RAPPORT TECHNIQUE
PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
DANS LE CADRE DU COURS GTI792 PROJET DE FIN D'ÉTUDES EN GÉNIE DES TI

EXPLORATION DE HADOOP ET HBASE
PERSISTANCE DES DONNÉES D'UNE APPLICATION WEB AVEC UN MODÈLE NON RELATIONNEL

HUGUES LAROCHELLE
LARH19038705

DÉPARTEMENT DE GÉNIE LOGICIEL ET DES TI

Professeur superviseur

Alain April

MONTRÉAL, 13 AVRIL 2011
HIVER 2011

REMERCIEMENTS

Je tiens à remercier Alain April d'avoir accepté de superviser mon projet de fin d’études ainsi que Pier-Luc Caron St-Pierre pour avoir partagé avec moi quelques documents à propos de Hadoop.

EXPLORATION DE HADOOP ET HBASE
PERSISTANCE DES DONNÉES D'UNE APPLICATION WEB AVEC UN MODÈLE NON RELATIONNEL

HUGUES LAROCHELLE
LARH19038705

RÉSUMÉ

Ce rapport cherche à résoudre le problème du manque d'extensibilité horizontale des bases de données relationnelles typiquement utilisées dans un contexte Web. Le volume de données stocké par ce type d'application est toujours grandissant et peut vite devenir une source majeure de difficultés à l'échelle du Web. Dans le but d'éviter de le régler une fois qu'une solution relationnelle est déjà en place, le mandat consiste à proposer, dès le départ, une solution de persistance des données extensible pour une application Web de gestion des liens favoris. Ce rapport s'adresse aux personnes responsables du développement de l'application et à toute personne confrontée à une situation semblable.

Afin de régler ce problème, les possibilités de la plateforme distribuée constituée des technologies Hadoop et HBase sont explorées. Une approche itérative dirigée par le modèle des cas d'utilisation de l'application Web est utilisée. Cette méthode permet de valider rapidement la capacité de ces technologies à se conformer aux exigences de l'application.

La solution proposée inclut une description de l'architecture de production à mettre en place pour déployer la plateforme, une description des classes logicielles de l'interface de programmation (API) pour l'accès aux données ainsi qu'une description de la structure des tables nécessaires pour supporter l'application. Enfin, l'installation d'un environnement de test permet de tester l'implémentation pour valider les choix de conception.

Les tests d'implémentation effectués ont permis de valider les principaux éléments de la conception de la solution réalisés pendant les deux premières itérations du projet. La méthodologie utilisée s'est avérée efficace pour l'exploration des technologies en lien avec les exigences de l'application. Cependant, la plateforme de traitement en lot MapReduce de Hadoop n'a pas encore été testée. Par ailleurs, des technologies complémentaires comme Apache Lucene et les librairies du projet Lily pourraient être utiles pour simplifier le développement de certaines composantes.

Mots clés :
Hadoop, HBase, base de données, extensibilité horizontale, système distribué, NoSQL.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

Aucune figure.

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

Abréviation/sigle/acronyme	**Signification**
API	Application programming interface
HDFS	Hadoop Distributed Filesystem
NoSQL	Non-SQL RDBMS
RDBMS	Relational database management system
SGBD	Système de gestion de base de données
SQL	Structured Query Language
Timestamp	Équivalent français de horadatage ou estampille temporelle
URL	Uniform Resource Locator

LISTE DES SYMBOLES ET UNITÉS DE MESURE

Symbole/unité	**Signification**
To	Téraoctet
Po	Pétaoctet

INTRODUCTION

Dans le cadre du développement d'une nouvelle application Web de gestion des liens favoris, il est nécessaire d'identifier une solution pour l'implantation du système de persistance des données. Le type de système habituellement mis en place pour ce genre d'application consiste en une base de données relationnelle. Cependant, étant donné l'augmentation et l'adoption sans cesse croissantes des applications Web ainsi que de la diversification des méthodes d'accès, les applications Web devenant largement utilisées semblent toutes un jour faire face aux mêmes défis et problèmes. Ceux-ci sont reliés au manque d'extensibilité de leur système de persistance de données (leur base de données relationnelle). En effet, plusieurs études de cas à propos d'applications Web de grande envergure font état de problèmes et de difficultés rencontrés en lien avec la gestion d'une quantité importante de données à l'échelle du Web.

Ces problèmes et difficultés émergent de la nécessité de faire évoluer une base de données relationnelle vers une architecture distribuée. Ce type d'évolution horizontale peut représenter un coût élevé (surtout pour une jeune entreprise), des problèmes reliés au partitionnement et à la réplication des données ainsi qu'à la gestion des transactions. De plus, le nombre élevé de requêtes par seconde et la quantité de données toujours grandissante finissent par faire apparaître des problèmes reliés à la taille des index des tables. Ces effets sont souvent atténués au détriment du respect des bonnes pratiques de conception et des propriétés caractéristiques des bases de données relationnelles. C'est-à-dire qu'il y a une « dénormalisation » des données pour des raisons de performances et que les règles en ce qui a trait à la cohérence des données et des transactions sont assouplies. Cette situation a pour effet que la proportion que finissent par occuper les activités d'opération et de maintenance de tels systèmes représente une trop grande part des activités réalisées par l'équipe. Du même coup, les nouveaux développements de l'application deviennent de plus en plus difficiles et il devient même difficile d'assurer le fonctionnement normal de l'application Web.

Il est donc nécessaire d'identifier la meilleure solution possible pour l'implantation du système de persistance des données de l'application Web dès le début du projet afin de prévenir et de gérer de façon optimale la problématique reliée à l'extensibilité. Un système offrant une bonne capacité d'extensibilité permettra de faire croître l'application, au fur et à mesure que les besoins augmenteront, plus rapidement et plus facilement.

Ce projet a comme objectif d'explorer les technologies *open source* Hadoop et HBase dans le but de démontrer la faisabilité et la pertinence d'utiliser cette technologie comme plateforme pour l'implémentation du système distribué de persistance des données de l'application Web. Pour ce faire, le projet devra fournir des détails sur la technologie, analyser les besoins de l'application Web, proposer et valider une architecture et une conception des différents composants du système ainsi que fournir quelques exemples d'implémentation de code permettant d'obtenir des résultats afin d'évaluer la technologie.

Les résultats du projet permettront d'implanter une solution de persistance des données extensible et peu coûteuse. Cette solution permettra d'assurer que le développement et la croissance de l'application Web puissent être réalisés de façon optimale en tenant compte des expériences passées de plusieurs autres applications Web. Le but est de développer une solution à l'aide d'une technologie ayant déjà fait ses preuves dans des cas réels, et ce, le plus tôt possible dans le processus de développement de l'application.

CHAPITRE 1

REVUE DE LA DOCUMENTATION

Étant donné la nouveauté des technologies Hadoop et HBase et le manque d'expérience avec celles-ci, il est primordial de commencer le projet en recueillant des sources d'informations pertinentes et crédibles à leur sujet. Il est aussi important de trouver des sources d'informations par rapport aux différences avec les systèmes de gestion de base de données relationnelle ainsi qu'à propos d'études de cas démontrant l'utilisation de ces technologies.

1.1 Hadoop

Le projet Hadoop est issu de l'organisme Apache Software Foundation. Tel que décrit sur le site Web du projet, il consiste essentiellement à développer un système de fichier distribué permettant un accès aux données à haut débit et un *framework* de traitement parallèle de type MapReduce fiables et extensibles (Hadoop 01, 2011). Le mélange d'une capacité de stockage sans cesse croissante des disques durs et de la relative stagnation de leur vitesse de lecture et d'écriture mène à un goulot d'étranglement pour le traitement des données et représente une des principales raisons de la nécessité d'une telle technologie (White, 2009, p. 3). Cette technologie sert principalement à stocker et effectuer des traitements par lot d'énormes ensembles de données de l'ordre de téraoctets et de pétaoctets. La technologie est optimisée dans l'optique d'être utilisée sur une grappe de plusieurs serveurs bas de gamme. La compagnie « Yahoo! » affirme utiliser Hadoop sur une grappe composée de 4000 machines (Hadoop 02, 2011).

1.2 HBase

Le projet HBase est un autre projet de l'organisme Apache Software Foundation. Ce projet consiste à développer une base de données distribuée dont la persistance des données s'appuie sur le système de fichier du projet Hadoop (HBase 01, 2011). Le projet a été inspiré d'un article rendu public et présentant le système Bigtable de Google (Chang, 2006). Le

projet vise à répondre à certains besoins non couverts par le projet Hadoop. Il permet notamment l'accès aux données en temps réel, et ce, même s'il s'agit d'accès aléatoires, la manipulation de milliards de petits fichiers et le support de données structurées (George, 2010).

Puisque la plupart des projets qui décident d'opter pour l'utilisation de HBase comme solution de stockage arrivent souvent du paradigme des bases de données relationnelles, certains travaux ont été réalisés afin de faciliter la transformation d'un schéma relationnel vers un schéma non relationnel orienté colonne. Ces tentatives de formalisation ont donné des approches heuristiques et quelques patrons applicables pour la transformation des principaux types de relation d'une base de données relationnelle en schémas fonctionnels pour HBase (Liu, 2009) (Chongxin, 2010) (Schulze, 2010). Par contre, il est clairement énoncé que chaque cas est unique et que l'application de patrons ne peut pas être considérée seule pour produire un schéma efficace (Chongxin, 2010). Par exemple, une conception minutieuse du format de clé d'une table est essentielle et doit tenir compte du type de requêtes visé pour que celles-ci soient réalisables et efficaces (Gray 02, 2009).

D'un côté plus technique et pratique, le projet HBase est développé avec le langage Java et dispose donc d'un API natif Java (HBase 02, 2011). Par ailleurs, l'entreprise Cloudera, qui fournit des produits et services reliés à Hadoop et HBase, rend disponibles gratuitement ses distributions (Cloudera 03, 2011).

1.3 Études de cas

Sur le Web, quelques cas réels d'utilisation du projet HBase pour résoudre des problèmes d'extensibilité ont été rendus publics et expliqués. C'est d'ailleurs le cas de Facebook qui a récemment mis en opération avec succès leur nouveau système de messagerie en temps réel (Hoff 01, 2010) ainsi qu'un nouveau système analytique en temps réel des événements (Hoff 02, 2010). La compagnie StumbleUpon a également donné quelques explications sur l'implantation réussie de HBase pour certaines de leurs opérations (Rawson, 2009).

CHAPITRE 2

MÉTHODOLOGIE DE TRAVAIL

La méthodologie utilisée afin d'atteindre les objectifs du projet consiste à l'exploitation d'une méthode de gestion itérative s'inspirant des recommandations du processus unifié *OpenUP*.

2.1 Déroulement d'une itération

Chaque itération commencera avec la planification de l'itération en cours et se déroulera ensuite par la réalisation des activités suivantes :

- recherche d'informations à propos de la technologie;
- synthèse des informations recueillies;
- analyse des besoins de l'application;
- réalisation de l'architecture de la solution;
- conception des composantes;
- installation d'un environnement de test;
- tests d'implémentation;
- évaluation des résultats obtenus.

Selon quelques lectures préliminaires à propos de la technologie HBase, il est important de connaître les requêtes les plus fréquemment utilisées pour interroger les données dans les tables pour s'assurer d'optimiser l'utilisation du parallélisme offert par cette technologie. De ce fait, la méthodologie utilisée intégrera les principes de développement dirigé par les cas d'utilisation afin de déterminer les exigences fonctionnelles prioritaires et les plus fréquemment utilisées en rédigeant un document de spécification des exigences. De plus, puisque le projet consiste à l'exploration d'une nouvelle technologie, les activités d'architecture et de conception seront réalisées selon les pratiques d'architecture et de conception en évolution. Ceci permettra d'ajuster l'architecture et la conception en fonction des nouvelles connaissances acquises sur la technologie tout au long du projet. Des notes

d'architectures et des artéfacts de conception seront donc réalisés et revus à chaque itération afin de documenter les décisions prises en cours de projet et de s'assurer que ces artéfacts sont maintenus à jour. De plus, les tests réalisés à la fin de chaque itération aideront à valider ces décisions. Ces activités seront toutes exécutées à chaque itération afin de faire progresser le projet par incréments.

2.2 Outils

Les listes ci-dessous énumèrent les principaux outils utilisés pour la réalisation des activités du projet.

Outils pour mettre en place un environnement de test :

- Machine virtuelle (Ubuntu 10.10) sur Oracle VirtualBox
- Distribution CDH3 Beta 3 de Cloudera
  - Hadoop (distribution)
  - HBase (distribution CDH3 Beta 3 de Cloudera)
  - ZooKeeper (distribution CDH3 Beta 3 de Cloudera)

Outils constituant l'environnement de développement pour les tests :

- Windows 7
- Eclipse Helios

Outils pour générer les artéfacts :

- Word
- Powerpoint
- Excel
- Visio

2.3 Livrables et planification

En fonction des objectifs du projet, de la méthodologie et des recommandations expliqués précédemment, cette section décrit les artéfacts nécessaires à l'accomplissement du projet et présente la planification de leur réalisation.

2.3.1 Description des artéfacts

Le Tableau 2.1 établit la liste des artéfacts identifiés et décrit brièvement les objectifs et le contenu de chacun. Ils sont présentés en ordre de réalisation pour une itération typique, mis à part les derniers qui sont des documents réalisés une seule fois pendant le projet (noms indiqués en caractères **gras**).

Tableau 2.1 Description des artéfacts

Nom de l'artéfact	Description
Description de la technologie	Présente une synthèse des éléments importants à propos du fonctionnement de la technologie ainsi que de ses points forts/faibles. Présente la liste des références propres à la technologie.
Modèle du domaine	Illustre sous forme visuelle les éléments composant le domaine d'affaires de l'application Web pour laquelle le système est développé.
Modèle des cas d'utilisation	Illustre sous forme visuelle les cas d'utilisation de l'application Web pour laquelle le système est développé.
Spécification des exigences logicielles	Présente les exigences fonctionnelles et non fonctionnelles du système de stockage de données de l'application Web. Les éléments présentés dans ce document sont : les exigences fonctionnelles, les exigences non fonctionnelles, les contraintes de conception, les interfaces logicielles et les exigences de documentation du système.

Nom de l'artéfact	Description
Schémas d'architecture	Illustre les hypothèses et les explications concernant les décisions d'architecture de la solution. Des notes les expliquant seront contenues directement dans les rapports.
Conception des composants	Illustre la réalisation des composants et des fonctionnalités de la solution (patrons, schémas, diagrammes de classes, diagrammes de table, etc.). Des notes les expliquant seront contenues directement dans les rapports.
Procédure d'installation de l'environnement de test	Explique l'installation et la configuration d'un environnement de test HBase pour la réalisation des tests d'implémentation.
Tests d'implémentation	Code source démontrant de façon concrète l'utilisation de la technologie pour l'accès aux données.
Fiche de renseignements	Décris en quelques lignes le sujet du projet et permets de valider celui-ci.
Proposition de projet	Définis la problématique, les objectifs, la méthodologie, la planification, les risques et les outils du projet.
Rapport d'étape	Reprends la proposition de projet et démontre l'état d'avancement du projet suite à la première itération.
Rapport final	Regroupe la définition du projet, tous les artefacts produits pendant la réalisation du projet, l'analyse des résultats et les recommandations.
Présentation	Présente une synthèse du rapport final sous forme de diapositives en vue de la présentation du projet.

2.3.2 Planification

La planification de la réalisation des activités et des artéfacts est détaillée dans le plan de travail révisé présenté sous forme de tableau en annexe (Tableau-A I).

Le projet débute avec une phase de mise en place pendant laquelle le sujet du projet est défini, accepté et détaillé. Une première étape de recherche de références et de lecture sur la technologie explorée est également réalisée afin de se familiariser avec le contexte de la technologie pour mieux planifier le projet.

Ensuite, deux itérations de durée approximativement égale (la durée varie légèrement afin de s'arrimer aux dates de remise des travaux académiques) s'enchaînent et sont constituées en grande partie des mêmes activités. Ces activités ont déjà été décrites à la section 2.1. Par contre, il y a quand même quelques différences entre les deux itérations. Dans la première itération, un peu plus de temps est accordé à l'installation de l'environnement de test puisqu'il s'agit du même environnement qui sera utilisé dans la deuxième itération. Un peu de temps est quand même prévu pour y apporter quelques ajustements pendant la deuxième itération, si nécessaire. À la fin de la première itération, il est aussi question de la remise du rapport d'étape, alors qu'à la fin de la deuxième itération il s'agit plutôt de la remise et de la revue de l'ébauche du rapport final.

À la fin du projet, une phase de clôture prévoit du temps pour la rédaction et la correction du rapport final et pour la préparation de la présentation. Par ailleurs, toutes les activités qui consistent en une remise ne comptent pas dans le total des efforts estimés.

Depuis la remise du rapport d'étape, certains ajustements ont été apportés au plan de travail. Le total des heures estimées a été ajusté pour se situer à 142 heures suite à l'annulation de la rencontre avec le professeur superviseur à la fin de la deuxième itération, tout simplement en raison du décalage de certaines activités et d'un manque de temps par la suite. Le nombre total d'heures réelles allouées à la fin du projet est de 138 heures. Les modifications apportées au plan de travail sont indiquées en caractères **gras**.

2.4 Risques

Afin de maximiser les chances de réussite du projet, certains risques ont été établits afin qu'ils soient connus, gérés et atténués, si nécessaire. L'échelle utilisée pour la caractérisation de l'impact et de la probabilité de chaque risque est la suivante : faible, modéré et élevé. Le Tableau 2.2 regroupe et décrit les risques ainsi que les méthodes de gestion et d'atténuation appliquées à chacun. Les risques les plus importants en matière d'impact et de probabilité sont identifiés en caractères **gras**.

Tableau 2.2 Identification et évaluation des risques

Risque	Impact	Probabilité	Mitigation / atténuation
Échéancier trop court pour compléter le projet	Élevé	Modéré	Utiliser des outils de gestion/planification et respecter les objectifs du projet. Valider fréquemment le bon déroulement du projet.
Non-disponibilité de matériel (*hardware*) pour tester une architecture distribuée	Modéré	Élevée	Utiliser un environnement de test qui représente le plus possible la réalité tout en gardant à l'esprit que les ressources sont limitées.
Difficultés de mettre en place l'environnement de test	Modéré	Faible	Procéder à l'installation et à des tests de fonctionnement le plus tôt possible.
Demande de changement des exigences	Modéré	Modéré	Faire un suivi continu des exigences en tenant à jour le document de spécification des exigences logicielles et en reflétant ces changements dans l'architecture et la conception à chaque itération.

Risque	Impact	Probabilité	Mitigation / atténuation
Documentation incomplète ou difficile à trouver	Modéré	Élevé	Diversifier les sources d'information, ajouter des références plus théoriques et ajouter des références à propos d'autres technologies semblables.
Manque d'expérience avec la technologie	Modéré	Modéré	Accorder plus de temps à la recherche d'informations et communiquer avec des développeurs de la technologie en cas de besoin.
Manque de maturité de la technologie	Modéré	Modéré	Appuyer les décisions et les hypothèses à l'aide d'études de cas et suivre de près les développements de la technologie.

Les risques reliés à l'échéancier, à la difficulté de tester une architecture réellement distribuée et à documentation incomplète sont très importants étant donné le contexte du projet et de la technologie explorée. Ils restent donc les risques les plus élevés pour le projet.

CHAPITRE 3

PRÉSENTATION DE HADOOP ET HBASE

Puisque le projet est essentiellement axé sur HBase, la présente section introduit les technologies Hadoop et HBase. Les notions importantes à connaître à propos de HBase sont présentées en détail.

3.1 Hadoop

Hadoop est un système distribué qui est composé de deux principaux éléments : un système de fichier distribué (Hadoop Distributed File System - HDFS) et une plateforme de traitement parallèle en lot MapReduce (Hadoop MapReduce).

3.2 HBase

HBase est une base de données non relationnelle distribuée et orientée colonne qui s'appuie sur HDFS. Il s'agit d'une base de données de type NoSQL. C'est donc dire que les données ne sont pas récupérées à l'aide du langage de requête SQL comme c'est le cas avec la plupart des SGBD relationnels. HBase propose essentiellement le même concept de table que le système Bigtable de Google (HBase 01, 2011). L'objectif du projet HBase est de permettre l'utilisation de tables pouvant contenir des milliards de lignes et des millions de colonnes (Gray 01, 2008). Par ailleurs, comme tout système distribué, HBase doit chercher à faire un compromis entre les différentes contraintes de cohérence, de disponibilité et de tolérance au partitionnement (Wikipédia, 2011). HBase garantit surtout le respect des contraintes de cohérence et de disponibilité des données jusqu'à un certain détriment de la tolérance au partitionnement (Stephens, 2009).

3.2.1 Modèle de données

Dans HBase, les données sont stockées dans des tables qui sont composées de lignes (row) et de colonnes (column). Les colonnes appartiennent chacune à une famille de colonnes (column family) particulière. Une cellule (cell) consiste en une version (timestamp) d'une intersection entre une ligne et une colonne et contient une valeur. Lors de la création d'une table, un nombre fixe de familles de colonnes sont spécifiées et leurs paramètres ne peuvent pas changés une fois la table créée, mais il reste possible d'ajouter des nouvelles familles ultérieurement. Les familles de colonnes permettent de structurer les données en regroupant plusieurs colonnes qui partagent les mêmes caractéristiques. De plus, une famille de colonnes peut contenir elle-même une valeur et aucune colonne. Une colonne est définie par sa famille et son nom. En théorie, le nombre de colonnes que peut contenir une famille est illimité. Les colonnes ne sont pas définies lors de la création de la table car le but est que chaque ligne puisse avoir des colonnes différentes et un nombre de colonnes différent. Les colonnes sont créées à la volée par l'application. Les tables HBase permettent donc de contenir des données semi-structurées. La représentation d'une clé, d'une famille, d'une colonne et de la valeur d'une cellule consiste en un tableau d'octets. La version (définie par le timestamp) d'une cellule est quant à elle représentée par un entier de type long.

Dans une table, les lignes sont ordonnées selon l'ordre lexicographique ascendant de leur clé (row key). La ligne dont la clé est la plus petite arrive en premier et la ligne dont la clé est la plus grande arrive en dernier. Le même principe est appliqué pour l'ordonnancement des colonnes dans une famille de colonnes.

Le Tableau 3.1 ci-dessous reprend l'exemple connu de la « webtable» (HBase 03, 2011) afin de donner une représentation visuelle d'une table HBase servant à conserver des informations sur des pages Web.

Tableau 3.1 Vue conceptuelle de la table « webtable»

ROW KEY	TIME STAMP	COLUMN FAMILIES	
		contents:	referer:
org.apache.hadoop	t7		wikipedia.org = "Hadoop"
	t6		nosql-database.org = "Apache Hadoop"
	t2	html = "<html>..."	
	t1	html = "<html>..."	
org.apache.hbase	t8		hadoop.apache.org = "HBase"
	t4	html = "<html>..."	

Dans l'exemple du Tableau 3.1, on observe que le tableau contient deux familles de colonnes : la famille «contents» et la famille «referer». Pour les deux lignes, la famille « contents » contient une seule colonne « html ». Pour la famille « referer », la ligne « org.apache.hadoop » contient les colonnes « wikipedia.org » et « nosql-database.org » alors que la ligne « org.apache.hbase » contient la colonne « hadoop.apache.org ». On peut noter que ces différentes colonnes ont été créées à différents moments dans le temps et que lorsque la valeur d'une colonne change, une nouvelle cellule est créée et l'ancienne version est conservée. Ainsi, lorsque l'on demande de récupérer la valeur de toutes les colonnes pour une ligne sans préciser un intervalle de temps particulier, les valeurs retournées sont les plus récentes.

Il est important de mentionner que la représentation visuelle n'est pas représentative de la structure physique du stockage des données. En effet, dans la représentation visuelle il semble que des cellules vides sont inutilement stockées alors que physiquement, étant donné que HBase est orienté colonne, les cellules sont stockées une seule fois, au moment de leur création. L'expression « nulls are stored for free » est souvent utilisée pour décrire cette méthode (Duxbury, 2008).

3.2.2 Différences avec les bases de données relationnelles

Il existe plusieurs différences entre HBase et les SGBD relationnels les plus connus. Notons, entre autres, que HBase ne dispose pas des caractéristiques suivantes :

- Opérations de jointure entre les tables
- Moteur de requête (comme pour SQL)
- Support natif d'index secondaires (possibilité de le faire soi-même)
- Types de données (toutes les données sont simplement traitées comme une suite d'octets)
- Plusieurs possibilités de tri pour les lignes et les colonnes
- Transactions sur plusieurs lignes et sur plusieurs tables

La question n'est pas de savoir si HBase est meilleur que les autres bases de données, mais plutôt de savoir dans quel (s) contexte (s) il est préférable d'utiliser chacune de ces technologies.

CHAPITRE 4

ANALYSE DE LA PROBLÉMATIQUE

Avant d'effectuer la conception du système de persistance à développer pour l'application Web, il faut commencer par analyser les besoins de l'application. Pour ce faire, les principaux éléments du domaine de l'application doivent être identifiés et analysés. Il est aussi nécessaire d'identifier les cas d'utilisation possibles. Ensuite, à partir de ces informations, les exigences logicielles seront identifiées et expliquées.

4.1 Besoins de l'application

Vu de haut niveau, le besoin de l'application peut être résumé à la nécessité de disposer d'un système de persistance des données très extensible (quantité de données de plusieurs téraoctets et plus), fiable, disponible et permettant de répondre à plusieurs requêtes aléatoires (à l'échelle du Web) en temps réel. La portée de ce projet se limite cependant à l'exploration des technologies Hadoop et HBase pour proposer une solution répondant à ce besoin.

4.2 Domaine de l'application

L'analyse du domaine de l'application permet de préciser et de détailler les différents besoins de l'application. Pour ce faire, il est nécessaire de décrire le domaine, d'élaborer le modèle du domaine et d'élaborer le modèle des cas d'utilisation de l'application.

L'application Web pour laquelle le système est développé est une application permettant à des utilisateurs membres d'enregistrer et de consulter leurs liens Web favoris en ligne. De cette façon, ces utilisateurs peuvent retrouver facilement leurs favoris, en ayant accès à Internet peu importe où ils se trouvent. Le but de l'application est également que les utilisateurs partagent entre eux leurs favoris afin de découvrir des liens intéressants grâce à leur réseau de contacts.

4.2.1 Modèle du domaine

Le modèle du domaine sert à identifier les concepts du domaine de l'application et les liens entre ces différents concepts. Le modèle du domaine de l'application a été réalisé et est présenté dans deux figures distinctes afin de faciliter sa compréhension. L'approche de modélisation en couleur a été utilisée afin de mieux différencier les types de concepts qu'on y retrouve (Coad, 1999). La première figure (Figure-A II-MD01) présente les concepts du domaine en rapport avec le rôle d'un utilisateur enregistré. Il s'agit du rôle qui a le plus de liens avec les autres concepts du domaine. Il a donc été déterminé qu'un annuaire répertorie tous les utilisateurs enregistrés de l'application. Les utilisateurs enregistrés ont chacun un profil qui les décrit et peuvent avoir plusieurs contacts. D'autre part, une liste enregistre tous les favoris créés par les utilisateurs. Un favori peut être classifié par aucune ou plusieurs étiquettes et un répertoire contient toutes les étiquettes utilisées pour classifier les favoris. Lorsqu'un favori est créé, il contient une URL. Une URL peut être utilisée par plusieurs favoris et une liste répertorie toutes les URL qui ont déjà été utilisées par un favori. La deuxième figure (Figure-A III-MD02) du modèle du domaine présente les concepts en lien avec le rôle d'un utilisateur Web (non enregistré). Les concepts illustrés sont les mêmes que ceux de la première figure du modèle, mais on note qu'un utilisateur Web n'a pas de profil et qu'il ne possède aucun favori.

4.2.2 Modèle des cas d'utilisation

Bien que le modèle du domaine permette d'identifier quelques cas possibles d'utilisation, le modèle des cas d'utilisation permet de les identifier plus clairement ainsi que de mieux identifier les différents acteurs pouvant réaliser les cas d'utilisation. Les cas d'utilisation permettent aussi d'identifier les fonctionnalités que doit supporter l'application pour répondre aux besoins de ses utilisateurs. De plus, ils permettent parfois d'identifier quelques concepts du domaine supplémentaires.

Le modèle des cas d'utilisation a été réalisé en trois figures pour faciliter sa lecture (Figure-A IV-DC01, Figure-A V-DC02, Figure-A VI-DC03). Déjà, on note l'identification de

nouveaux rôles (acteurs) par rapport au modèle du domaine. Les acteurs identifiés sont les suivants :

- AC01 – Utilisateur Web
- AC02 – Utilisateur Enregistré
- AC03 – Site Web Externe
- AC04 – Analyste système
- AC05 – Analyste marketing

Le Tableau 4.1 Cas d'utilisation résume les cas d'utilisation identifiés dans le modèle des cas d'utilisation.

Tableau 4.1 Cas d'utilisation

Numéro du cas	**Nom du cas**
CU01	S'inscrire
CU02	Modifier son profil
CU03	S'authentifier
CU04	Se déconnecter
CU05	Ajouter un favori
CU06	Modifier un favori
CU07	Supprimer un favori
CU08	Parcourir ses favoris (par ordre croissant et décroissant de date)
CU09	Rechercher ses favoris par étiquette
CU10	Rechercher des utilisateurs par nom
CU11	Créer une demande de contact
CU12	Accepter une demande de contact
CU13	Consulter sa liste de contacts
CU14	Supprimer un contact
CU15	Consulter les URL les plus populaires
CU16	Consulter les favoris par étiquettes populaires
CU17	Consulter les favoris par recherche d'étiquettes

Numéro du cas	Nom du cas
CU18	Consulter les favoris d'un autre utilisateur (par ordre croissant et décroissant de date)
CU19	Consulter les statistiques d'utilisation du système
CU20	Consulter le statut du système
CU21	Consulter les rapports de journalisation des erreurs
CU22	Consulter les statistiques d'accès aux pages Web
CU23	Consulter les statistiques sociodémographiques des utilisateurs

Afin de mieux identifier quel (s) cas d'utilisation chaque acteur est en mesure de réaliser, le Tableau 4.2 Matrice de participation aux cas d'utilisation identifie sous forme de matrice quels sont les cas d'utilisation auxquels participent les acteurs.

Tableau 4.2 Matrice de participation aux cas d'utilisation

Acteur ► Cas d'utilisation ▼	**AC01**	**AC02**	**AC03**	**AC04**	**AC05**
CU01	X				
CU02		X			
CU03		X			
CU04		X			
CU05		X	X		
CU06		X			
CU07		X			
CU08		X			
CU09		X			
CU10	X	X			
CU11		X			
CU12		X			
CU13		X			

Acteur ► Cas d'utilisation ▼	**AC01**	**AC02**	**AC03**	**AC04**	**AC05**
CU14		X			
CU15	X	X			
CU16	X	X			
CU17	X	X			
CU18	X	X			
CU19				X	
CU20				X	
CU21				X	
CU22					X
CU23					X

Il est important de noter que pour qu'un utilisateur joue le rôle d'un acteur AC02 – Utilisateur Enregistré, il doit avoir déjà été un acteur « AC01 – Utilisateur Web » et doit avoir réalisé avec succès le cas d'utilisation « CU01 – S'inscrire ». Par ailleurs, l'acteur « AC03 – Site Web Externe » participe au cas d'utilisation « CU05 – Ajouter un favori » en permettant d'obtenir automatiquement le titre de l'URL du favori.

4.3 Exigences pour le système de persistance

À partir des concepts et des cas d'utilisation identifiés précédemment, un document de spécification des exigences logicielles (SRS) a été rédigé (Annexe XX - SRS) et établit les exigences à respecter pour le développement du système de persistance de l'application. Le modèle des cas d'utilisation a grandement été utilisé afin d'identifier les exigences fonctionnelles du système qui se retrouve à la section 2 du SRS. Les exigences fonctionnelles seront priorisées un peu plus loin dans le présent rapport. La section 3 du SRS présente les exigences non fonctionnelles qui concernent la convivialité, la sécurité, la disponibilité, la fiabilité et la performance du système. Des critères d'évaluation à respecter ont été énoncés

pour la plupart de ces exigences non fonctionnelles. La seule exigence d'interface du système consiste à fournir une API d'accès aux données écrite en Java à l'application Web. De plus, l'implémentation de cette API doit utiliser l'API Java natif de HBase pour l'accès aux données. Finalement, l'API doit être fonctionnelle dans les environnements Linux et Windows et doit fournir une documentation sous forme de *Javadoc*.

4.4 Priorisation des exigences fonctionnelles

Dans le but de tester le plus rapidement possible les fonctionnalités clés du système de persistance et ainsi minimiser les risques du projet, les exigences fonctionnelles sont priorisées dans le Tableau 4.3 selon leur niveau de criticité.

Tableau 4.3 Priorisation des exigences fonctionnelles selon leur criticité

Numéro de l'exigence	**Ordre de priorité**
EF01	1
EF04	2
EF07	3
EF08	4
EF16	5
EF19	6
EF09	7
EF10	8
EF12	9
EF15	10
EF05	11
EF03	12
EF02	13
EF11	14
EF14	15
EF17	16

Numéro de l'exigence	Ordre de priorité
EF18	17
EF20	18
EF21	19
EF13	20
EF06	21

Les six premières exigences fonctionnelles dans l'ordre de criticité représentent des fonctionnalités clés du système qui ont le plus grand potentiel de valeur ajoutée pour le système. Ces exigences permettront de valider rapidement la faisabilité du projet avec la technologie HBase selon les résultats des tests. Le Tableau 4.4 présente un résumé des six fonctionnalités les plus critiques qui permettront de tester convenablement les possibilités et les limites de la technologie par rapport aux besoins de l'application.

Tableau 4.4 Résumé des exigences fonctionnelles critiques

Numéro	Description
EF01	Le système doit permettre d'ajouter un nouvel utilisateur.
EF04	Le système doit permettre d'ajouter un favori pour un utilisateur.
EF07	Le système doit permettre de récupérer tous les favoris d'un utilisateur (par ordre croissant et décroissant de date).
EF08	Le système doit permettre de récupérer tous les favoris d'un utilisateur correspondant à une étiquette.
EF16	Le système doit permettre de récupérer tous les favoris correspondant à une étiquette.
EF19	Le système doit permettre de générer des rapports de journalisation des erreurs.

Puisque la technologie HBase n'offre pas des fonctionnalités d'indexation comme on retrouve dans la plupart des SGBD relationnels, la capacité à respecter les exigences EF07,

EF08 et EF16 sera déterminante pour l'adoption, ou non, de la technologie pour les fonctionnalités reliées aux favoris.

CHAPITRE 5

CONCEPTION DE LA SOLUTION

À partir des exigences logicielles identifiées à la section 4.3, il est possible de concevoir une première version des principales composantes du système de persistance. Ce chapitre décrit la philosophie, les contraintes, les justifications ainsi que tous les éléments significatifs qui ont permis d'orienter les choix de conception et d'implémentation des composantes. Des détails sont donnés concernant l'architecture du système, la structure de classes logicielles de l'API ainsi que la structure des tables impliquées.

5.1 Architecture de haut niveau

Deux types d'architecture sont nécessaires à la réalisation du projet. En effet, une architecture de test pour le développement et une architecture de production doivent être proposées afin de faciliter leur compréhension, leur mise en place et leur explication. Il est important de noter que HBase et Hadoop limitent les choix d'architecture possibles étant donné les contraintes du fonctionnement de leur propre architecture.

5.1.1 Architecture de test

L'architecture de test du système de persistance doit pouvoir être déployée à même le poste de travail d'un développeur. Cette contrainte signifie également que cette architecture doit fonctionner sur un seul ordinateur. La technologie HBase bénéficie justement de plusieurs modes de fonctionnement dont le mode pseudo distribué. Ce mode permet d'exécuter toutes les composantes nécessaires à un environnement de test dans des machines virtuelles Java distinctes sur un même ordinateur. La figure (Figure-A VII-AR01) illustre le déploiement d'une telle architecture. Sur la figure, on observe que les composantes essentielles de Hadoop, HBase et ZooKeeper sont utilisées. Dans le cadre d'activités de développements, cette architecture permet amplement de valider le fonctionnement du code. Cependant, cette

architecture ne peut pas être considérée pour effectuer des tests de performance puisqu'elle n'est pas représentative d'une architecture de production réellement distribuée.

5.1.2 Architecture de production

L'architecture de production de la figure (Figure-A VIII-AR02) illustre les composantes et mécanismes essentiels que le système de persistance distribué devra comporter pour assurer les exigences en termes de disponibilité, fiabilité et intégrité. Les objectifs de l'architecture sont d'assurer une haute disponibilité du système et l'intégrité des données en tout temps. Toutes les composantes du système doivent avoir au moins un niveau de redondance double afin de garantir un tel taux de disponibilité.

On remarque d'abord que le système devra supporter le fait que plusieurs instances de serveur/application Web doivent exécuter leur propre instance de l'API d'accès aux données. L'accès à la plateforme HBase se fait habituellement par l'entremise d'une simple synchronisation avec l'instance HMaster. Cependant, afin d'assurer un taux de disponibilité élevé, l'architecture propose d'utiliser une configuration active/passive d'au moins deux instances HMaster. Pour que l'API d'accès aux données puisse savoir quelle instance HMaster est actuellement active, elle devra faire une requête au quorum ZooKeeper (étiquette 1) qui lui s'occupe d'élire l'instance HMaster active. Une fois connue, l'API établit une communication avec l'instance HMaster (étiquette 2). Ensuite, l'API récupère les informations des instances HRegionServer que l'instance HMaster détient déjà. À partir de ce moment là, l'API conserve dans une cache ces informations pendant un certain temps et peut les utiliser pour effectuer des requêtes aux différentes instances HRegionServer (étiquette 3).

Concernant Hadoop, il sera nécessaire de mettre en place une instance BackupNode pour assurer une bonne disponibilité du système de fichier (non illustré sur la figure).

5.2 Structure de classes logicielles de l'API

La structure de classes logicielles de l'API d'accès aux données doit permettre d'effectuer toutes les opérations nécessaires au bon fonctionnement de l'application Web. De plus, elle doit permettre d'éviter de coupler directement les classes logicielles de l'application Web avec l'API natif de HBase, ceci dans un objectif de maintenabilité.

5.2.1 Détails des classes d'accès aux données

Les détails de quelques classes logicielles de l'API d'accès aux données qui seront utilisées par l'application Web sont illustrés dans la Figure-A IX-CL01. Les interfaces « BookmarksDao » et « UsersDao » permettent de découpler le code client de l'implémentation utilisée pour l'accès aux données. La classe DaoFactory permet au code client d'obtenir des instances des objets « Dao » sans savoir comment elles ont été créées. Par ailleurs, l'instanciation de la classe « HTable » de l'API HBase s'occupe déjà de partager les ressources avec d'autres instances si on lui passe en référence une instance de configuration déjà utilisée. C'est pourquoi la classe « SharedConfiguration » dispose d'une méthode statique permettant de récupérer une instance unique de configuration.

5.2.2 Intégration avec l'application Web

La Figure-A X-CL02 illustre comment l'API d'accès aux données sera utilisée par l'application Web. Tout d'abord, les objets de l'application Web utiliseront les méthodes de la classe « DaoFactory » pour obtenir des instances des objets « Dao » (via les interfaces UsersDao ou BookmarksDao). Ainsi, les objets de l'application Web n'ont aucune idée des détails de création des objets d'accès aux données.

5.3 Structure des tables

Afin de se conformer aux exigences fonctionnelles du système, les structures des tables doivent être soigneusement établies afin de profiter au maximum des propriétés du modèle de

données de HBase. Il faut particulièrement bien choisir le format des clés de ligne (row key) pour que les données soient le mieux réparties possible sur les HRegionServer et les HRegion à travers le temps tout en tenant compte de la nécessité de la proximité de certaines lignes pour optimiser leur accès en séquence selon les patrons des requêtes. Les structures des tables « users », « bookmarks » et « bookmarks_tags_index » sont décrites dans la suite de cette section.

Il est important de noter que dans tous les tableaux référencés dans cette section pour présenter les structures des tables, la notion de *timestamp* des cellules a été ignorée dans le but de simplifier leur lecture.

5.3.1 Table « users »

Pour les utilisateurs, il est nécessaire de stocker leur courriel, leur mot de passe et leur nom d'usager. De plus, il doit être possible de récupérer tous leurs favoris en ordre croissant et décroissant de date. Il faut aussi pouvoir récupérer la liste de toutes les étiquettes appliquées aux favoris d'un utilisateur. Finalement, il faut aussi pouvoir récupérer la liste des contacts d'un utilisateur. Le Tableau-A XI-TB01 montre la structure proposée pour la table « users » afin de répondre aux exigences fonctionnelles.

La clé de ligne utilisée pour cette table est le nom de l'utilisateur. Cette clé permet de pouvoir récupérer facilement la ligne correspondant à un usager et permettra aussi de distribuer relativement uniformément les lignes dans la table. Une famille de colonne « info » contient toujours les colonnes « user », « email », « password » et « salt », car il s'agit de données obligatoires pour créer un nouvel utilisateur. Les familles de colonnes « bookmarks_ts_desc » et « bookmarks_ts_asc » contiennent respectivement des colonnes dont leur identifiant et leur valeur permettent de garder une référence aux favoris de l'utilisateur en ordre descendant et ascendant de date. La famille de colonnes « tags » contient des colonnes dont leur identifiant correspond aux étiquettes utilisées par l'utilisateur pour classifier tous ses favoris. L'ordre lexicographique appliqué aux identifiants de colonnes permettra de les récupérer dans l'ordre alphabétique. Enfin, la famille de colonnes

« contacts » contient des colonnes dont leur identifiant et leur valeur permettent de garder une référence aux contacts (et des statuts des contacts) d'un utilisateur.

5.3.2 Table « bookmarks »

Pour les favoris, il est nécessaire de stocker le nom de l'usager, le *timestamp*, le *timestamp* inverse, l'URL, un titre, des notes et les étiquettes utilisées pour les classifier. Le Tableau-A XII-TB02 montre la structure proposée pour la table « bookmarks » afin de répondre aux exigences fonctionnelles.

La clé de ligne utilisée est une clé composite. C'est-à-dire qu'elle est composée du nom de l'usager suivi du *timestamp* inverse. Chaque favori est donc stocké dans une ligne selon l'utilisateur qui l'a créé et la date à laquelle il a été créé. Cette clé permet donc de distribuer relativement uniformément les lignes dans la table. De plus, elle permet de récupérer de façon séquentielle tous les favoris d'un utilisateur en ordre descendant de date. La famille de colonnes « index » contient toujours les colonnes « user », « ts » et « rts » pour récupérer de façon simple l'utilisateur, le *timestamp* et le *timestamp* inverse d'un favori. La famille de colonnes « meta » contient toujours les colonnes « url », « title » et « notes » pour stocker les informations du favori. Enfin, la famille de colonnes « tags » contient des colonnes dont leur identifiant correspond aux étiquettes utilisées par l'utilisateur pour classifier le favori.

5.3.3 Table « bookmarks_tags_index »

Afin de pouvoir retrouver tous les favoris de tous les utilisateurs selon une étiquette, il a été nécessaire de créer une table d'index (Chin, 2011). Le Tableau-A XIII-TB03 montre la structure proposée pour la table « bookmarks_tags_index » afin de répondre à cette exigence fonctionnelle.

La clé de ligne de cette table est une clé composée de la valeur d'une étiquette suivie du *timestamp* inverse du favori indexé. Cette clé permet donc de distribuer relativement uniformément les lignes dans la table et de récupérer séquentiellement tous les favoris

classifiés selon une étiquette en ordre descendant de date. La famille de colonnes « index » contient toujours une colonne « bookmark » qui contient la clé de ligne du favori indexé.

5.3.4 Exemple de données dans les tables

Afin de mieux comprendre la mécanique des structures des différentes tables expliquée précédemment, les Tableaux-A XIV-TB04 présentent un exemple avec des données potentielles.

CHAPITRE 6

INSTALLATION D'UN ENVIRONNEMENT DE TEST

Étant donné le manque d'expérience avec la technologie, il est important de pouvoir tester rapidement le bon fonctionnement de certains mécanismes reliés aux structures des tables proposées pour HBase. Pour ce faire, un environnement de test doit être installé sur un poste de travail selon l'architecture de test décrite à la section 5.1.1. Une installation a donc été réalisée afin de définir une procédure d'installation pouvant être réutilisée à l'avenir.

6.1 Pré-requis

L'installation des composantes Hadoop, HBase et ZooKeeper a été effectuée sur un poste de travail Windows 7 dans une machine virtuelle gérée par le logiciel Oracle VM VirtualBox. Cette machine virtuelle est l'hôte d'un système d'exploitation Linux Ubuntu (maverick 10.10).

Certaines composantes logicielles doivent être installées pour que la plateforme Hadoop/HBase puisse fonctionner. Ces composantes sont les suivantes :

- Java 1.6
- CURL
- SSH
- RSYNC

L'annexe XV-PR01 détaille les commandes à utiliser pour installer ces composantes sur un système Ubuntu.

6.2 Choix d'une distribution Hadoop/HBase

Après quelques tentatives manuelles infructueuses d'installation de Hadoop et HBase sur le système Ubuntu, il a été décidé d'utiliser une distribution de la compagnie Cloudera

permettant d'installer facilement toutes les composantes nécessaires. Les distributions de cette compagnie sont émises sous la même licence que Hadoop, HBase et ZooKeeper et sont disponibles gratuitement. La distribution CDH3 Beta 3 a été installée et configurée en mode pseudo distribué selon la documentation de la compagnie (Cloudera 01, 2011) (Cloudera 02, 2011). L'annexe XVI-PR02 résume de façon condensée toutes les commandes exécutées lors de l'installation. Il a été très facile d'installer les composantes à l'aide de cette distribution. De plus, la configuration pour le mode pseudo distribué n'a qu'à être téléchargée pour rendre l'installation fonctionnelle.

CHAPITRE 7

TESTS D'IMPLÉMENTATION

Afin de mettre en pratique et de valider rapidement le fonctionnement de l'API Java de HBase et la conception des tables, des tests d'implémentation ont été réalisés. Ces tests ne couvrent pas toutes les classes logicielles et les opérations définies jusqu'ici, mais permettent simplement de valider la faisabilité de certains mécanismes essentiels au succès du projet.

7.1 Implémentation du script de création des tables

Le script de l'annexe XVII-SC01 sert à créer la structure des tables définies lors de la conception. Ce test a été facile à réaliser et le résultat obtenu est satisfaisant puisqu'il a permis de passer au test suivant.

7.2 Implémentation de classes d'accès au données

Les scripts des annexes XVIII-SC02 et XIX-SC03 démontrent l'utilisation de l'API Java de HBase pour insérer et récupérer des données dans les tables. Les premiers résultats obtenus sont satisfaisants puisqu'ils répondent aux exigences en termes de fonctionnalités pour les cas implémentés.

On observe, entre autres, qu'avec l'implémentation d'une couche d'accès aux données, il est possible de retourner des résultats selon un ordre croissant ou décroissant à partir d'un « scan » séquentiel même si les tables de HBase possèdent un seul ordre de tri. En effet, l'implémentation de la méthode « getBookmarks(String user, int limit, int offset, String sort) » dans la classe « HbaseBookmarksDao » le prouve en utilisant une liste dans laquelle il est possible d'y ajouter des éléments à la fin ou au début selon l'ordre désiré.

CHAPITRE 8

INTERPRÉTATION DES RÉSULTATS ET DISCUSSION

L'architecture de test a pu être déployée avec succès sur un poste de test grâce à l'utilisation d'une distribution de l'entreprise Cloudera. Cette distribution simplifie beaucoup la mise en place et la configuration d'une architecture Hadoop/HBase.

D'autre part, les tests qui ont été effectués permettent de valider une partie de la conception effectuée jusqu'à maintenant. Effectivement, les tests ont permis de répondre à quelques-unes des exigences fonctionnelles critiques identifiées dans le Tableau 4.3. Les tests ont permis de valider la conception de la structure des tables et des objets d'accès aux données pour les exigences EF01, EF04 et EF07. De plus, même en raison du manque de temps pour tester davantage de fonctionnalités pendant la deuxième itération, une brève analyse permet de dire que les exigences EF08 et EF16 devraient être facilement réalisables. Finalement, aucun test n'a encore été effectué avec la plateforme MapReduce de Hadoop pour générer des rapports de journalisation des erreurs. L'exigence EF19 concernée est pourtant une exigence jugée prioritaire.

Pendant le projet, certaines difficultés ont été rencontrées. La première difficulté consiste à l'impossibilité de tester l'architecture de production en raison du manque de ressources matérielles pour y arriver. La deuxième difficulté était de concevoir la structure des tables. En effet, leur conception a été corrigée et réévaluée plusieurs fois avant d'en arriver à un résultat satisfaisant.

Finalement, quelques améliorations pourraient être apportées aux tests réalisés. Par exemple, le script de création des tables pourrait être peaufiné en spécifiant des paramètres plus spécifiques pour la configuration des familles de colonnes que ceux par défaut.

CONCLUSION

En conclusion, le projet avait comme objectif principal de proposer une solution au problème d'extensibilité horizontal des SGBD relationnel utilisés dans un contexte Web. La solution proposée devait utiliser les technologies Hadoop et HBase afin de définir un système de persistance des données répondant aux besoins de l'application. Pour ce faire, une approche itérative inspirée des méthodes agiles proposées par le processus unifié OpenUP a été utilisée afin de gérer le développement du projet. Le développement du projet était principalement dirigé à l'aide du modèle des cas d'utilisation de l'application.

Pendant les deux itérations réalisées, plusieurs activités ont mené à l'avancement du projet. Premièrement, une recherche d'informations à propos des technologies Hadoop et HBase a permis de produire une synthèse des éléments d'information les plus importants dans le but d'augmenter le niveau de connaissance de ces technologies. Ensuite, l'analyse du domaine de l'application Web a permis de produire un modèle du domaine et un modèle des cas d'utilisation afin de formaliser les concepts clés. Ces informations ont permis de rédiger un document de spécification des exigences logicielles concernant le système de persistance à développer. Les exigences ainsi identifiées ont été priorisées afin de mieux orienter les efforts de développement. Elles concernent les fonctionnalités à développer, les critères de qualité à respecter ainsi que les caractéristiques des interfaces avec d'autres systèmes et de la documentation nécessaire.

Les différentes activités de conception ont permis d'établir une architecture de test et une architecture de production pour le système. De plus, la structure des classes de l'API d'accès aux données a été définie ainsi que celle des tables qui seront utilisées. Une bonne partie de la conception réalisée jusqu'à maintenant a pu être validée suite à l'installation d'un environnement de test à l'aide d'une distribution gratuite de l'entreprise Cloudera. Quelques implémentations simples de code ont en effet permis de tester avec succès les exigences les plus prioritaires.

Les différentes informations trouvées à propos des technologies explorées et les résultats obtenus suite aux deux itérations réalisées permettent de faire quelques recommandations à propos du projet.

RECOMMANDATIONS

Premièrement, la méthodologie utilisée jusqu'à maintenant s'est avérée être très efficace et il est donc conseillé de conserver la même approche pour les prochaines itérations. Deuxièmement, il serait très important de tester rapidement les possibilités qu'offre la plateforme MapReduce pour évaluer la possibilité de respecter l'exigence consistant à générer des rapports de journalisation des erreurs. Troisièmement, la distribution de l'entreprise Cloudera utilisée pour mettre en place l'environnement de test permet d'éviter de perdre du temps à essayer de faire fonctionner l'architecture de test et de se concentrer sur la réalisation de tests proprement dits. Quatrièmement, vu les moyens importants nécessaires pour arriver à mettre en place l'architecture de production, il est recommandé d'attendre un avancement un peu plus important au niveau de la conception et de sa validation par des tests avant d'emboîter le pas. Ceci dans le but de limiter les risques reliés au manque d'expérience avec les technologies impliquées. Cinquièmement et finalement, il serait intéressant d'explorer des technologies complémentaires offertes par les projets Apache Lucene et Lily qui semble faciliter la création et la manipulation d'indexes secondaires dans HBase.

LISTE DE RÉFÉRENCES

Chang, F.; Dean, J.; Ghemawat, S.; Hsieh, W. C.; Wallach, D. A.; Burrows, M.; Chandra, T.; Fikes, A.; Gruber, R. E.;. 2006. « Bigtable: A Distributed Storage System for Structured Data ». In *OSDI'06: Seventh Symposium on Operating System Design and Implementation*. Seattle, WA. <http://labs.google.com/papers/bigtable.html>.

Chin, J.; Wang, Z.;. 2011. « HBase A Comprehensive Introduction ». In *Le site Web du cours CSCI 2270: Advanced Topics in Database Management*. <http://cs.brown.edu/courses/cs227/slides/mar14-hbase.pdf>. Consulté le 9 avril 2011.

Chongxin, Li;. 2010. « Transforming relational database into HBase: A case study ». In *Software Engineering and Service Sciences (ICSESS), 2010 IEEE International Conference on* (16-18 July 2010). p. 683-687.

Cloudera 01, ;. 2011. « CDH3 Installation ». In *Le site Web de l'entreprise Cloudera*. <https://docs.cloudera.com/display/DOC/CDH3+Installation#CDH3Installation-InstallingCDH3onUbuntuSystems>. Consulté le 28 février 2011.

Cloudera 02, ;. 2011. « CDH3 Deployment in Pseudo-Distributed Mode ». In *Le site Web de l'entreprise Cloudera*. <https://docs.cloudera.com/display/DOC/CDH3+Deployment+in+Pseudo-Distributed+Mode>. Consulté le 28 février 2011.

Cloudera 03, ;. 2011. « Cloudera's Distribution including Apache Hadoop ». In *Le site Web de l'entreprise Cloudera*. <http://www.cloudera.com/hadoop/>. Consulté le 9 avril 2011.

Coad, P.; Lefebvre, E.; De Luca, J.; . 1999. *Java Modeling In Color With UML : Enterprise Components and Process*. Prentice Hall PTR, 229 p.

Duxbury, Bryan;. 2008. « Matching Impedance: When to use HBase ». In *Le blogue du site Web RapLeaf*. <http://blog.rapleaf.com/dev/2008/03/11/matching-impedance-when-to-use-hbase/>. Consulté le 9 avril 2011.

George, L.;. 2010. « Advanced HBase ». In *Navteq Architect Summit*. <http://www.docstoc.com/docs/document-preview.aspx?doc_id=66356954>.

Gray 01, J.;. 2008. « Hadoop/HBase vs. RDBMS ». In *CTO Forum*. Los Angeles, CA. <http://www.docstoc.com/docs/2996433/Hadoop-and-HBase-vs-RDBMS>.

Gray 02, J.;. 2009. « HBase 101: Row key design for paging (LIMIT, OFFSET) queries ». In *Le site Web Streamy Development Blog*. <http://devblog.streamy.com/2009/04/23/hbase-row-key-design-for-paging-limit-offset-queries/>.

Hadoop 01, The Apache Software Foundation. 2011. « Welcome to Apache Hadoop! ». In *Le site Web du projet Hadoop*. <http://hadoop.apache.org/>. Consulté le 15 janvier 2011.

Hadoop 02, The Apache Software Foundation;. 2011. « PoweredBy - Hadoop Wiki ». In *Le wiki du projet Hadoop*. <http://wiki.apache.org/hadoop/PoweredBy#Y>. Consulté le 9 avril 2011.

HBase 01, The Apache Software Foundation. 2011. « This is Apache HBase ». In *Le site Web du projet HBase*. <http://hbase.apache.org/>. Consulté le 6 avril 2011.

HBase 02, The Apache Software Foundation. 2011. « HBase API ». In *Le site Web du projet HBase*. <http://hbase.apache.org/apidocs/index.html>.

HBase 03, The Apache Software Foundation. 2011. « The Apache HBase Book ». In *Le site Web du projet HBase*. <http://hbase.apache.org/book.html>. Consulté le 9 avril 2011.

Hoff 01, T.;. 2010. « Facebook's New Real-Time Messaging System: HBase To Store 135+ Billion Messages A Month ». In *Le site Web High Scalability*. <http://highscalability.com/blog/2010/11/16/facebooks-new-real-time-messaging-system-hbase-to-store-135.html>.

Hoff 02, T.;. 2010. « Facebook's New Realtime Analytics System: HBase To Process 20 Billion Events Per Day ». In *Le site Web High Scalability*. <http://highscalability.com/blog/2011/3/22/facebooks-new-realtime-analytics-system-hbase-to-process-20.html>.

Liu, E.;. 2009. « Hbase Schema Design Case Studies ». In *Le site slideshare*. <http://www.slideshare.net/hmisty/20090713-hbase-schema-design-case-studies>.

Rawson, R.;. 2009. « HBase ». In *NoSQL Meetup* (11 juin 2009). San Francisco, CA. <http://static.last.fm/johan/nosql-20090611/hbase_nosql.pdf>.

Schulze, B.;. 2010. « Successfully used Patterns in application and table design with Hbase ». In *Apache Hadoop Get Together*. Berlin. <http://isabel-drost.de/hadoop/slides/ecircle.pdf>.

Stephens, Bradford;. 2009. « HBase vs. Cassandra: NoSQL Battle! ». In *Le site Web Road to Failure*. <http://www.roadtofailure.com/2009/10/29/hbase-vs-cassandra-nosql-battle/>. Consulté le 9 avril 2011.

White, T. 2009. *Hadoop : The Definitive Guide*. O'Reilley.

Wikipédia. 2011. « Théorème CAP ». In *Le site Web Wikipédia*. <http://fr.wikipedia.org/wiki/Th%C3%A9or%C3%A8me_CAP>. Consulté le 9 avril 2011.