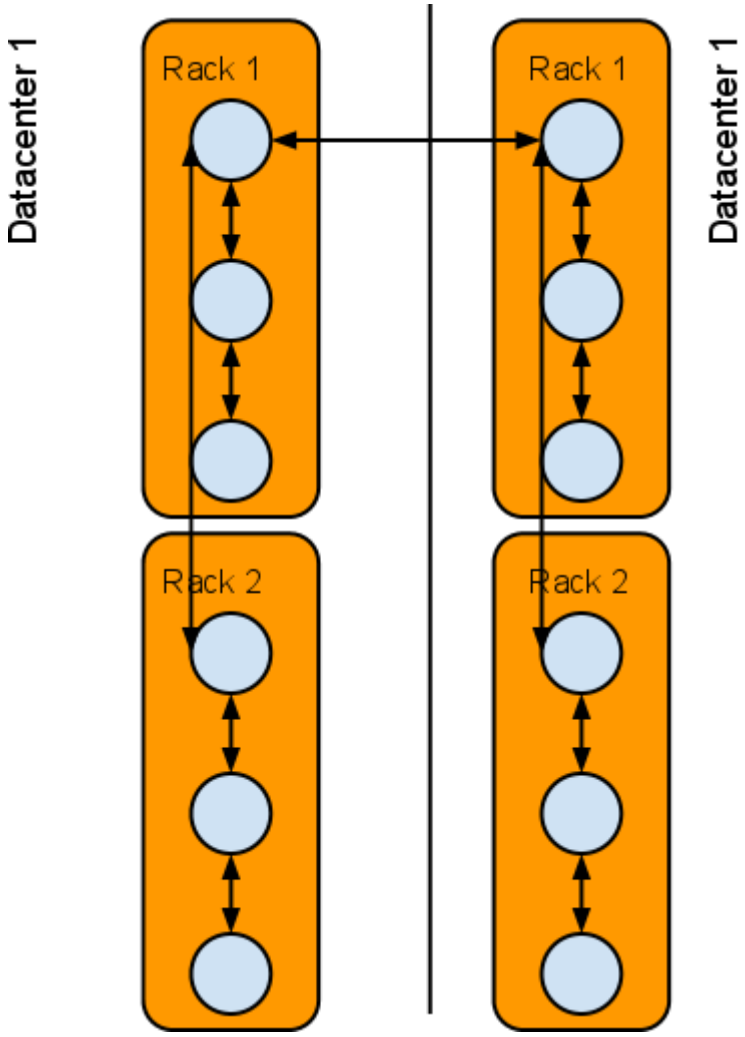
Architecture

L'architecture de Cassandra a été conçue pour être complètement distribuée et ne pas présenter de point de défaillance unique (**S**ingle **P**oint **O**f **F**ailure ou **SPOF**). Pour obtenir ce résultat, tous les nœuds de Cassandra sont équivalents. L'ajout d'un nouveau nœud est très simple et il suffit pour cela de connaître un seul nœud du cluster (en général, on choisit un certain nombre de nœuds assez stables nommés **Seed** pour ajouter de nouveaux nœuds). Au final, les nœuds sont organisés sous la forme d'un anneau. En cas de défaillance de l'un des nœuds, il est simplement retiré de l'anneau.

Les données sont partitionnées et répliquées entre les différents nœuds. Chaque famille de colonnes est en fait stockée comme une table de hashage partitionnée de manière homogène sur le cluster. La stratégie de génération de clés et de partitionnement peut être modifiée par le concepteur de la base pour chaque famille de colonnes.

En outre Cassandra est capable de gérer deux concepts différents afin de faciliter son exploitation en datacenters (éventuellement répartis) :

- Les racks : Il s'agit de groupe de serveurs liés entre eux. Le groupe de serveurs d'un même rack contient l'ensemble des données et est autonome. On place souvent deux racks dans le même datacenter. Les données sont automatiquement dupliquées entre deux racks
- Le datacenter : A chaque rack, on associe un jumeau dans un autre datacenter. Cela permet d'assurer la réplication entre les deux sites distants.



La mise en place de ce type de topologie est gérée via des fichiers de configuration qui doivent être distribués sur l'ensemble des nœuds. Pour faciliter la gestion de telles configurations distribuées, on préconisera l'utilisation de l'outil Zookeeper.

Outils de développement

Stabilité de l'API :

Cassandra est un produit relativement jeune dont l'API évolue fortement. Dans un premier temps, l'accès à la base était uniquement possible via un mécanisme de procédure stockée (RPC) au travers de Thrift[7]. L'équipe de développement a entrepris une migration de Thrift vers Avro[8] (tous deux sont des projets de la fondation Apache) ce qui risque de rendre Thrift obsolète à terme.

Au fur et à mesure de l'avancement des développements, l'équipe projet ajoute de nouveaux modes d'accès comme le langage CQL, permettant d'accéder à la base à l'aide d'une syntaxe proche du SQL. Ces modes d'accès sont relativement récents dans la vie du projet et leur API n'est donc pas à considérer comme stable.

D'une manière générale, le projet reste assez neuf à ce jour et son API est encore sujette à de nombreuses évolutions et variations.

Langages supportés :

Thrift est la méthode d'accès la plus universelle pour accéder à Cassandra. Cependant, il s'avère plutôt rudimentaire dans sa manipulation. A ce titre, bien que Thrift offre le support le plus universel, on privilégiera, lorsqu'elles existent, des bibliothèques de plus haut niveau s'appuyant sur Thrift. Le support du langage CQL est pour le moment réservé aux seuls langages Java et Python.

Langage	Thrift	CQL	Commentaires
Java	Oui	Oui	Existence de bibliothèques de plus haut niveau : Hector, Pelops, Cassandrelle
Python	Oui	Oui	Existence de bibliothèques de plus haut niveau : Pycassa, Telephus
PHP	Oui	-	Existence de bibliothèques de plus haut niveau : Cassandra PHP Client Library, phpcassa
Ruby	Oui	-	Existence de bibliothèques de plus haut niveau : Cassandra
C++	Oui	-	

[7] Thrift : http://thrift.apache.org

[8] Avro : http://avro.apache.org

Perl	Oui	-	
C# / .Net	Oui	-	Existence de bibliothèques de plus haut niveau : Aquiles
JavaScript	Oui	-	
Cocoa (via Objective-C)	Oui	-	

On notera par ailleurs, qu'à ce jour, aucun des frameworks majeurs de développement couramment utilisés (Zend, Symfony, Django, ...) n'intègre de support pour Cassandra. Certains planifient néanmoins son support. Cela risque de s'avérer difficile tant que l'API ne sera pas stabilisée.

Gestion de la cohérence des données :

Comme indiqué plus tôt, il est techniquement impossible de maintenir la cohérence des données sur l'ensemble du cluster tout en conservant les propriétés de disponibilité ainsi que le partitionnement des données (théorème CAP). Cassandra assure uniquement la cohérence finale des données, ce qui signifie uniquement après un délai de propagation au travers du datacenter. Cela fait de Cassandra un système de type AP (Available et Partition-Tolerant).

Cependant et dans certains cas précis, il peut être nécessaire de s'assurer que la donnée écrite soit écrite sur l'ensemble des nœuds du cluster avant de valider l'opération. De manière analogue, lors d'une lecture un développeur peut souhaiter disposer de la dernière mise à jour d'une donnée effectuée à l'échelle du cluster tout entier. Ces cas sont gérés directement par le développeur, qui doit préciser lors de sa requête quel degré de cohérence il souhaite lui voir appliquer. Le développeur dispose des degrés de cohérence suivants :

Nom	**Signification en écriture**	**Signification en lecture**

ANY	Retourne "Succès" si la requête à été écrite sur au moins un nœud du cluster. Le système se chargera par la suite de répliquer l'écriture sur les autres nœuds.	Inexistant en lecture
ONE	Ce mode se comporte comme le mode ANY mais assure une sécurité additionnelle en garantissant que la requête a été écrite dans la mémoire et le commitlog du nœud. Il s'agit du mode d'écriture par défaut de Cassandra.	Dans ce mode, la donnée est lue sur le premier nœud qui répond. La donnée peut potentiellement ne pas être à jour. Dans tous les cas le système répond immédiatement et s'assure ensuite en tâche de fond que la donnée était à jour afin de répondre correctement à la prochaine requête (système dit **ReadRepair**). Il s'agit du mode d'écriture par défaut de Cassandra.
QUORUM	Dans ce mode, le système retourne "Succès" uniquement une fois que la donnée a été écrite sur la majorité des nœuds sur lesquels elle doit être répliquée.	Requête l'ensemble des nœuds utilisés en réplication et retourne la dernière donnée mise à jour lorsque la majorité des nœuds a répondu. Les autres nœuds utilisés en réplication seront contrôlés en tâche de fond selon le système **ReadRepair**.
LOCAL_QUORUM	Ce mode est analogue au mode QUORUM. Dans un contexte multi-datacenter, elle s'assure uniquement que la donnée a été répliquée sur la majorité des nœuds au sein du datacenter.	Ce mode est analogue au mode QUORUM. Dans un contexte multi-datacenter, la requête est adressée uniquement aux autres nœuds du même datacenter.

EACH_QUORUM	Ce mode est analogue au mode QUORUM. Dans un contexte multi-datacenter, elle s'assure que la donnée a été répliquée sur la majorité des nœuds dans chaque datacenter.	Ce mode est analogue au mode Quorum à la différence que la majorité des nœuds de chaque datacenter doit avoir répondu.
ALL	Dans ce mode, le système répond "Succès" uniquement lorsque la donnée a été répliquée sur l'ensemble des nœuds. Dans ce mode de fonctionnement, si l'un des nœuds ne répond pas, la requête sera en échec.	Requête l'ensemble des nœuds utilisés en réplication et ne répond que lorsqu'il a reçu la réponse de l'ensemble des nœuds. Dans ce mode de fonctionnement, si l'un des nœuds ne répond pas, la requête sera en échec.

Il faut bien avoir à l'esprit que plus le degré de cohérence requis est élevé et plus la requête sera longue à être exécutée. Dans un certains sens, plus le degré de cohérence augmente et plus le système tend à se comporter non plus comme un système AP (Available et Partition tolerant) mais comme un système CP (Consistent et Partition tolerant).

Cette dualité de comportement est très intéressante quoi que probablement un peu complexe à manipuler. Le comportement AP par défaut est satisfaisant dans la majeure partie des cas réels (liste d'articles, description d'un produit, …). Cependant, il existe des cas dans lesquels le besoin d'une donnée fiable est important (par exemple dans le cas de l'interrogation d'un stock au moment d'une validation de commande).

Transactions :

Comme la plupart des outils NoSQL, **Cassandra ne propose aucun mécanisme de transaction**. C'est donc à l'application d'assurer les logiques de retour en arrière en cas d'erreur ou d'interruption d'une procédure multi-étapes. Cela rend un peu plus complexe l'écriture d'applications métiers transactionnelles. Néanmoins, on commence à voir apparaître des système de gestions de transactions distribuées qui s'appuie sur Apache Zookeeper pour fournir ces mécanismes à Cassandra, à l'image du projet prometteur nommé **cages**[9].

Autres considérations :

[9] Projet Cages : http://code.google.com/p/cages/

Cassandra capitalise sur les développements de la fondation Apache et plus spécifiquement Hadoop dont il intègre certains développements dont une implémentation de MapReduce, de Pig et de Hive.

Outils de production

Scalabilité :

Cassandra, de par son architecture complètement décentralisée, permet d'assurer très efficacement aussi bien la distribution des traitements que des données sur des nœuds multiples.

Tolérance à la panne :

De par son architecture, conçue pour ne présenter aucun point de défaillance unique et ses capacités de déploiement sur plusieurs datacenters différents, **Cassandra est un système qui présente une très forte résistance aux pannes**.

Outils de monitoring :

Les outils de monitoring spécifiquement adaptés pour Cassandra ne sont pas très nombreux à ce jour. Il existe cependant un certain nombre de projets communautaires qui permettent le monitoring dans les applications de supervision les plus courantes que sont Nagios, Munin ou Cacti.

Par ailleurs, Cassandra expose l'essentiel des métriques utiles du cluster au monitoring via JMX, rendant possible l'utilisation de l'ensemble des outils de monitoring basée sur JMX.

Gestion des sauvegardes :

La sauvegarde de données distribuées est un sujet complexe pour deux raisons :

- la volumétrie est souvent importante ce qui impose de fragmenter la sauvegarde.
- des problématiques de cohérences de données peuvent apparaître lorsque l'on essaie de sauvegarder chaque nœud individuellement

La meilleure pratique est d'utiliser l'outil nodetool livré avec Cassandra et qui permet de réaliser une image instantanée de chaque nœud tout en garantissant la cohérence de l'ensemble de la sauvegarde. Le fait de réaliser une image de chaque nœud permet de fragmenter la sauvegarde.

Dans un contexte de développement, on pourra utiliser les outils sstable2json et json2sstable qui permettent de réaliser des imports / exports.

Conclusion

Cassandra est un outil relativement complexe. Cette complexité concerne principalement les développeurs qui devront appréhender de nouvelles manières de construire leur base de données. Cela peut apparaitre peu naturel de prime abord car intimement liées au paradigme de base orientée colonnes. Par ailleurs ils devront gérer des paramètres supplémentaires comme le degré de cohérence des requêtes. En outre, la maturité des outils de développement pourra s'avérer rebutante tout du moins au départ. Nous avons par ailleurs constaté que la documentation n'était pas d'une qualité irréprochable.

En contrepartie cet outil présente des caractéristiques très intéressantes. Nous retenons principalement comme avantage :

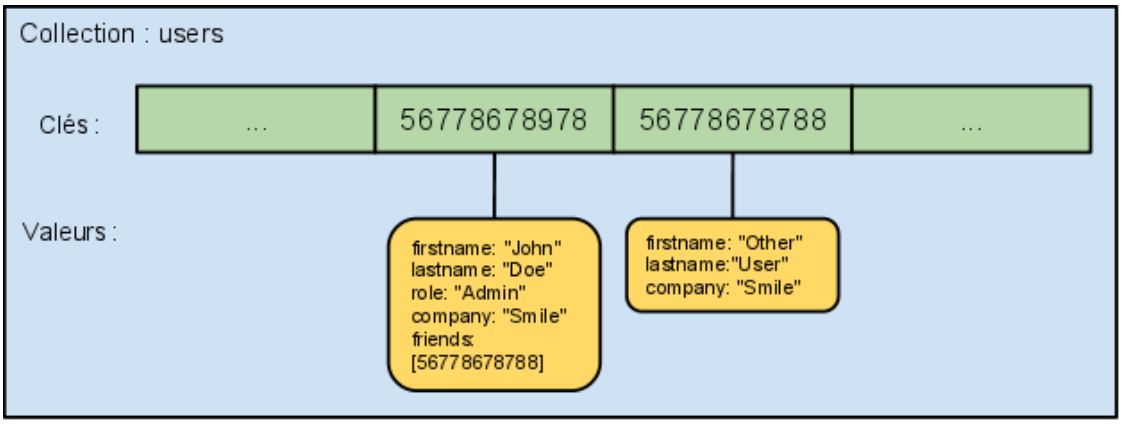
- Des soutiens solides assurant la pérennité de l'outil, avec en chef de file la fondation Apache.
- Un outil éprouvé pour de gros volumes de données par de grands acteurs du Web.
- Un outil conçu pour être exploité de manière entièrement distribuée, y compris entre plusieurs datacenters géographiquement éloignés.
- Une structure de données flexible qui permet de stocker des données hétérogènes
- La capacité pour le développeur de choisir lui même le comportement attendu en privilégiant soit la cohérence, soit la performance. Ce permet à l'outil de s'adapter à des situations assez variées.

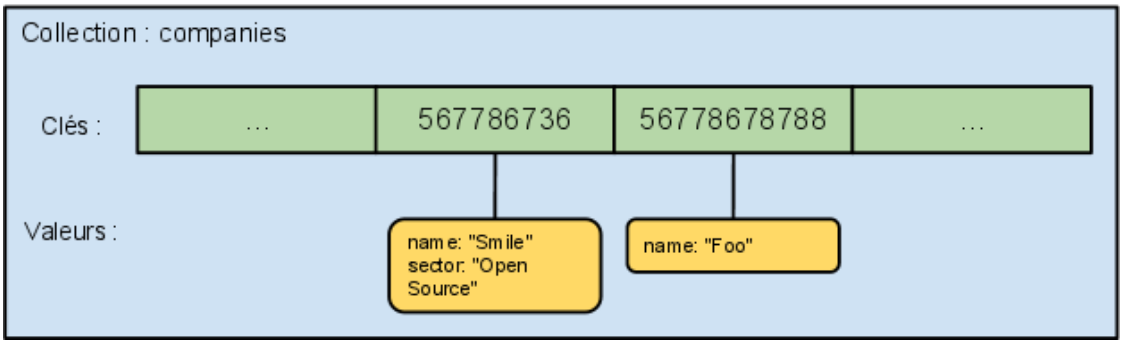
MongoDB

MongoDB est une base de données **orientée document** développée en C++. Le projet dispose d'une popularité importante et aussi d'utilisateurs de renom comme MTV, Disney, Foursquare, Sourceforge, bit.ly, …. Par ailleurs la société 10gen, entité commerciale en charge du développement de MongoDB fournit des services de support et de formation autour de la solution, ce qui peut aider à son déploiement.

Documents sous MongoDB

MongoDB est une base documentaire dans laquelle les documents sont regroupés sous forme de collections, les collections étant l'équivalent des tables du SQL. Il est possible de représenter chaque document au format JSON (MongoDB utilise une variante binaire plus compacte de JSON nommé BSON pour son stockage interne). **Chaque document dispose d'une clé unique** permettant de l'identifier dans la collection.





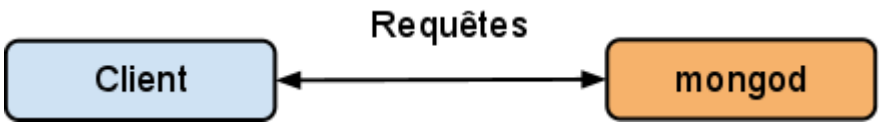
Architecture

Mongo a été conçu pour fonctionner selon plusieurs modes distincts :

- serveur seul
- réplication maitre / esclave
- réplication via Replica Set (ensemble de serveurs traitant les mêmes données)
- partitionnement des données selon leur clé via sharding (partitionnement des données sur plusieurs serveurs)

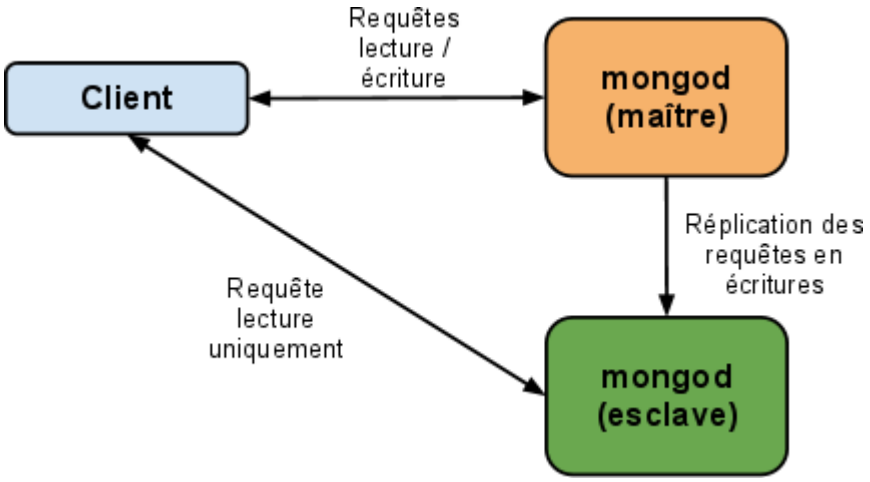
Serveur seul :

Dans ce mode mettant en jeu un seul serveur, un seul processus nommé mongod est utilisé et traite directement les données issues des requêtes du client. L'architecture est alors très simple :



Réplication maître esclave :

Dans le mode maitre / esclave, on lance deux serveurs mongod nommés trivialement 'maître' et 'esclave' :



Les données sont écrites sur le maître uniquement. Le maître réplique l'ensemble des écritures, avec une certaine latence. Le client peut choisir de lire soit sur l'esclave s'il accepte les erreurs liées à la latence de réplication, soit sur le maître dans le cas contraire.

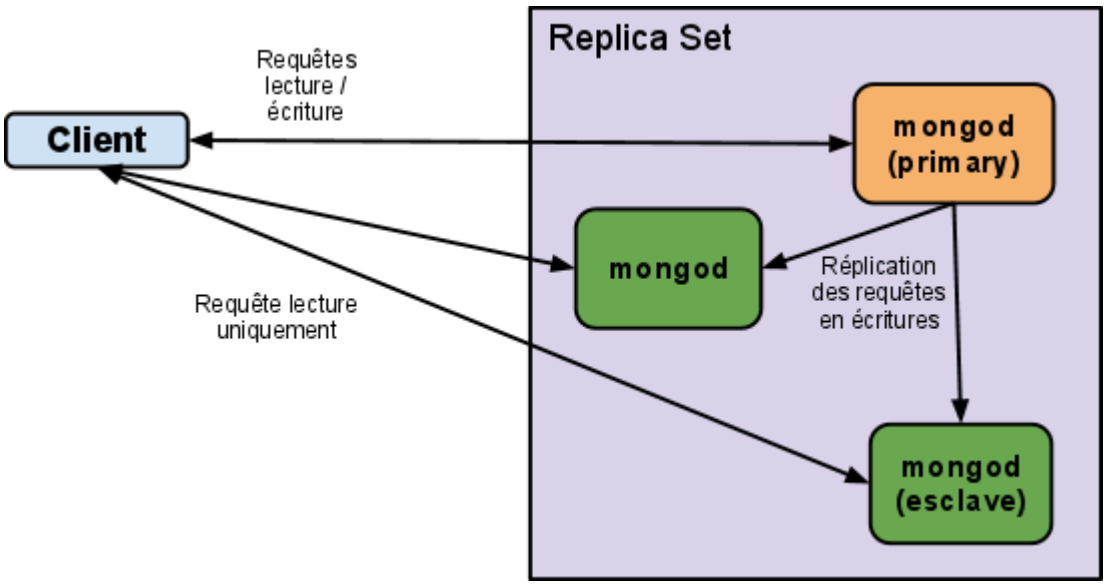
Ce mode de fonctionnement présente comme faiblesse majeure **sa faible tolérance à la panne**, car en cas de panne du serveur maître, l'application cliente est en incapacité d'écrire sur la base ! Il est nécessaire de procéder à un redémarrage **manuel** du serveur esclave en mode maître. Il s'agit alors d'un point de défaillance unique.

Ce mode reste important car il permet aux développeurs de tester les effets de bords liés à la réplication et donc d'assurer que l'application développée sera à même de fonctionner avec de telles contraintes. Cela est d'autant plus vrai que le délai de réplication minimale peut être fixé manuellement afin de simuler une latence réseau allant jusqu'à quelques secondes.

Réplication par Replica Set :

Il s'agit d'un mode de réplication plus avancé que le mode maître / esclave présenté ci-dessus et que l'on peut mettre en place dés lors que l'on dispose d'au moins **trois machines physiques**. Ce mode permet d'éviter la présence d'un point de défaillance unique au sein de l'infrastructure par la méthode suivante :

- On ajoute n serveurs au Replica Set. Chaque serveur dispose d'une priorité.
- Un des serveurs est considéré comme le nœud primaire. On peut voir le rôle de nœud primaire comme celui du serveur maître dans le modèle maître / esclave : il s'agit du nœud qui sera en charge des écritures et des réplications vers les autres serveurs.
- En cas de défaillance du nœud primaire, un nouveau nœud au sein du Replica Set est choisi et devient nœud primaire. Dans le processus d'élection d'un nouveau nœud sa priorité est prise en compte. Cela permet de privilégier certains nœuds, typiquement les plus puissants.



Partitionnement des donnés via sharding :

Les mécanismes de réplication sont intéressants dans la mesure où ils permettent de faire évoluer les performances en lecture sur la base de données. Cependant avec ce système, l'écriture de données repose un seul serveur. Pour pallier à ce problème, il est nécessaire d'utiliser la notion de sharding :

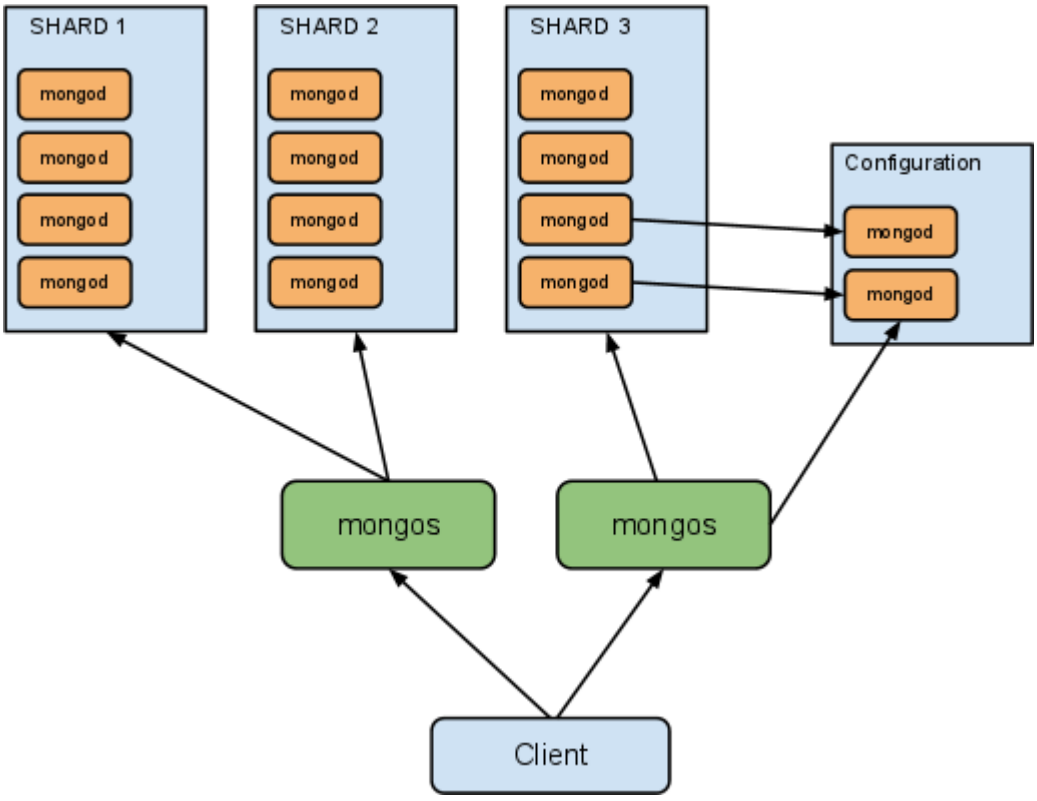
- Un shard contient un ou plusieurs serveurs : un serveur seul, un couple maitre/esclave ou un replica set. L'utilisation d'un replica set est conseillée en production pour une meilleure sécurité.
- Pour partitionner une collection sur plusieurs shards on définit sur celle-ci une shard key. Il s'agit d'une liste de champs du document qui permettront au système de définir sur quel shard un document doit être stocké. Les données seront réparties automatiquement de manière la plus homogène possible entre les différents shards.

Deux éléments d'architecture sont ajoutés :

- Serveur mongos : il s'agit d'un ou de plusieurs serveurs dont le rôle et de router les requêtes du client vers le shard approprié. Bien qu'un seul serveur puisse être utilisé il est conseillé d'en placer au moins deux pour éviter que ce serveur soit un point de défaillance unique.

Par ailleurs, ces serveurs sont susceptibles de recevoir une charge significative. En effet, lors de requêtes faisant intervenir plusieurs shards, ils sont responsables entre autres de l'agrégation des résultats. Notons que ces effets de charge peuvent être limités par un choix approprié des shard keys qui permet aux requêtes de faire intervenir un nombre limité de shards.

- Configuration mongod : Ces serveurs stockent la configuration du sharding et sont utilisés par les autres instances afin d'être en mesure de déterminer sur quel shard les opérations doivent être effectuées. Là encore, il est conseillé d'utiliser plusieurs serveurs afin d'éviter l'apparition d'un point de défaillance unique.



Outils de développement

Stabilité de l'API :

MongoDB est un outil mature dont l'API évolue peu à ce jour. Deux modes d'accès principaux peuvent être utilisés :

- driver spécifique au langage de programmation permettant une communication directement en BSON
- une API REST disponible sous forme d'un serveur HTTP embarqué optionnel

Langage supportés :

Mongo supporte officiellement un grand nombre de langage au travers de drivers. Pour les autres langages, on pourra compter sur un support communautaire assez vaste. Enfin, certains frameworks commencent à implémenter un support de Mongo (sous forme de plugins).

Langage	Support officiel	Support communautaire	Intégration framework
Java	Oui	Oui	Intégration JPA via DataNucleus JPA/JDO, Intégration Spring Data via Spring MongoDB
Python	Oui	Oui	
PHP	Oui	Oui	Intégration avec les framework : CakePHP, Doctrine via le projet Object Document Mapper
Ruby	Oui	Oui	Intégration à Rails via MongoMapper
C++	Oui	Oui	
Perl	Oui	Oui	
C#/.Net	Oui	Oui	
Javascript	Oui	Oui	
Cocoa (via Objective-C)	-	Oui	

Cohérence des données :

Les écritures étant toujours effectuées sur la même machine, la cohérence en écriture est assurée sans aucun problème.

En lecture, la cohérence des données est assurée en partie par le développeur qui peut choisir d'accepter de lire sur un serveur esclave, quitte à ce que celui-ci n'ait pas pu répliquer l'ensemble des données écrites depuis le serveur maître, ou sur le serveur maître.

Transactions :

MongoDB n'implémente pas de mécanisme de transactions.

On pourra néanmoins s'appuyer sur un certain nombre d'opérations atomiques et donc transactionnelles par définition dont on peut trouver la liste sur le site de mongodb.org[10].

Autres considérations :

A l'image de la plupart des solutions NoSQL, MongoDB propose une implémentation de MapReduce, utilisable en Javascript.

Par ailleurs, Mongo implémente un système de fichiers distribués nommé GridFS permettant le stockage de fichiers volumineux (pièce jointes, …).

Outil de production

Scalabilité :

De par sa conception, Mongo est scalable selon deux modes :

- scalabilité en lecture : celle-ci peut être obtenue simplement en utilisant les replica set
- scalabilité en écriture : celle-ci peut être obtenue via les mécanismes de sharding

[10] http://www.mongodb.org/display/DOCS/Atomic+Operations

Tolérance à la panne :

Les mécanismes de replica set permettent au système de résister face à la disparition de l'un de ces nœuds de stockage. Par ailleurs il est possible de placer les nœuds d'un replica set dans plusieurs datacenters (par exemple en assignant au nœud du datacenter de secours, une priorité inférieure) pour obtenir un système qui supporte la perte complète d'un datacenter.

Le système est légèrement plus centralisé que certains autres systèmes SQL lorsqu'on utilise le sharding (présence des nœuds de routage) mais ceux-ci peuvent être dupliqués sans problème, même sur plusieurs datacenters. Cela ne constitue donc pas non plus un point de défaillance unique.

Outils de monitoring / Administration:

Outre le fait que MongoDB dispose de capacités de monitoring basiques embarquées, de nombreux plugins existent permettant l'intégration de MongoDB dans les outils d'analyse les plus populaires du marché comme Munin, Ganglia, Cacti, Zabbix ou Nagios.

Il existe par ailleurs un certain nombre d'outils SaaS de monitoring intégrant des modules spécifiques pour MongoDB (Server Density, CloudKick, AppFirst, …).

Gestion des sauvegardes :

MongoDB dispose d'un outil de sauvegarde nommé mongodump. Celui-ci procède à un export de la base de données qui peut, par la suite, être restauré via mongorestore.

Cet outil n'est pas adapté à des configurations traitant un grand volume de données. En général, celles-ci utilisent des shards contenant un replica set. La technique habituelle pour sauvegarder de telles données est la suivante :

- On stoppe les fonctionnalités de répartition du sharding durant l'opération de sauvegarde
- On stoppe un des serveurs de configuration (les autres continuent à fonctionner pendant ce temps) puis on sauvegarde ces fichiers
- Pour chaque shard, on stoppe l'un des serveurs du replica set que l'on sauvegarde
- On redémarre le serveur de configuration ainsi que les nœuds éteints dans les shards
- On redémarre les fonctionnalités de répartition du sharding

Conclusion

MongoDB est l'un des outils les plus matures de la mouvance NoSQL et dispose par ailleurs du support commercial officiel de son éditeur. Il présente en outre l'avantage de disposer d'un grand nombre de bibliothèques officiellement supportées par son éditeur lui permettant son intégration dans un grand nombre de langages de programmation. On voit par ailleurs apparaître l'intégration de MongoDB dans un nombre croissant de frameworks parmi les plus populaires.

En outre, le paradigme documentaire s'avère assez simple à prendre en main et il est assez aisé de construire des requêtes passablement complexes à l'aide de MongoDB. La documentation de l'API est par ailleurs d'excellente qualité et riche de nombreux exemples concrets.

On regrettera néanmoins les quelques points suivants :

- L'architecture de MongoDB implique de mettre à disposition un nombre important de serveurs dès lors que l'on souhaite distribuer les données, et requiert un temps de configuration important
- Le produit gagnerait à ce que son éditeur communique de manière plus importante au sujet de la roadmap

CouchDB

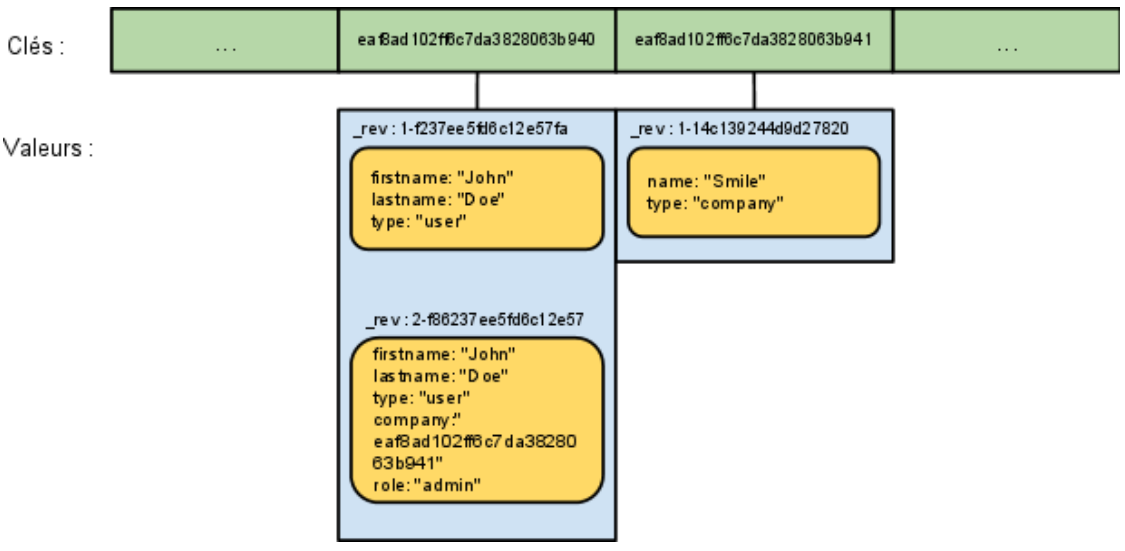
CouchDB est un projet de base de données orientée documents de la fondation Apache. La premiere version stable du projet a été publiée en août 2010 après 5 ans de développement.

Développé en langage Erlang, CouchDB présente certaines fonctionnalités uniques dans les catégories des bases de données documentaires :

- propose un back office qui permet de gérer les bases de données
- propose le versionning documentaire nativement
- entièrement décentralisée, la base permet le développement d'applications offline
- utilisation d'un mécanisme de vues comme système de requête principal

Documents et vues sous CouchDB

Une instance de CouchDB peut accueillir plusieurs bases. Au sein de chaque base, les documents sont stockés dans une unique collection sous la forme clé/valeur. La valeur du document est sauvegardée au format JSON. Pour chaque document, l'ensemble des versions est stockée. Une nouvelle version est créée automatiquement pour chaque modification du document.



L'API HTTP REST de CouchDB est relativement rudimentaire et ne permet pas beaucoup plus que le modèle CRUD des bases de type clés/valeurs. En général, les développeurs CouchDB utilisent le mécanisme de vues fourni par la base afin de réaliser des requêtes. Le système de vue consiste en une requête MapReduce. Généralement, on considère qu'il s'agit du seul moyen d'accès en lecture aux documents sous formes de liste. Il existe deux modes pour les vues :

- Mode document : Les vues sont stockées dans des documents spéciaux appelés document de design. Ces vues disposent de cache et sont très rapides
- Mode temporaire : La vue est interprétée et exécutée à la volée. On réserve ce mode au développement car ses performances sont largement inférieures à celle du mode document

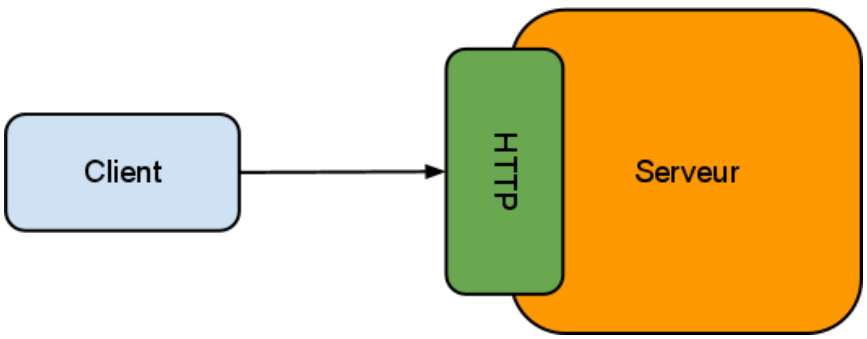
Par défaut, le langage utilisé pour le développement des vues est le Javascript. Il existe des plugins afin de développer dans d'autres langages interprétés (PHP, Ruby, Python, …).

Exemples de vues :

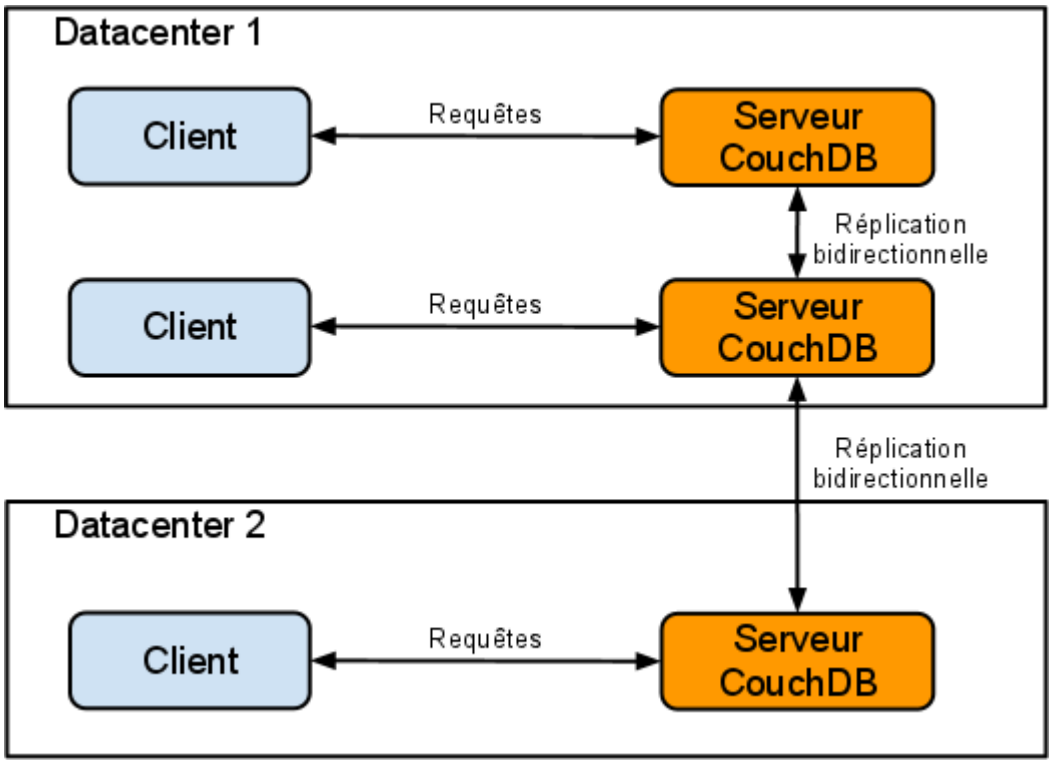
- Sélection des documents dont le champ type est 'product'
- Comptabilisation des documents par type
- Liste des tags et du nombre d'articles associés à chacun des tags
- …

Architecture

Dans sa version la plus simple, une installation de CouchDB se compose d'un unique serveur. Le moteur de stockage embarque son propre serveur HTTP permettant aux clients d'effectuer des requêtes via l'API :

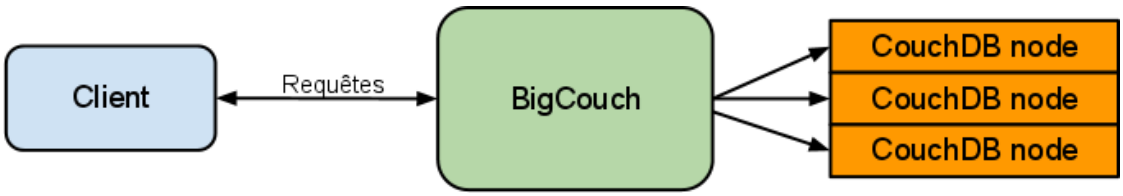


Afin d'augmenter les performances et/ou de permettre à l'application de tourner sur plusieurs sites (typiquement deux datacenters), il est possible de travailler avec le mécanisme de réplication bidirectionnelle de CouchDB. Il permet de synchroniser tous les changements intervenus sur une base vers une autre.



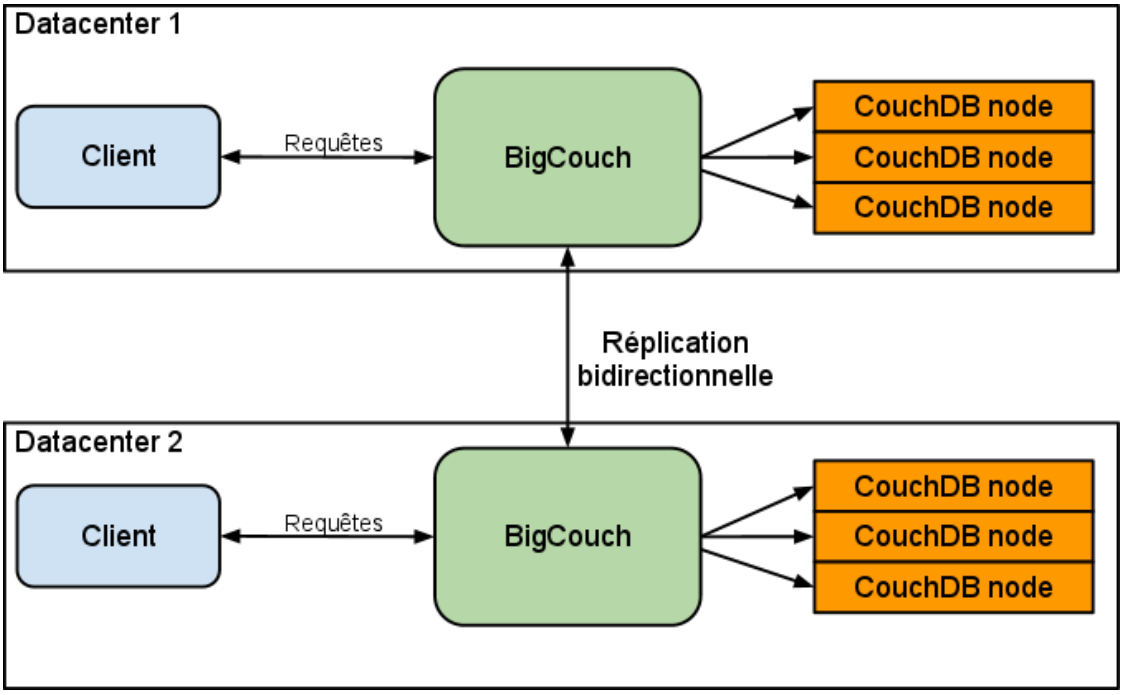
En utilisant le mécanisme de réplication, chaque serveur dispose en permanence de l'ensemble des données du site (moyennant les délais de réplication). Cela permet de distribuer les traitements (lecture, écriture, vues, …). Cette pratique a quand même une limite : la taille de la base est limitée par les capacités de stockage du plus petit serveur de stockage. Aucun mécanisme n'est embarqué par défaut dans CouchDB pour résoudre ce problème. On pourra néanmoins s'appuyer sur l'outil BigCouch dont c'est précisément le rôle.

BigCouch s'insère comme une brique additionnelle devant le client et se comporte comme un proxy qui répartit les requêtes sur différents nœuds CouchDB dont il assure la gestion. Du point de vue de l'application cliente, cet élément additionnel se comporte de manière transparente, l'interface d'accès à BigCouch étant compatible à 100% avec celle de CouchDB.



BigCouch permet par ailleurs la distribution des étapes de mapping sur plusieurs nœuds du cluster, ce qui permet d'accélérer les traitements.

Il est par ailleurs possible de procéder à des réplications bi-directionnelles entre instances de BigCouch afin d'assurer une répartition entre plusieurs datacenters.



Outils de développement

Stabilité de l'API :

On peut considérer l'API de CouchDB comme stable depuis la publication de la première distribution du projet en août 2010. Depuis cette date, on assiste uniquement à des ajouts de nouvelles fonctionnalités.

Langages supportés :

La construction d'une application basée sur CouchDB est très différente de ce qui est habituellement pratiquée avec les autres SGDB. En effet, il est impossible de procéder à des requêtes un tant soit peu complexes sans utiliser le mécanisme de vues de CouchDB. Pour des raisons de performances, il est par ailleurs indispensable de stocker les vues dans les documents de design de la base de données. Ces deux contraintes impliquent que les développeurs procèdent dans un premier temps à l'écriture des vues au sein de la base avant de les appeler en utilisant l'interface HTTP de CouchDB depuis l'application cliente.

Bien qu'il soit possible de procéder à des requêtes HTTP directes depuis les applications clientes, il existe des bibliothèques de plus haut niveau qui simplifient le travail du développeur dans différents langages. Le tableau ci-dessous liste quelques bibliothèques de bonne qualité pour les principaux langages utilisés pour la programmation Web :

Langage	Bibliothèques
Java	Ektorp : une couche de persitance pour CouchDB utilisant JSON jcouchdb : un driver CouchDB pour Spring CouchDB4J : une bibiliothèque permettant la manipulation POJO de documents et de vues CouchDB
Python	couchdbkit : une bibiliothèque permettant notamment le mapping entre documents CouchDB et objets Python. couchdb-python : une autre bibliothèque, plus riche notamment sur le plan de la manipulation des vues
PHP	PHPillow : une bibliothèque permettant le mapping entre des objets PHP et des documents ou des vues CouchDB. Sag : une bibilothèque de plus bas niveau permettant de requêter simplement CouchDB.

Cohérence des données :

Du fait de la conception de la base, les données ne sont pas distribuées sur plusieurs serveurs dans l'architecture de CouchDB. Cela permet au système d'assurer une cohérence forte des données. Cela permet même à CouchDB d'être l'un des seuls SGBD NoSQL à pouvoir respecter les contraintes ACID.

Lors des réplications bidirectionelles, un système de gestion des conflits permet de résoudre les cas pour lesquels un document aurait été mis à jour sur les deux serveurs. Celui-ci s'appuie sur le fait que lors de la mise à jour d'un document les versions antérieures ne sont jamais effacées et que les deux mises à jours concurrentes seront présentes dans deux versions différentes. Le système choisit l'une de ces deux versions comme version courante mais conserve l'autre.

Transactions :

A l'image de la majorité des SGBDs NoSQL, CouchDB n'implémente pas de mécanisme de transaction. Cependant, un certain nombre d'opérations atomiques permettent de pallier en partie ce problème.

Outil de production

Scalabilité :

Comme indiqué dans la section traitant de l'architecture de CouchDB, une installation native de CouchDB propose uniquement une scalabilité concernant les traitements. L'utilisation de BigCouch permet néanmoins d'obtenir la scabilité des données et d'augmenter la distribution des traitements sans requérir de modifier l'applicatif cible.

Tolérance à la panne :

Le système de réplication permet de mettre à disposition plusieurs nœuds CouchDB éventuellement répartis sur plusieurs datacenters. Cela lui confère une résistance élevée à la panne à la condition expresse que chaque base soit répliquée au moins une fois.

Gestion des sauvegardes :

Pour gérer les sauvegardes, on utilise généralement le processus de réplication de CouchDB. Celui-ci est en effet capable de fonctionner sur une base que l'on connecte uniquement lors des opérations de sauvegarde. Il est ensuite possible de procéder à des sauvegardes des fichiers de cette même base.

Conclusion

L'intérêt majeur de CouchDB réside dans l'implémentation d'une logique documentaire radicale. Cela ne va pas sans heurts pour les développeurs qui utilisent pour la première fois cette technologie sur un projet. Il leur faut en effet passer par une phase un peu fastidieuse d'apprentissage du design des vues. Cette approche s'avère néanmoins extrêmement puissante une fois maitrisée. Les outils de manipulation dans les langages de programmation ne sont pas nécessairement les plus aboutis mais restent corrects tant que l'on se cantonne aux langages Web majeurs que sont PHP, Java et Python.

Par ailleurs, CouchDB n'est pas à proprement parler une base distribuée. Il faut en passer par des outils complémentaires pour distribuer les données et les traitements.

CONCLUSION

NoSQL : pour quels usages ?

Nous avons vu que NoSQL était une technologie qui avait été principalement développée dans un contexte où la volumétrie des données rendait indispensable l'utilisation de systèmes distribués pour assurer leur stockage et leur traitement. Il s'agit finalement du cas le plus évident qui pourrait décider de l'utilisation d'une technologie NoSQL dans un projet.

La plupart des moteurs NoSQL apportent par ailleurs une flexibilité plus importante que leurs pendants relationnels en ce qui concerne la gestion de données hétérogènes. Cela entraîne une simplification importante des modélisations qui, lorsqu'elles sont bien menées, peuvent aboutir à des gains considérables en termes de performances pour une application. Le plus souvent cela peut aussi s'avérer bénéfique en termes de maintenabilité sur l'application.

Dans tous les cas, une analyse poussée de l'application doit être menée. Cette analyse doit permettre de répondre à plusieurs questions essentielles.

Existe t-il un problème fondamental de « design » dans mon application ?

Pour une application existante et avant toute chose, il est important de s'assurer par un audit de l'application que celle-ci ne présente pas de défaut de conception entraînant des problèmes de performances importants ou limitant sa capacité à évoluer.

Mon application doit-elle être construite sur une base NoSQL ?

Les technologies NoSQL sont populaires, notamment auprès des développeurs qui sont toujours très enclins à vouloir tester de nouvelles solutions. Mais il se peut que votre application ne soit pas une bonne candidate pour être implémentée sous NoSQL. La plupart du temps, les applications ayant une forte composante transactionnelle ne sont pas de bonnes candidates à une implémentation au dessus d'une base NoSQL, bien qu'il existe des contre-exemples.

Mon application peut-elle intégrer des technologies NoSQL ?

Si votre application souffre de sous-performances qui ne sont pas liées à une erreur de design de la base ; et que celle-ci n'est pas une bonne candidate à un passage en totalité sous une base NoSQL, alors il peut être possible d'optimiser les choses en plaçant uniquement certaines parties de l'application sous NoSQL.

L'application résultante utilisera donc NoSQL uniquement pour les parties qui le requièrent. Cette approche est le plus souvent choisie pour une application existante, pour d'évidentes raisons de coût.

Rappelez-vous toujours que NoSQL ne signifie pas No SQL mais bien Not Only SQL. Dans un certain sens, le mélange des technologies pour obtenir le meilleure de chacune est une approche fondamentalement encouragée par les tenants du NoSQL.

Quel paradigme choisir ?

Il n'existe pas de réponses tout faite à cette question. Cependant, on aura plutôt tendance à privilégier les modèles documentaires de manière assez mécanique pour plusieurs raisons dont la plus évidente est la simplicité de mise en œuvre. Il se peut néanmoins qu'il ne s'agisse pas de la meilleure solution (application d'analyse, BI, ...). S'agissant d'un choix aussi structurant, il convient d'analyser systématiquement les différentes possibilités qui s'offrent à vous.

Quel(s) outil(s) choisir ?

Ce choix ne peut intervenir que lorsqu'on a répondu à l'ensemble des précédentes questions. Outre les caractéristiques purement techniques de la solution retenue qui seront évidemment déterminantes, il est conseillé de prêter une attention importante au temps de montée en compétence des équipes de développement.

NoSQL : les solutions matures

Comme nous l'avons indiqué plus avant, l'essentiel des acteurs de la mouvance NoSQL utilisent des licences open source. Parmi ces produits, les trois solutions que nous décrivons dans ce livre blanc ont particulièrement retenu notre attention et sont aujourd'hui suffisamment matures pour être utilisées en production.

Nous avons particulièrement apprécié MongoDB pour sa maturité, sa simplicité ainsi que son intégration plutôt réussi au sein des langages de programmation. En outre son intégration à différents frameworks nous semble un avantage indéniable en faveur de se produit.

CouchDB semble un peu en retrait par rapport à MongoDB, notamment du fait qu'il ne s'agisse pas nativement d'une technologie distribuée. En revanche, CouchDB offre des perspectives intéressantes en termes de design applicatif.

Cassandra est probablement le produit le moins mature des trois et celui qui sera le plus sujet à des changements majeurs dans le futur. Il s'avère néanmoins celui qui propose le design le plus intéressant en termes de topologies de réseaux. Sa capacité à être déployé de manière complètement distribuée sur plusieurs datacenters en fait un choix incontournable dans les applications qui manipulent des quantités très importantes de données.