



Bibha Gupta

# Development of a System for Recommendation and Search via Cloud Computing

Helsinki Metropolia University of Applied Sciences  
Master of Engineering  
Information Technology – Mobile Programming  
Thesis  
5 March 2011

Author(s) Title	Bibha Gupta Development of a System for Recommendation and Search via Cloud Computing
Number of Pages Date	70 pages 5 March 2011
Degree	Master of Engineering
Degree Programme	Information Technology – Mobile Programming
Specialisation option	Mobile Programming
Instructor(s)	Kitsnik Peeter, Ph.D, Senior Lecturer
<p>Cloud computing is one of the emerging technologies. This thesis aimed to outline cloud computing and its features. The thesis considered cloud computing for machine learning and data mining. The goal of the thesis was to develop a recommendation and search system using cloud computing. The main focus was on the study and understanding of Hadoop, one of the new technologies used in the cloud for scalable batch processing, and HBase data model which is a scalable database on top of the Hadoop file system.</p> <p>The thesis project involved the design, analysis and implementation phases for developing the search and recommendation system for staffing purpose. So, mainly the action research method was being followed for this project.</p> <p>Software project staffing is one of the main problems in software organizations. Searching for an employee based on simple queries to relational database is not sufficient to find a suitable match for a project. The Recommendation System based on Hadoop, HBase and MapReduce can efficiently recommend persons or teams from a set of available developers and according to project requirements. As a result this project developed an efficient staffing recommendation system on cloud computing platform, using Hadoop.</p> <p>The System recommends a list of persons that can replace the person leaving the project. The processing of data is very fast because of its parallel processing feature. This system takes less time to get the search results compared to other system based on the non-scalable Oracle database.</p>	
Keywords	cloud computing, Java, MapReduce, Hadoop, HDFS, HBase, ZooKeeper, recommendation algorithms, REST service

## Contents

1	Introduction	1
2	Conceptual Background	3
2.1	Definition of Cloud Computing	3
2.2	Apache Hadoop	4
2.3	ZooKeeper	5
2.4	MapReduce Programming Model	6
2.5	Recommendation and Search Algorithms	7
2.5.1	K-Nearest Neighbor Algorithm	7
2.5.2	User-based Collaborative Filtering Algorithm	7
2.6	REST Services	9
3	Hadoop Distributed File System	12
3.1	Overview	12
3.2	Architecture	12
3.2.1	NameNode	13
3.2.2	DataNodes	15
3.2.3	HDFS Client	16
3.3	Replication Management	18
3.4	Features of HDFS	18
3.5	Advantages and Disadvantages	20
4	HBase a Scalable, Distributed Database	22
4.1	Overview	22
4.2	Architecture	22
4.3	Data Model	24
4.4	HBase in Action	27
4.5	HBase Uses	29
4.6	HBase vs Relational Database	31
5	Recommendation and Search System Development	32
5.1	Analysis	32
5.2	Development Environment	34
5.3	System Architecture	35

5.4	Loading Data to HBase	37
5.4.1	Employee Table	39
5.4.2	Competence Table	42
5.4.3	Project Table	44
5.5	Using RESTful Services	49
5.5.1	Server	50
5.5.2	Client	53
5.6	Data Mining	55
5.6.1	Competence Data Analysis	55
5.6.2	Project Data Analysis	58
5.6.3	Recommendation	61
6	Conclusion	65
	References	67

## Abbreviations and Terms

ACID	Atomicity, Consistency, Isolation, Durability
API	Application Programming Interface
CRUD	Create, Retrieve, Update, Delete
DBA	Database Administrator
GNU	GNU's Not Unix
HDFS	Hadoop Distributed File System
HTTP	Hypertext Transfer Protocol
IO	Input/Output
IP	Internet Protocol
JDK	Java Development Kit
KNN	K-nearest Neighbor Algorithm
MIME	Multipurpose Internet Mail Extensions
OS	Operating System
RDMS	Relational Database Management System

REST	Representational State Transfer
RPC	Remote Procedure Call
SLAs	Service Level Agreements
SQL	Structured Query Language
TCP	Transmission Control Protocol
UI	User Interface
URL	Uniform Resource Locator
XML	Extensible Markup Language

## 1 Introduction

In the present day software era the ever growing data demands elastically scalable data centers which can be conveniently accessible with high quality in a secure way. This demand led to cloud computing as one of the emerging technologies of today. Cloud computing releases computer services, computer software, and data storage away from locally hosted infrastructure and into cloud-based solutions. The ability of the cloud services to provide apparently an unlimited supply of computing power on demand to users has caught the attention of industry as well as academia. In the past couple of years software providers have been moving more and more applications to the cloud[1].

Several big companies such as Amazon, Google, Microsoft, Yahoo, and IBM are using the cloud. Today, forward-thinking business leaders are using the cloud within their enterprise data centers to take advantage of the best practices that cloud computing has established, namely scalability, agility, automation, and resource sharing. By using a cloud-enabled application platform, companies can choose a hybrid approach to cloud computing that employs an organization's existing infrastructure to launch new cloud-enabled applications. This hybrid approach allows IT departments to focus on innovation for the business, reducing both capital and operational costs and automating the management of complex technologies. [2]

Cloud services claim to provide nearly everything needed to run a project without owning any IT infrastructure. From e-mail, Web hosting to fully managed applications resources can be provided on-demand. This helps to reduce development cost and hardware cost. [3]

Large corporations are regularly faced with the problem of staffing new projects. Assigning employees to projects based on their list of competences and project requirements requires complex queries. Cloud computing can provide a scalable method for answering these complex queries.

The goal of this thesis is to get an understanding of cloud computing. This includes the study and understanding of Hadoop Distributed File System (HDFS), HBase Data Storage and MapReduce. The main goal is to develop and implement a recommendation and search system for staffing using cloud computing. This system is for competences mining and is based on cloud-based technology using Hadoop.

If single person leaves a project, find replacement. From a given list of competence requirements, find a matching person. From a given complete list of requirements for a new project, find a team. From a given incomplete lists of competences for an employee, estimate missing competences. These are some use cases for this thesis.

This thesis project is done at Tieto Company in collaboration with a Alto University, and I am one of the research team members. I am responsible for the design and implementation of the software.

The study is to use Tieto Employee Competence Data for data mining as an input for searching and recommending the required profile. This study is done only for the data which has all the required information.



## 2 Conceptual Background

### 2.1 Definition of Cloud Computing

Cloud computing is basically services-on-demand over the Internet. It is a natural evolution of the widespread adoption of virtualization, service-oriented architecture and utility computing. [4]

#### Definitions of Cloud Computing

- a) Cloud computing is a computing capability that provides an abstraction between the computing resource and its underlying technical architecture (e.g., servers, storage, networks), enabling convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction.[5]
- b) There is no formal definition commonly shared in industry, unlike for Web 2.0, and it is very broadly defined as on-demand provisioning of application, resources, and services that allow resources to be scaled up and down.[6]
- c) Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs.[7]

The Key features of cloud computing are:

- Service: Everything, infrastructure, platform and software, is delivered as services.
- Scalability: Resources can be dynamically scalable over data center without much difficulty
- Cost effectiveness: It follows the “pay as use basis” model

- Interface: Independent of location, services are accessible as Web services via Web browsers
- Availability
- Virtualization

Cloud computing includes a number of technologies such as Virtualization, Web services, Service Oriented Architecture, Web 2.0, Web Mashup. One of the key new technologies used in the cloud are scalable batch processing systems. The main reference implementation here is Google MapReduce and its open source implementation Apache Hadoop originally developed at Yahoo.

Distribution and scalability are playing important roles in the cloud, and for that Hadoop can be used. Hadoop is a cloud computing program created to deal with the growing demands of modern computing and storage of massive amounts of data. Many companies, especially ones that operate through demanding websites, e.g. Amazon, Facebook, Yahoo, eHarmony and eBay, use Hadoop.[4]

## 2.2 Apache Hadoop

Hadoop is an open-source cloud computing environment that implements the MapReduce based on Google MapReduce. Hadoop is written in Java language; any machine that supports Java can run the Hadoop software. It has its own distributed file system called Hadoop Distributed File System (HDFS) which is based on GoogleFile System. Hadoop uses the HDFS to divide files among several nodes, with the processor of each node only working off their own storage. Hadoop is an [Apache project](#). Hadoop enables the development of reliable, scalable, efficient, economical and distributed computing using very simple Java interfaces - massive parallel code without the pain. HDFS based database used mainly for batch processing is HBase which is heavily inspired by Google Bigtable. The main applications for Hadoop seem to be log analysis, Web indexing, and various data mining and customer analysis applications.

HDFS and HBase are discussed in detail in chapters 3 and 4.

## 2.3 ZooKeeper

ZooKeeper is a Hadoop's distributed coordination service for building distributed applications. It provides a centralized service for managing queues of events, configuration maintenance, leader election, naming, distributed synchronization and providing group services. At its core, ZooKeeper is modeled after a straightforward, tree based, file system API [8]. It runs in Java and has bindings for both Java and C [9].

The ZooKeeper service is provided by a cluster of servers to avoid a single point of failure. ZooKeeper uses a distributed consensus protocol to determine which node in the ZooKeeper service is the leader at any given time.

### Architecture

There is only one leader among the servers and it is elected at the startup. Figure 1 showing the architecture diagram of Zookeeper. All servers store a copy of the data in memory. The main task of the leader is to coordinate and accept writes. All other servers are direct, read-only replicas of the master. This way, if the master goes down, any other server can pick up the slack and immediately continue serving requests. ZooKeeper allows the standby servers to serve reads. [8]

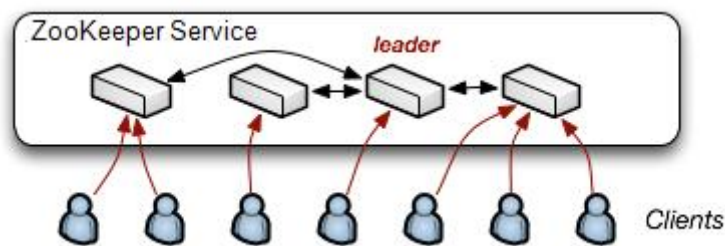


Figure 1. Architecture [8]

A limitation here is that every node in the cluster is an exact replica - there is no sharing and hence the capacity of the service is limited by the size of an individual machine. [8]

## 2.4 MapReduce Programming Model

MapReduce is a parallel programming model for processing large sets of data. Google introduced MapReduce in 2004 for distributed computing. This framework is implemented in C++ with interfaces in Python and Java. MapReduce consists of two main functions Map and Reducer. Computation takes input as a set of key/value pairs and produces output as set of key/value pairs.

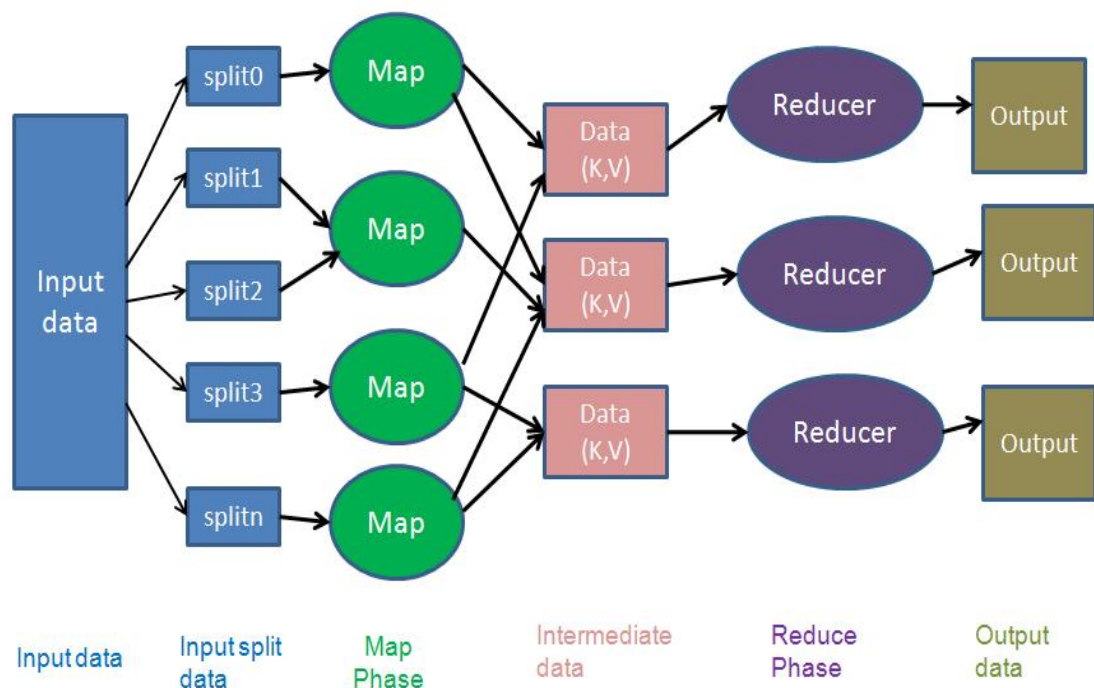


Figure 2. MapReduce

As illustrated in figure 2, MapReduce function call splits the input data into N pieces. Mapper maps input  $\langle \text{key}, \text{value} \rangle$  pairs to a set of intermediate  $\langle \text{key}, \text{value} \rangle$  pairs. Then all the intermediate pairs are sorted by unique keys, so that all the values of the same key are grouped together. The reducer reduces this grouped set of intermediate values. The outputs of all reduce functions are merged to produce the output of the complete MapReduce job.

## 2.5 Recommendation and Search Algorithms

The recommendation algorithm is for analyzing data for a particular problem to find the items a user is looking for and to produce a predicted likeliness score or a list of top N recommended items for a given user. Different algorithms can be used based on each use case. For this thesis, K-nearest neighbor has been used for finding a similar replacement of an employee leaving a project. User-based Collaborative Filtering Algorithm is used for predicting missing competences.

### 2.5.1 K-Nearest Neighbor Algorithm

The K-nearest neighbor algorithm (KNN) is part of supervised learning that has been used in many applications in the field of data mining, statistical pattern recognition and many others. KNN is a method for classifying objects based on the closest training examples in the feature space. An object is classified by a majority vote of its neighbors. K is always a positive integer. The neighbors are taken from a set of objects for which the correct classification is known. It is usual to use the Euclidean distance, though other distance measures can also be used instead. [10]

One of the advantages of the KNN is that it is well suited for multi-modal classes as its classification decision is based on a small neighborhood of similar objects (i.e., the major class). So, even if the target class is multi-modal (i.e., consists of objects whose independent variables have different characteristics for different subsets), it can still lead to good accuracy. A major drawback of the similarity measure used in the KNN is that it uses all features equally in computing similarities. This can lead to poor similarity measures and classification errors, when only a small subset of the features is useful for classification. [11]

### 2.5.2 User-based Collaborative Filtering Algorithm

A user-based collaborative filtering algorithm produces a recommendation list for the object user according to the view of other users. The assumption is that users with similar preferences will rate products similarly. Thus missing ratings for a user can be

predicted by first finding a neighborhood of similar users and then aggregating the ratings of these users to form a prediction.

User rating data can be a matrix  $A(m,n)$ , where  $m$  represents the number of users,  $n$  represents the number of items. Element  $R$  (at the  $i$ th line and  $j$ th column) represents the rating of the item  $j$  rated by user  $i$ . User rating data matrix is shown in figure 3. [12]

<i>User</i>	<i>Item</i>				
	Item <sub>1</sub>	...	Item <sub>k</sub>	...	Item <sub>n</sub>
User <sub>1</sub>	R <sub>1,1</sub>	...	R <sub>1,k</sub>	...	R <sub>1,n</sub>
...	...	...	...	...	...
User <sub>i</sub>	R <sub>i,1</sub>	...	R <sub>i,k</sub>	...	R <sub>i,n</sub>
...	...	...	...	...	...
User <sub>m</sub>	R <sub>m,1</sub>	...	R <sub>m,k</sub>	...	R <sub>m,n</sub>

Figure 3. User-item rating data matrix [12]

A cosine similarity algorithm can be used to measure the similarity between user  $i$  and  $j$ . Similarity between user  $i$  and user  $j$  is  $sim(i, j)$ :

$$sim(i, j) = \cos(\bar{i}, \bar{j}) = \frac{\bar{i} \cdot \bar{j}}{\|\bar{i}\| \cdot \|\bar{j}\|}$$

Rating of item  $i$  rated by user  $u$  can be predicted by rating of nearest neighbors set  $NBS_u$  rated by user  $u$ . The predicted rating of item  $i$  rated by user  $u$  is:

$$P_{u,i} = \bar{R}_u + \frac{\sum_{n \in NBS_u} sim(u, n) * (R_{n,i} - \bar{R}_n)}{\sum_{n \in NBS_u} (|sim(u, n)|)}$$

$NBS_u$  - the nearest neighbor set of user  $u$ ;

$P_{u,i}$  – Predicted rating by user  $u$  on item  $i$ ;

$sim(u,n)$  – the semblance between user  $u$  and  $n$ ;

$R_{n,i}$  – the rating of item  $i$  rated by user  $n$ ;

$\bar{R}_u$  – the average rating of items rated by user  $u$ ;

$\bar{R}_n$  – the average rating of items rated by user  $n$ .

According to the rating of items, select  $N$  items that have the highest rating to compose a recommendation set and recommend them to the object user. [12]

## 2.6 REST Services

REST is an architectural style which is based on Web standards and the HTTP protocol. In REST-based architecture everything is a resource. A resource is accessed via a common interface based on the HTTP standard methods. In REST architecture there is a REST server which provides access to the resources and a REST client which accesses and modifies the REST resources. [13] Figure 4 shows the JAX RS Architecture.

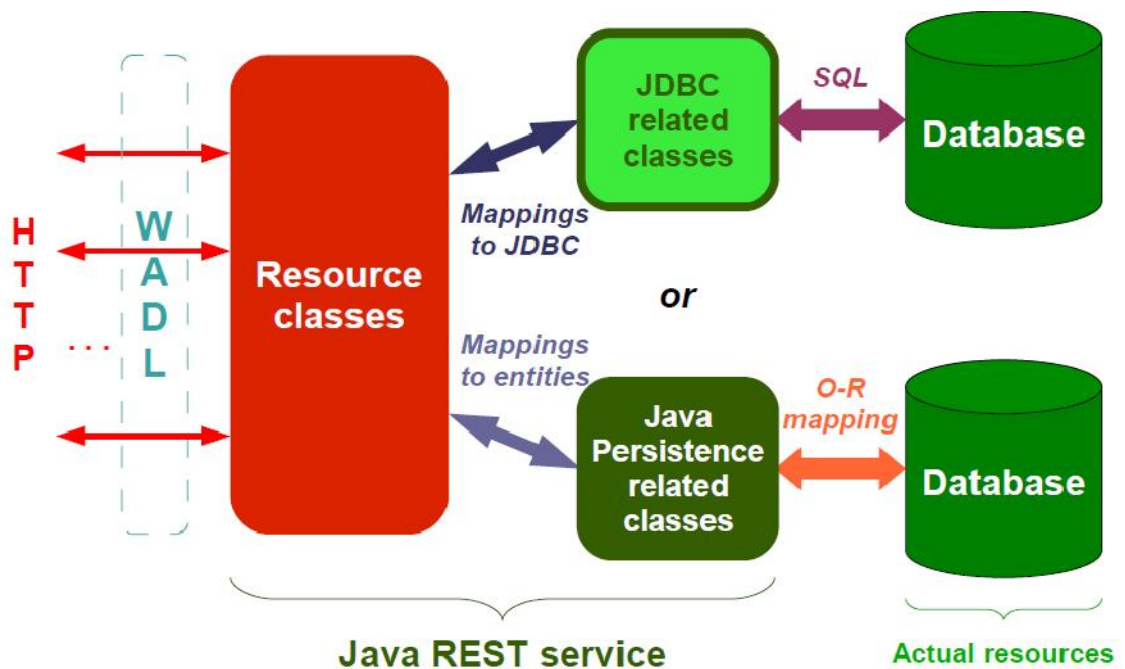


Figure 4. JAX RS Architecture

As shown in figure 4, every resource should support the HTTP common operations. Resources are identified by global IDs (which are typically URIs). REST allows that resources have different representations, e.g. text, XML, JSON etc. The client can ask for specific representation via the HTTP protocol (Content Negotiation). The HTTP standard methods which are typically used in REST are PUT, GET, POST and DELETE. [13]

### Java, REST and Jersey

Java technology (Java EE) defines standard REST support via JAX-RS (The Java API for RESTful Web Services) in [JSR 311](#) . [Jersey](#) is the reference implementation for this specification. Jersey contains basically the core server and the core client. The core client provides a library to communicate with the server. [13]

JAX-RS uses the following annotations to define the REST relevance of classes:

- `@Path`
- `@GET`
- `@POST`
- `@PUT`
- `@DELETE`
- `@Consumes`
- `@Produces`



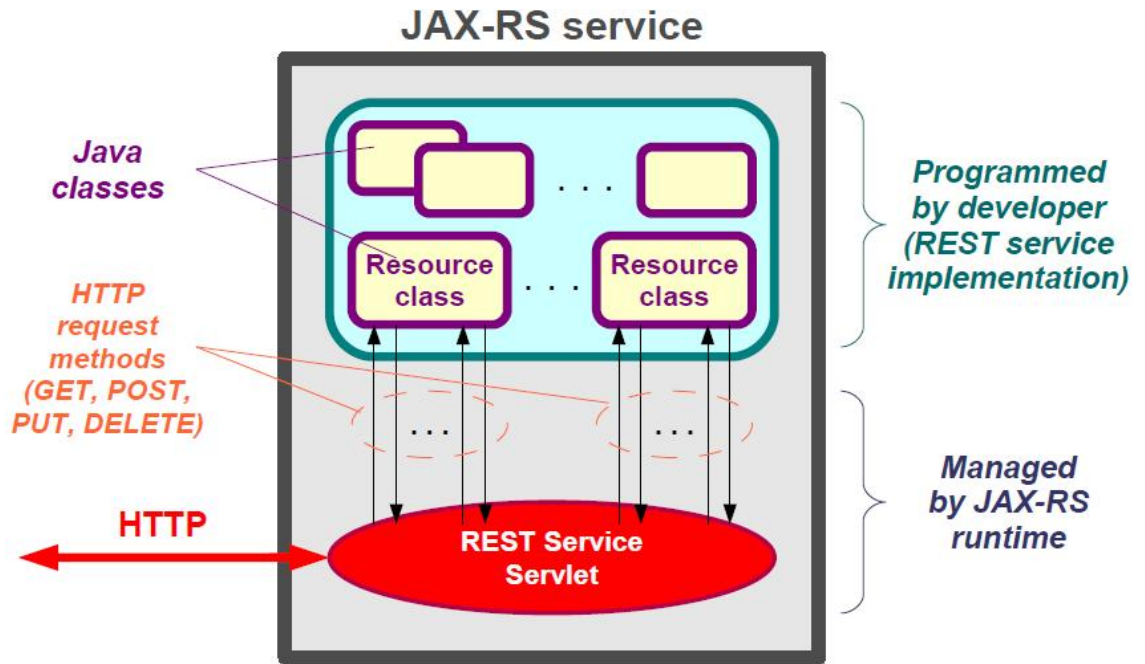


Figure 5. JAX RS in Action

Figure 5 illustrates the technical implementation of JAX RS services.

## 3 Hadoop Distributed File System

### 3.1 Overview

Hadoop comes with a distributed filesystem called HDFS (*Hadoop Distributed Filesystem*). HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets.

The Hadoop filesystem is designed for storing petabytes of a file with streaming data access using the idea that most efficient data processing pattern is a write-once, read-many-times pattern. HDFS stores metadata on a dedicated server, called NameNode. Application data are stored on other servers called DataNodes. All the servers are fully connected and communicate with each other using TCP-based protocols.

### 3.2 Architecture

HDFS is based on master/slave architecture. A HDFS cluster consists of a single NameNode (as master) and a number of DataNodes (as slaves). The NameNode and DataNodes are pieces of software designed to run on commodity machines. These machines typically run a GNU/Linux operating system (OS). The usage of the highly portable Java language means that HDFS can be deployed on a wide range of machines. A typical deployment has a dedicated machine that runs only the NameNode software. Each of the other machines in the cluster runs one instance of the DataNodes software. The architecture does not preclude running multiple DataNodes on the same machine but in a real deployment one machine usually runs one DataNode.

The existence of a single NameNode in a cluster greatly simplifies the architecture of the system. The NameNode is the arbitrator and repository for all HDFS metadata. The system is designed in such a way that user data never flows through the NameNode.

### 3.2.1 NameNode

As illustrated in figure 6, NameNode manages the filesystem namespace, metadata for all the files and directories in the tree.

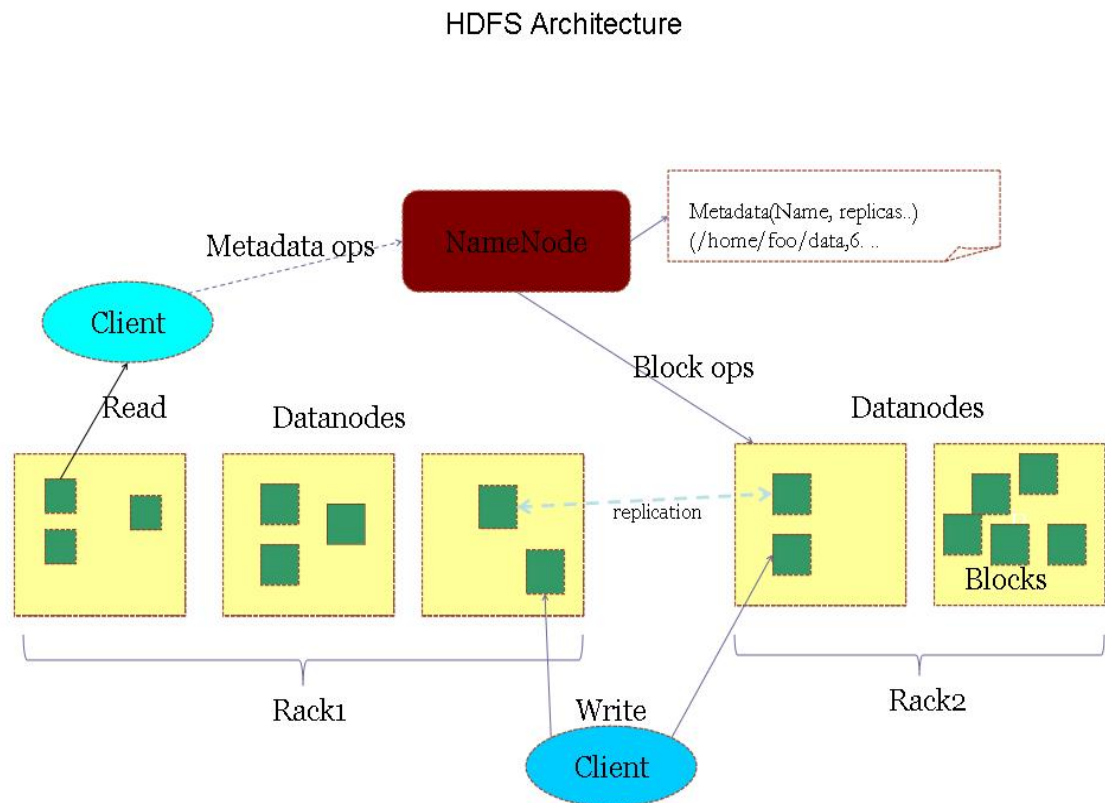


Figure 6. HDFS Architecture[15]

The file is divided into large blocks (typically 128 megabytes, but the user selectable file-by-file) and each block is independently replicated at multiple DataNodes (typically three, but user selectable file-by-file) to provide reliability. The NameNode maintains and stores the namespace tree and the mapping of file blocks to DataNodes persistently on the local disk in the form of two files: the namespace image and the edit log. The NameNode also knows the DataNodes on which all the blocks for a given file are located. However, it does not store block locations persistently, since this information is reconstructed from DataNodes when the system starts.

On the NameNode failure, the filesystem becomes inaccessible because only NameNode knows how to reconstruct the files from the blocks on the DataNodes. So,

for this reason, it is important to make the NameNode resilient to failure, and Hadoop provides two mechanisms for this: Checkpoint Node and Backup Node.

### Checkpoint Node

Checkpoint is an image record written persistently to disk. NameNode uses two types of files to persist its namespace:

- Fsimage: the latest checkpoint of the namespace
- Edits: logs containing changes to the namespace; these logs are also called journals.

NameNode creates an updated file system metadata by merging both files i.e. fsimage and edits on restart. The NameNode then overwrites fsimage with the new HDFS state and begins a new edits journal.

The Checkpoint node periodically downloads the latest fsimage and edits from the active NameNode to create checkpoints by merging them locally and then to upload new checkpoints back to the active NameNode. This requires the same memory space as that of NameNode and so checkpoint needs to be run on separate machine. Namespace information lost if either the checkpoint or the journal is missing, so it is highly recommended to configure HDFS to store the checkpoint and journal in multiple storage directories.

The Checkpoint node uses parameter `fs.checkpoint.period` to check the interval between two consecutive checkpoints. The Interval time is in seconds (default is 3600 second). The Edit log file size is specified by parameter `fs.checkpoint.size` (default size 64MB) and a checkpoint triggers if size exceeds. Multiple checkpoint nodes may be specified in the cluster configuration file.

### Backup Node

The Backup node has the same functionality as the Checkpoint node. In addition, it maintains an in-memory, up-to-date copy of the file system namespace that is always synchronized with the active NameNode state. Along with accepting a journal stream of the filesystem edits from the NameNode and persisting this to disk, the Backup node

also applies those edits into its own copy of the namespace in memory, thus creating a backup of the namespace. [14]

Unlike the Checkpoint node, the Backup node has an up-to-date state of the namespace state in memory. The Backup node requires same RAM as of NameNode. The NameNode supports one Backup node at a time. No Checkpoint nodes may be registered if a Backup node is in use. The Backup node takes care of the namespace data persistence and NameNode does not need to have persistent store. [14]

### 3.2.2 DataNodes

There are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes the file system namespace operations such as opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. DataNodes store and retrieve blocks when requested (by clients or the NameNode), and they report back to the NameNode periodically with lists of blocks they are storing. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode. [15]

DataNodes and NameNode connections are established by handshake where *namespace ID* and the *software version* of the DataNodes are verified. The namespace ID is assigned to the file system instance when it is formatted. The namespace ID is stored persistently on all nodes of the cluster. A different namespace ID node cannot join the cluster.

A new DataNode without any namespace ID can join the cluster and receive the cluster's namespace ID and DataNode *registers* with the NameNode with storage ID. A DataNode identifies block replicas in its possession to the NameNode by sending a *block report*. A block report contains the *block id*, the *generation stamp* and the length for each block replica the server hosts. The first block report is sent immediately after the DataNodes registrations. Subsequent block reports are sent every hour and provide

the NameNode with an up-to date view of where block replicas are located on the cluster. [16]

### 3.2.3 HDFS Client

#### Reading a file

To read a file, HDFS client first contacts NameNode. It returns list of addresses of the DataNodes that have a copy of the blocks of the file. Then client connects to the closest DataNodes directly for each block and requests the transfer of the desired block. Figure 7 shows the main sequence of events involved in reading data from HDFS.

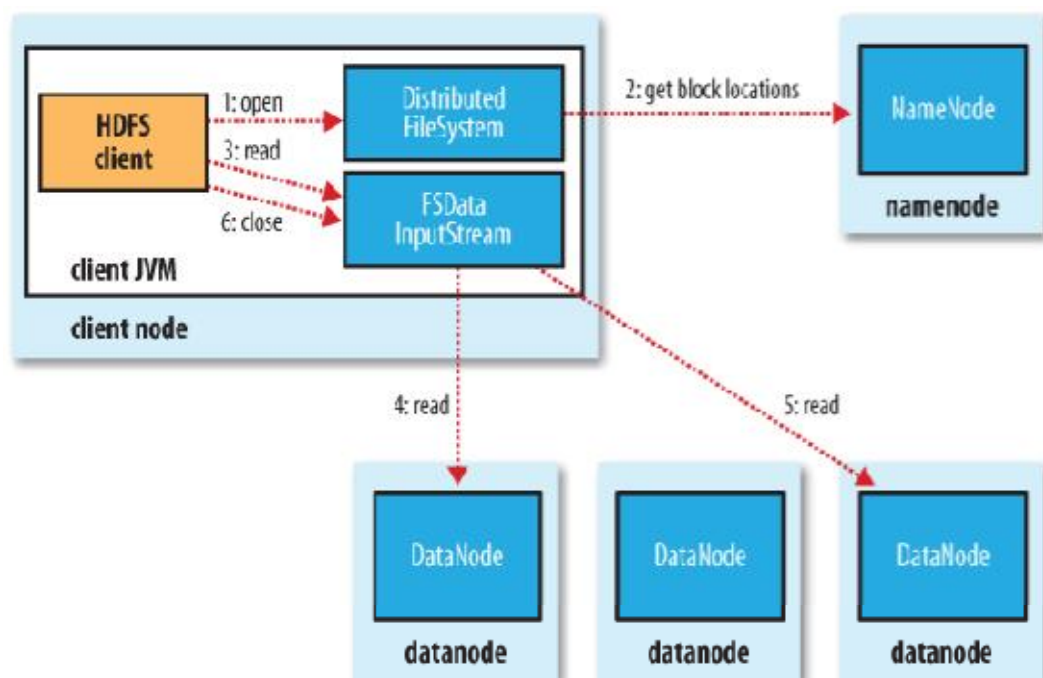


Figure 7. A client reading data from HDFS [16,63]

## Writing to a File

For writing to a file, HDFS client first creates an empty file without any blocks. File creation is only possible when the client has writing permission and a new file does not exist in the system. NameNode records new file creation and allocates data blocks to list of suitable DataNodes to host replicas of the first block of the file. Replication of data makes DataNodes in pipeline. When the first block is filled, new DataNodes are requested to host replicas of the next block. A new pipeline is organized, and the client sends the further bytes of the file. Each choice of DataNodes is likely to be different.

If a DataNode in pipeline fails while writing the data then pipeline is first closed and partial block on failed data node is deleted and failed DataNode is removed from the pipeline. New DataNodes in the pipeline are chosen to write remaining blocks of data. Figure 8 shows the steps involved in creating a new file, writing data to it and then closing the file.

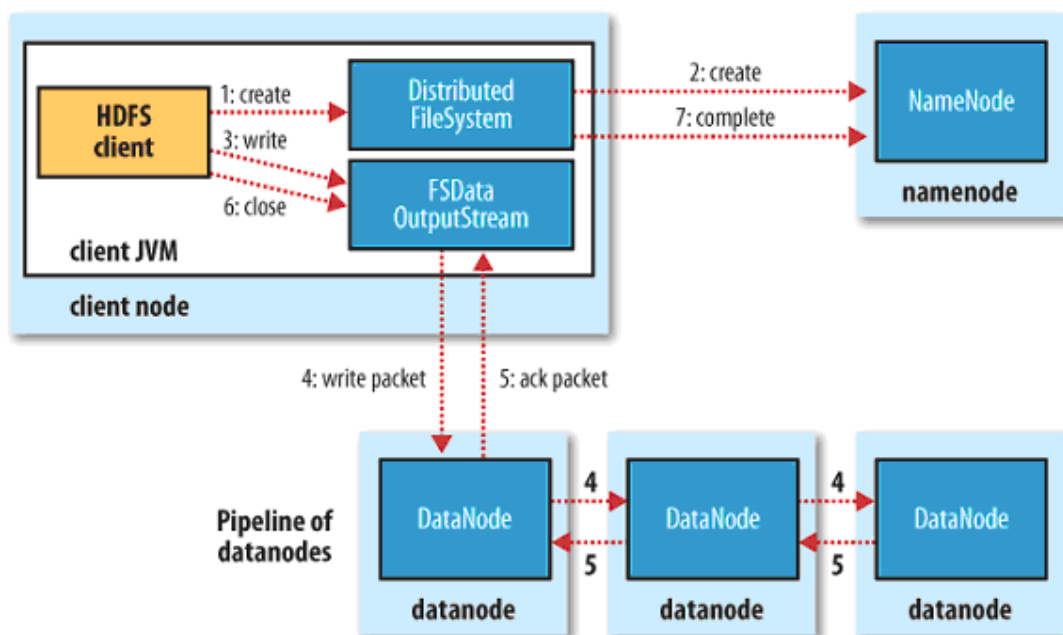


Figure 8. A client writing data to HDFS [16,66]

### 3.3 Replication Management

The NameNode is responsible for block replication. Replica placement determines HDFS reliability, availability and performance. Each replica on unique racks helps in preventing data loses on entire rack failure and allows use of bandwidth from multiple racks when reading data. This policy evenly distributes replicas in the cluster which makes it easy to balance load on component failure. However, this policy increases the cost of writes because a write needs to transfer blocks to multiple racks.

The NameNode keeps checking the number of replicas. If a block is under replication, then it is put in the replication priority queue. The highest priority is given to low replica value. Placement of new replica is also based on priority of replication. If the number of existing replicas is one, then a different rack is chosen to place the next replica. In case of two replicas of the block on the same rack, the third replica is placed on a different rack. Otherwise, the third replica is placed on a different node in the same rack as an existing replica.

The NameNode also checks that all replica of a block should not be at one rack. If so, NameNode treats the block as under-replicated and replicates the block to a different rack and deletes the old replica.

### 3.4 Features of HDFS

#### Communication Protocols

All HDFS communication protocols are layered on top of the TCP/IP protocol. A client establishes a connection to a configurable TCP port on the NameNode machine. It talks the ClientProtocol with the NameNode. The DataNodes talk to the NameNode using the DataNodes Protocol. A Remote Procedure Call (RPC) abstraction wraps both the Client Protocol and the DataNodes Protocol. By design, the NameNode never initiates any RPCs. Instead, it only responds to RPC requests issued by DataNodes or clients. [15]



## Data Disk Failure, Heartbeats and Re-Replication

The primary objective of the HDFS is to store data reliably even in the presence of failures. The three common types of failures are NameNode failures, DataNodes failures and network partitions. [15]

NameNode considers DataNodes as alive as long as it receives Heartbeat message (default Heartbeat interval is three seconds) from DataNodes. If the NameNode does not receive a heartbeat from a DataNodes in ten minutes the NameNode considers the DataNodes as dead and stop forwarding IO request to it. The NameNode then schedules the creation of new replicas of those blocks on other DataNodes.

Heartbeats carry information about total storage capacity, fraction of storage in use, and the number of data transfers currently in progress. These statistics are used for the NameNode's space allocation and load balancing decisions. The NameNode can process thousands of heartbeats per second without affecting other NameNode operations.

## Cluster Rebalancing

The HDFS architecture has data rebalancing schemes in which data is automatically moved from one DataNode to another if the free space threshold is reached. In the event of a sudden high demand for a particular file, a scheme might dynamically create additional replicas and rebalance other data in the cluster. These types of data rebalancing schemes are not yet implemented. [15]

## Data Integrity

Block of data can be corrupted due to many reasons such as network faults, buggy software or faults in a storage device. So, at the time of file creation checksum is used and stored for each block. While retrieving a file, it is first verified with those checksums and if verification fails, then another replica of data is used.

## Metadata Disk Failure

Corrupted Fsimage and the EditLog may stop the HDFS functioning. For redundancy, NameNode is configured to have multiple copies of these files and are updated synchronously.

## Snapshots

A Snapshot saves the current state of the file system at any instance of time. The main usage of this feature is to rollback to the previous state if the upgrading resulted in data loss or corruption.

### 3.5 Advantages and Disadvantages

Advantages of the HDFS are:

- **Reliable storage** - HDFS is a fault tolerant storage system. HDFS can significantly store huge amounts of data, scale up incrementally and can effectively handle the failure of significant parts of the storage infrastructure without losing data.
- **Commodity hardware** - HDFS is designed to run on highly unreliable hardware and so is less expensive compared to other fault tolerant storage systems.
- **Distributed** - HDFS data are distributed over many nodes in a cluster and so parallel analyses are possible and this eliminates the bottlenecks imposed by monolithic storage systems.
- **Availability** - Block replication is one of the main features of HDFS. By default each block is replicated by the client to three DataNodes but replication factor can be configured more than 3 at creation time. Because of replication HDFS provides high availability of data in high demand.

Limitations in HDFS are:

- **Architectural bottlenecks** - There are scheduling delays in the Hadoop architecture that result in cluster nodes waiting for new tasks. More over disk is not used in a streaming manner, the access pattern is periodic. HDFS client

serializes computation and I/O instead of decoupling and pipelining those operations.

- Portability limitations - HDFS being in Java could not able to support some performance-enhancing features in the native filesystem.
- Small file - HDFS is not efficient for large numbers of small files.
- Single MasterNode – There might be risk of data loss because of single NameNode.
- Latency data access – At the expense of latency HDFS delivers a high throughput of data. If an application needs low-latency access to data then HDFS is not a good choice.

One of the major advantages of the HDFS is scalability. Besides its limitations, HDFS is highly in demand when data sets are very large where scalability plays an important role.

## 4 HBase a Scalable, Distributed Database

### 4.1 Overview

HBase is the open-source distributed column-oriented database system for the management of a large volume of structured data. HBase includes most of the functionalities provided by Google BigTable. HBase is written in Java and it is the Hadoop application to use real-time read/write random-access to very large datasets. HBase can scale linearly by adding nodes; it does not support backward scaling.

Data in HBase are organized in tables, rows and columns. Each particular column can have several versions for the same row key. It does not support SQL but it is able to host very large, sparsely populated tables on clusters made from commodity hardware.

HBase can be accessed through technologies such as Java Client/API and MapReduce. HBase classes and utilities in the `org.apache.hadoop.HBase.mapred` package facilitate using HBase as a source and/or sink in MapReduce jobs. REST server, Thrift Gateway API and Iruby Shell are also used to access HBase. [16,350-353;22]

History/ background of HBase:

The HBase project was started by Chad Walters and Jim Kellerman of Powerset at the end of 2006. It was modeled after Google's "BigTable". The first HBase release was bundled as part of Hadoop 0.15.0. HBase became a Hadoop subproject at the beginning of 2008. HBase has been in production use at Powerset since late 2007. Other production users of HBase include WorldLingo, Streamy.com, OpenPlaces, and groups at Yahoo! and Adobe. [16,344]

### 4.2 Architecture

There are three major components of the HBase architecture:  
HBaseMaster, HRegionServer and HBase Client.

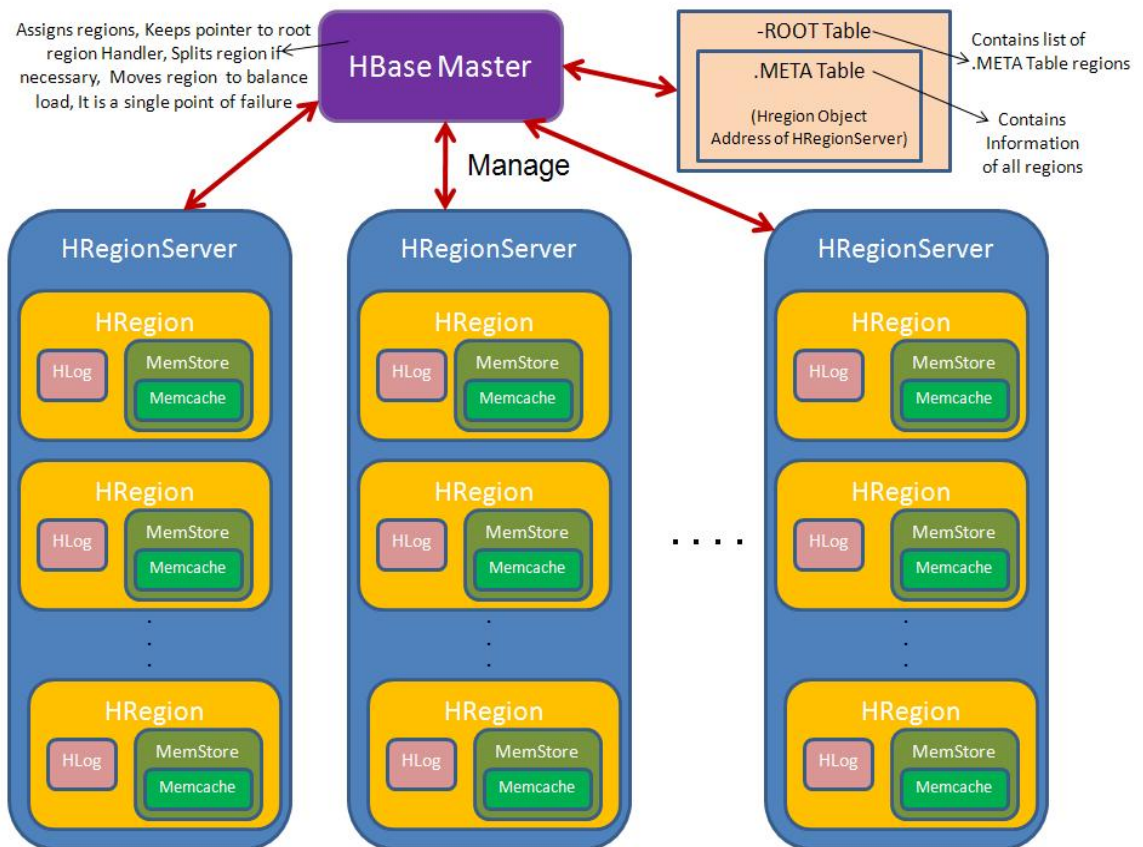


Figure 9. HBase Architecture

As illustrated in figure 9, HBase master manages the HBase cluster by assigning regions to registered HRegionServers, and responsible for recovering HRegionServer failures.

HRegionServers carries zero or more regions and responsible for handling client read/write requests. It contacts HBaseMaster to get a list of regions to serve. HRegionServers also send notifications to the HBaseMaster that it is alive.

The HBase client's main task is to find HRegionServers that are serving the particular row range of interest. As explained in figure 10, on instantiation, HBase client directly contacts HBaseMaster to find the location of the -ROOT region. Then, the client contacts that region server and scans the -ROOT region to find the .META region that will contain the location of the user region. Then the client contacts HRegionServer

servicing that region and issues the read or write request. Above information is cached in the client so that subsequent requests need not go through this process. [22]

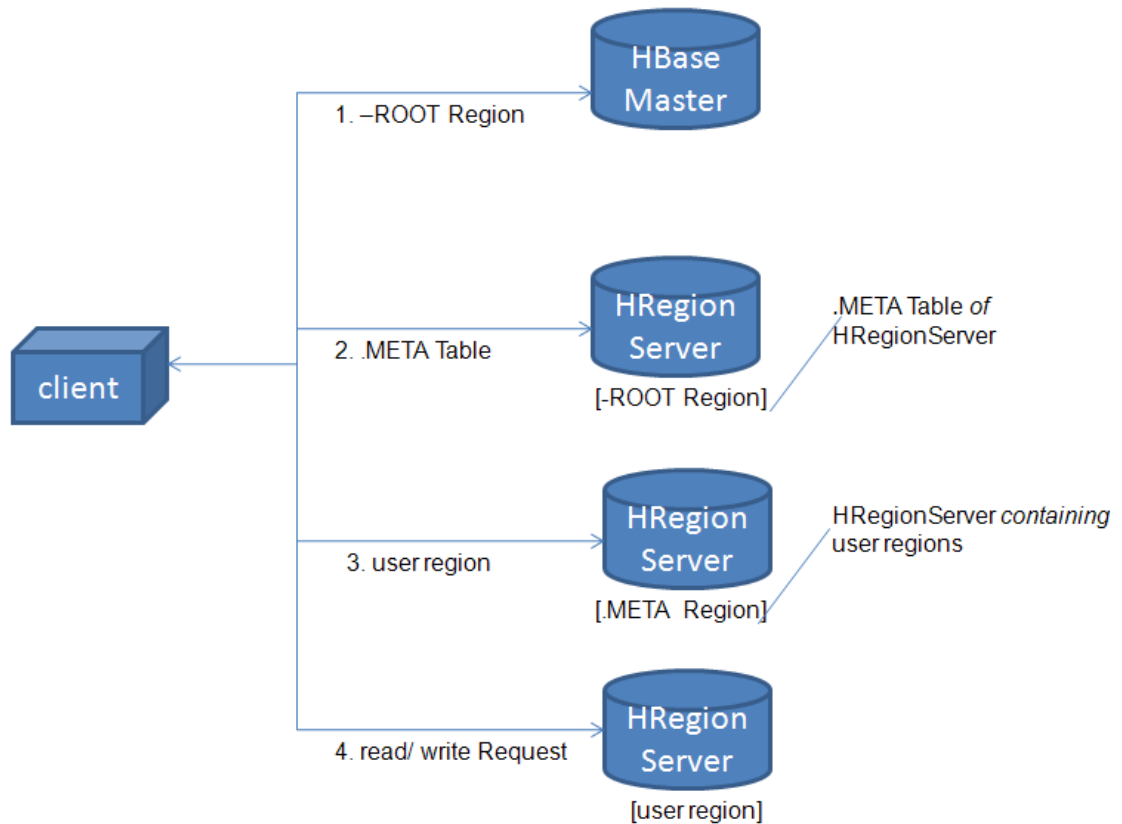


Figure 10. Client Communication

If a region is reassigned either by the master for loads balancing or because a HRegionServer has died, the client will rescan the .META table to determine the new location of the user region. If the .META region has been reassigned, the client will rescan the -ROOT region to determine the new location of the .META region. If the -ROOT region has been reassigned, the client will contact the master to determine the new -ROOT region location and will locate the user region by repeating the original process described above. [22]

### 4.3 Data Model

HBase uses a data model which is similar to Google BigTable's data model. The applications keep the rows of data in labeled tables. Each row has a sorting key and an

arbitrary number of columns. Table row keys are byte arrays. Sorting of table rows can be done by row key and the table's primary key.

id	Name	Age	Interests
1	Ricky		Soccer, Movies, Baseball
2	Ankur	20	
3	Sam	25	Music

Multi-valued

null

Figure 11. Row Oriented Database Table (*RDBMS Model*) [23]

Figure 11 shows a row-oriented database table layout.

id	Name
1	Ricky
2	Ankur
3	Sam

id	Age
2	20
3	25

id	Interests
1	Soccer
1	Movies
1	Baseball
3	Music

Figure 12. Column Oriented Database Tables (*Multi-value sorted map*) [23]

Figure 12 shows how a column-oriented database table differs from a row-oriented database table when same data is loaded to it.

### Column Families

Rows of one table can have a variable number of columns. A column name is of the form "<family>:<label>" with an arbitrary string of bytes. A table is created with a <family> set, known as "column families". All column family have same prefix for example: *book:author* and *book:publication* columns are members of book column family. The family name must be string whereas the label can be of any arbitrary bytes.

Column can be updated by adding new column family members as per the update request by client. Example: *book:science* can be added to column family *book* as long as *book* exists otherwise not.

A new <label> can be used in any writing operation, without any previous specification. HBase stores "column families" physically grouped on the disk, so the items in a certain column have the same particular read/write characteristics and contain similar data. By default only single row may be locked at a time. Row writes are always atomic, but single row may be locked to perform both read and write operations on that row atomically. Recent versions allow blocking several rows, if the option has been explicitly activated. [16;22]

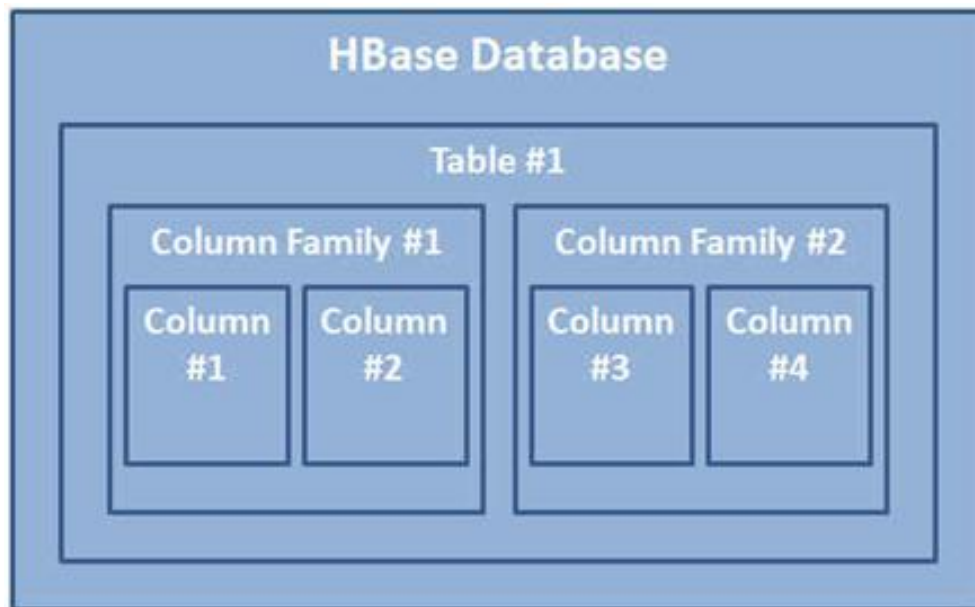


Figure 13. HBase Data Organization [24]

Figure 13 shows the structure of a table in HBase.

#### Table Cells

The HBase data is modeled as a multidimensional map in which values (the table *cells*) are indexed by four keys:

value = Map(TableName, RowKey, ColumnKey, Timestamp)



where:

- TableName is a string
- RowKey and ColumnKey are binary values (Java byte[])
- Timestamp is a 64-bit integer (Java long)
- value is an uninterrupted array of bytes (Java byte[])

Binary data is encoded in Base64 for transmission over the wire. The row key is the primary key of the table and is typically a string. Rows are sorted by row key in lexicographic order. [25]

## Regions

Tables are divided into row range called regions. Each region has subsets of table's rows. A region is defined by including its first row and excluding last row along with an identifier generated randomly. Mainly a table has one region but if the table size exceeds the threshold then it gets split into two equal regions. As the table grows, the number of its regions grows. Regions are the units that get distributed over an HBase cluster.

### 4.4 HBase in Action

HBase maintains the current list, state, recent history, and location of all regions on the cluster with an internal catalogue tables `-ROOT` and `.META`. As shown in figure 14, the `-ROOT` table contains the list of `.META` table regions.

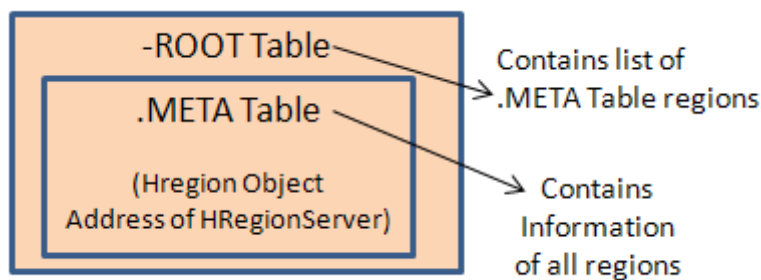


Figure 14. `-ROOT` and `.META` Tables

The .META table keeps the information of all user-space regions [16] by including a HRegionInfo object containing information such as the start and end row keys, status of region whether the region is on-line or off-line, etc. and the address of the HRegionServer that is currently serving the region. The .META table can grow as the number of user regions grows. [22]

Regions are redeployed when there is need of load balancing or if HRegionServer crashes. Region can be in disabled or enabled state or may be deleted, all these transaction of regions states are updated to catalog tables to keep current state of regions. Figure 15 shows HBase cluster members.

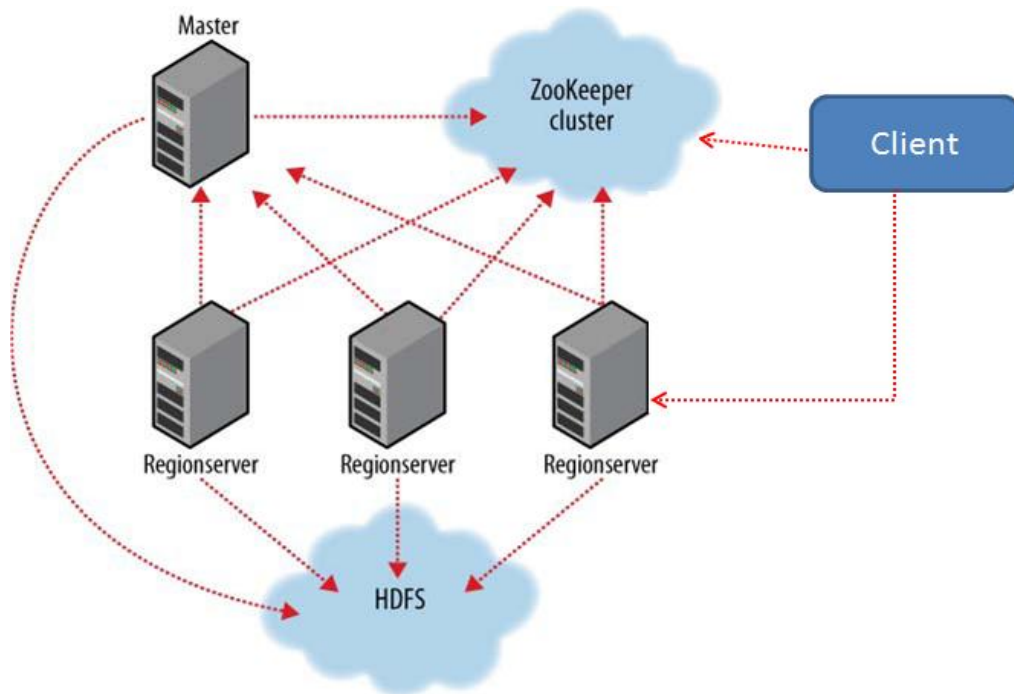


Figure 15. HBase cluster members [16]

As shown in figure 15, clients find the location of user region by consulting ZooKeeper cluster, -ROOT and .META region step by step as explained earlier in figure 10.

### Write Requests

When a HRegionserver receives a write request, it first writes to a commit log and then adds to an in-memory cache called *Memcache*. When this cache fills, its content is flushed to the filesystem. The commit log is hosted on HDFS, so it remains available

through a HRegionserver crash. If HRegionserver is no longer reachable, it is considered as dead and master splits the dead HRegionserver's commit log by region. On reassignment, regions that were on the dead HRegionserver, before they open for business, pick up their just-split file of not yet persisted edits and replay them to bring themselves up-to-date with the state they had just before the failure[16,347].

### Read Request

For read request, the region's memcache is consulted first. If sufficient versions are found to satisfy the query, then it returns. Otherwise, flush files are consulted in order, from newest to oldest until sufficient versions are found or until it run out of flush files to consult. [16,348;22]

## 4.5 HBase Uses

HBase should be used when there is a need for the following:

- Fault-tolerance: Replication is built in to provide fault tolerance, high availability and locality.
- Random access: Random access of real time read/write access to data stored in HDFS.
- Sparse data: When data is non-structured or semi-structured, not homogeneous or having complex data model, then HBase is a good choice. In HBase records, data are addressed with a row key/column family/cell qualifier/timestamp. Heterogeneous data can store much anything in a column family without having to know what it will be in advance. This allows to store one-to-many relationships in a single row. A given row can have any number of columns in each column family, or none at all.
- Large data set: A relational database fails with large data with big queries or table scans with big tables, with terabytes or petabytes. Standard RDBMS cannot process such workload in a timely and in cost-effective manner.

Processing such data raises waits and deadlocks nonlinearly with transaction size and concurrency. HBase table scans run in linear time, and row lookup or update times are logarithmic with respect to the size of the table.

- Scalability: Scalability is one of the facts where traditional RDBMS fails. Most RDBMSs are single process system, so scaling such database means replacing the existing hardware with more expensive hardware which have more CPU, RAM, and disk space. [16,361]. The scaling of an RDBMS usually involves loosening ACID restrictions, forgetting conventional DBA wisdom, and losing most of the desirable properties that made relational databases so convenient in the first place [16,361]. HBase is distributed over numbers of commodity servers so scaling in HBase is very simple and cost effective.
- Reliability: HBase has replication mechanism to avoid single node of failure. Backups are available to provide reliability of data.

[16,361;26;27]

HBase should not be used in the following cases:

- If the data model is simple and the entities are homogenous then probably RDBMS is a good choice. Mapping object to table is simple and has ability to query on non-primary-key values.
- When data is not large and can fit into standard RDBMS definitely there is no need of HBase. RDBMSs are fixed-schema, row-oriented databases with ACID properties and a sophisticated SQL query engine. The emphasis is on strong consistency, referential integrity, abstraction from the physical layer, and complex queries through the SQL language. It is easy to create secondary indexes, perform complex inner and outer joins, count, sum, sort, group, and page the data across a number of tables, rows, and columns. Mainly small-volume to medium-volume applications, there is no substitute for the ease of use, flexibility, maturity, and powerful feature set

of available open source RDBMS solutions such as MySQL and PostgreSQL [16,361].

- Another case when HBase shouldn't be used is to store large amounts of binary data. RDBMSs are built to be fast metadata stores.
- Finally, when SQL is required HBase cannot be used as HBase does not support SQL.

[16,361;26;27]

#### 4.6 HBase vs Relational Database

Table 1 lists the HBase and SQL benefits.

Table 1. Benefits of SQL and HBase

SQL benefits	HBase benefits
<ul style="list-style-type: none"> <li>- Joining: easy to get all products in an order with their product information in a single query.</li> <li>- Secondary indexing is possible</li> <li>- Realtime analysis: Group by and order by allows simple statistical analysis</li> </ul>	<ul style="list-style-type: none"> <li>- Dataset scale</li> <li>- Read/Write scale: read/write is distributed as tables are distributed across nodes</li> <li>- Replication is automatic</li> <li>- Batch analysis</li> </ul>

#### RDBMS vs Hbase

Cost: RDBMS usually need expensive disks. Have a single node and requires a backup server with same specifications. Hbase is designed for commodity hardware.

Reliability: RDBMS has slave replication and single node failure. Hbase has built in replication and backups are available. [16;28]

## 5 Recommendation and Search System Development

### 5.1 Analysis

In big software organizations successful products is one of the most important aspects of organization achievement. Team formation directly affects the performance of the team and the product development quality. Staffing is an important issue to be analyzed when software development is undertaken as a value-driven business.

Creating competence-based competitiveness is a great challenge because competence identification and consequently, its management, helps in innovation, decision support, faster process and product quality improvement, and constitute an important input to the creation of the firm's organizational knowledge. The finding a value driven developer-to-project assignment is a complex task. Predicting the compatibility of one employee with other team members is much more complex. [30;31;32]

Tieto is a huge company that focuses on project-based work, such as consulting or ITservice businesses, which has team-based work structure as the standard way of working. Employees are frequently staffed to project teams and dispersed as soon as the project ends. Hence, the effectiveness of work teams becomes crucial for the success of Tieto. So, Tieto Company is regularly facing with the problem of staffing new projects in terms of finding an employee that best fits with the project and the team.

Tieto competence database contains records for approx. 15.000 employees and 3.500 possible competences. Assigning employees to projects, based on their list of competences and project requirements requires a scalable and efficient system to process such large set of data.

Tieto's existing staffing system only supports common keyword-based search and filter techniques that focus solely on a candidate's technical skill set. A required candidate can be filtered by required compulsory competences and non compulsory competences of a given project. Search process is based on standard database queries which are time consuming process to make a search in 15.000 employees of 3.500 competencies.

It just considers a match between employee's competences with job competence requirements. The existing system does not support the fitness of the employee with team members in terms of interpersonal compatibility, which is equally important while finding a team member for a particular project.

To develop such a smart, scalable recommendation and search system, a cloud cluster of four servers was configured. Hadoop framework has been chosen to provide the scalability and efficient data mining. HBase distributed column-oriented database on top of Hadoop (Google Bigtable) was chosen for storing data.

### Use Cases

Some of the use cases that have been carried out for this project are:

- A single person leaves the project, find replacement.
- Given a list of competence requirements, find matching persons.
- Given a complete list of requirements for new project, find a team.
- Given an incomplete list of competences for employee, estimate missing competences.

### Data Set

Sparsity of data causes problems for pre-determined clustering according to competences. The user should give relevant feedback to focus the search.

Table 2. Missing data

Employee	C++	Java	SQL	JavaEE
102	2	3	0	2
106	4	0	3	0
107	3	3	1	?

Moreover, inference of missing competences may be challenging but highly desirable, as summarized in table 2

## 5.2 Development Environment

Four Linux servers were used to build a cluster for cloud environment. Hadoop framework and HBase file storage were installed at each server in which one server node is acting as master (as NameNode) and others are slaves (as DataNodes). HBase tables were created to store Employees Competence Data (of Tieto company).

### Runtime Environments

Hadoop engine was deployed over the 4 node cluster and Java runtime was setup to use the common Hadoop configuration, as specified by the NameNode (master node) in the cluster

### Software and Language Versions

- Hadoop Official release 0.20.1
- HBase official release 0.20.6
- Java, JDK 1.6.0\_23
- Tomcat 6.0.29
- Jersey for REST Services

### Hardware Specification of Each Hadoop Node:

Hadoop clusters have identical hardware specifications for all the cluster nodes. Table 3 lists the specification of nodes.

Table 3. Specification of cluster nodes

Operating system	SUSE Enterprise Linux 11 (32-bit)
Memory	2048MB
Processor	2 V CPU
Disk space	40 GB



### 5.3 System Architecture

Figure 16 shows the layered stack of Recommendation and Search System:

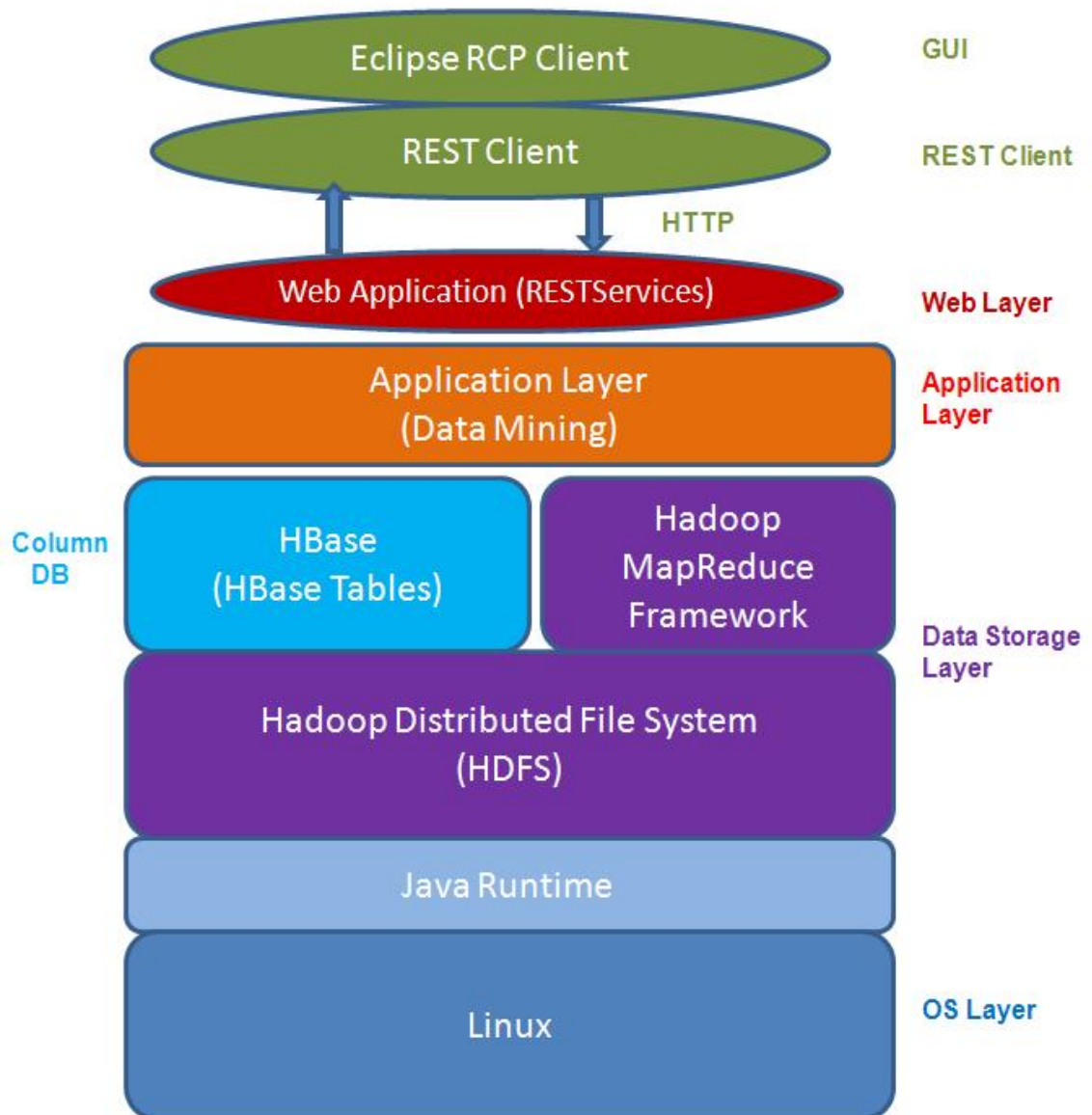


Figure 16. System Architecture

As shown in figure 16, Linux is used as an operating system and Java runtime environment is installed on top of it. Other main components of the System are explained as follows:

- Hadoop is installed on Java Runtime Environment.
- HDFS stores input and output files.

- HBase stores data in tables.
- MapReduce framework has been used to create tables in HBase and loads input data from HDFS to HBase tables.
- Application layer includes recommendation algorithms and search logics.
- REST Services have been deployed to retrieve the output results from HBase tables.
- REST Client sends HTTP requests and receives HTTP responses from target REST WebServices
- Eclipse RCP Client is GUI to show the search and recommendation results and interacting with end-user.

Figure 17 illustrates the component interactions.

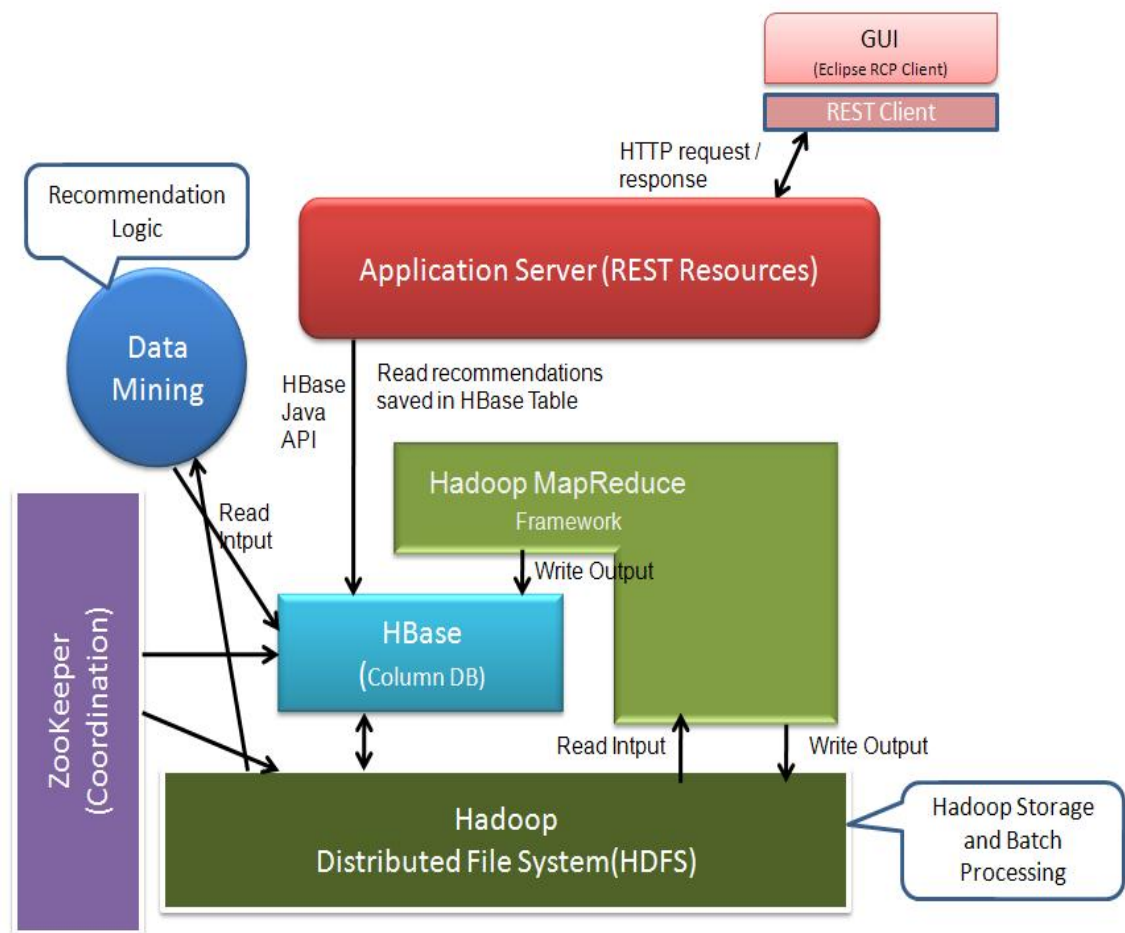


Figure 17. Workflow

The recommendation was based on K-N-N algorithm to find the replacement of a person leaving from the project. The competence data of employees and project data were two input files for processing. These two data files were loaded to HDFS and saved in CSV file format. The lists of competences of employees were grouped. Grouping was done on the basis of target use i.e. where competences can be used. For example, competence can be used for mobile, networking, server side development, or for management etc.

MapReduce implements this logic of clustering the competence lists into groups of competence. MapReduce function reads input files from HDFS, processes them and creates table in HBase to store the output data in table format. Data mining involves the finding the distance between employees based on their competence.

REST services provide access to output results stored in HBase tables. It also supports basic database CRUD operations to the HBase table. New profiles of competences can be created. Existing profiles can be updated and deleted. All profiles and a profile at a time can be retrieved.

#### 5.4 Loading Data to HBase

Competence data and project data were saved into HBase tables by using MapReduce. As shown in figure 18, competence data and project data files were loaded as CSV files in Hadoop distributed file system (HDFS). MapReduce function takes these files from HDFS as input for mapper class and maps to an intermediate <key,value> pairs. Then the reducer combines these intermediate pairs based on similar key to produce the reduced output. And finally, loads this reduced output by creating table in HBase and loading data to the created table.

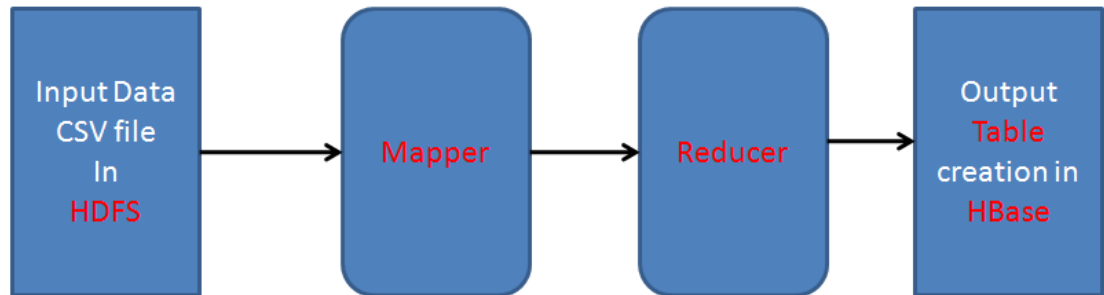


Figure 18. Competence data load by MapReduce

Tieto competence database contains records for approx. 15.000 employees, 3.500 possible competences and 3000 projects. To get proper relationship between employees, competence and projects competence data and project data are loaded in three tables named "Employee", "Competence" and "Project".

A snippet of source code for mapping input CSV file to intermediate <key,value> pairs in the MapReduce framework is given in the next page.

```

    /**
     * This method is called as many times as there are rows in CSV
     * file.
     * @param key the key
     * @param line the line from CSV file
     * @param context the map where the data is written from CSV file
     * @throws IOException Signals that an I/O exception has
    occurred.
     * @throws InterruptedException the interrupted exception
     */
    @Override
    public void map(LongWritable key, Text line, Context context)
    throws IOException, InterruptedException {
        if (rowsWritten == 0) {
            rowsWritten = 1;
            // Skip the first row as its a "header" row
            return;
        }
        // Save the information from CSV file to map. Map item's
        // key is the ID value of the CSV file's row

        String[] values =
line.toString().split(ProjectConstants.SeparatorChar);
        MapWritable map = new MapWritable();
        map.put(new
IntWritable(ProjectConstants.PROJECTSTARTDATEINDEX), new Text(
            values[ProjectConstants.PROJECTSTARTDATEINDEX]));
        map.put(new
IntWritable(ProjectConstants.PROJECTENDDATEINDEX), new Text(
            values[ProjectConstants.PROJECTENDDATEINDEX]));
        map.put(new IntWritable(ProjectConstants.ROLEINDEX), new
Text(
            values[ProjectConstants.ROLEINDEX]));
        map.put(new IntWritable(ProjectConstants.EMPLOYEEIDINDEX),
new Text(
            values[ProjectConstants.EMPLOYEEIDINDEX]));
        ImmutableBytesWritable outKey = new ImmutableBytesWritable(
            Bytes.toBytes(values[ProjectConstants.IDINDEX]));
        context.write(outKey, map);
    }
}

```

#### 5.4.1 Employee Table

The Employee table contains a list of competence of each employee. Employee\_Id is taken as the row key and Column family has three columns:

- Competence: lists all the competence of an employee.  
Column name is "Competence", qualifiers names are list of competences an employee has and value is rating of competence rated by employees.

- Info: contains information such as last updated, country, region, reporting level1 and reporting level2.  
Column name is "Info", qualifiers are "Country", "LastUpdated", "Region", "ReportingLevel1" and "ReportingLevel2", and values are country, last updated date, region, name of reporting level1, name of reporting level2 respectively.
- Project: gives the Project\_Id of project an employee is currently assigned to.  
Column name is "Project", qualifier name is Project\_Id and value is type of project.

Table 4 shows the employee table schema.

Table 4. Employee Table Schema

Row	Column Families		
	Info	Competence	Project
<Employee_ID>	Info:Country=value Info:Region=value Info:LastUpdated =value Info:ReportingLevel1=value Info:ReportingLevel2=value	Competence:<Competence_Name>=rating	Project:<Project_ID>= type
12345	Info:Country=Finland Info:Region= Finnish Region Info:LastUpdated=7/2/2009 8:24 Info:ReportingLevel1=Industries Info:ReportingLevel2=TGF Financial Services	Competence:Java=3 Competence:PHP =3 Competence: J2EE_JSP=4 Competence:SQL=3	Project:1=internal

## Creating Employee Table

Below is the snippet of source code in Java for creating Employee table in HBase:

```

private static Boolean createTableInHBase(HBaseAdmin hbase, String tableName)
{
    Boolean result = false;

    HTableDescriptor desc = new HTableDescriptor(tableName);
    desc.addFamily(new HColumnDescriptor(Bytes
        .toBytes(Constants.employeeTcolumnName[0]), 15,
        HColumnDescriptor.DEFAULT_COMPRESSION,
        HColumnDescriptor.DEFAULT_IN_MEMORY,
        HColumnDescriptor.DEFAULT_BLOCKCACHE,
        HColumnDescriptor.DEFAULT_TTL, false)); // general info
    desc.addFamily(new HColumnDescriptor(Bytes
        .toBytes(Constants.employeeTcolumnName[1]), 15,
        HColumnDescriptor.DEFAULT_COMPRESSION,
        HColumnDescriptor.DEFAULT_IN_MEMORY,
        HColumnDescriptor.DEFAULT_BLOCKCACHE,
        HColumnDescriptor.DEFAULT_TTL, false)); // competence ID and
rating
        desc.addFamily(new HColumnDescriptor(Bytes
            .toBytes(Constants.employeeTcolumnName[2]), 15,
            HColumnDescriptor.DEFAULT_COMPRESSION,
            HColumnDescriptor.DEFAULT_IN_MEMORY,
            HColumnDescriptor.DEFAULT_BLOCKCACHE,
            HColumnDescriptor.DEFAULT_TTL, false)); // project ID and
type
        try {
            hbase.createTable(desc);
            System.out.println("SUCCESS: Table was created
successfully");
            result = true;
        } catch (IOException e) {
            System.out.println("Wasn't able to create a table: '" +
tableName
                + "' in HBase");
            result = false;
        }

        return result;
    }
}

```

Figure 19 shows Employee table in HBase

Home CloudInstance List New CloudInstance

SCAN from LocalHost

Table: Employee RowKey:

Versions: 100 Rows: 100

Scan

RowKey/Timestamp	Competence	Info	Project
<u>13297</u> Show all 2 Timestamps	<b>Communication &amp; co-operation:</b> 2 <b>English:</b> 2 <b>Experience in cross border projects:</b> 3 <b>Finnish:</b> 5 <b>German:</b> 1 <b>Intercultural skills:</b> 2 <b>Life Insurance:</b> 4 <b>Life Insurance_Claims Handling:</b> 2 <b>Life Insurance_United Linked insurances:</b> 4 <b>On-line communications:</b> 3 <b>Pension Insurance:</b> 3	<b>Country:</b> Finland <b>LastUpdated:</b> 7/2/2009 8:24 <b>Region:</b> Finnish Region <b>ReportingLevel1:</b> Industries <b>ReportingLevel2:</b> TGF Financial Services	1: customer
<u>17684</u> Show 1 Timestamp	<b>English:</b> 4 <b>Finnish:</b> 5 <b>Genesys:</b> 1 <b>German:</b> 1 <b>MS Access:</b> 1 <b>MS SQL server:</b> 2 <b>DMT:</b> 4 <b>Swedish:</b> 2	<b>Country:</b> Finland <b>LastUpdated:</b> 11/26/2009 8:10 <b>Region:</b> Finnish Region <b>ReportingLevel1:</b> Industries <b>ReportingLevel2:</b> TGI Industry Group	1: internal
<u>61023</u> Show 1 Timestamp	<b>Finnish:</b> 5 <b>J2EE_JSP:</b> 2 <b>PHP:</b> 3 <b>SQL:</b> 3 <b>Sw engineering_SEI/SAPP:</b> 1 <b>Swedish:</b> 4 <b>XML_XML:</b> 2	<b>Country:</b> Finland <b>LastUpdated:</b> 4/1/2009 15:54 <b>Region:</b> Finnish Region <b>ReportingLevel1:</b> Service Lines <b>ReportingLevel2:</b> TGS Service Lines	1: customer

Figure 19. HBase Employee Table

As illustrated in figure 19, based on schema in table 4 the Employee table is created in HBase using the mapreduce framework using java as programming language.

#### 5.4.2 Competence Table

The Competence table contains a list of competences, its hierarchy using parent-child relationship and a list of employees belonging to that competence, along with general information. The competence name is taken as the row key. Table 5 shows the Competence table schema.



Table 5. Competence Table Schema

Row	Column Families			
	Info	Parent	Child	Employee
<CompetenceName>		Parent:<Competence_ID>=Competence_Level	Child:<Competence_ID>=Competence_Level	Employee:<Employee_ID>=rating
General	Info:Level=L1		Child:Communication=L2 Child:Language=L2 Child:English=L3	
Java	Info:Level=L3	Parent:SW Development=L1 Parent: Programming languages=L2		12234=3 23456=4

The Column Family has four columns as summarized in table 5:

- Info: indicates the level of competence.  
Column name is "Info", qualifier is "Level" and value is type of level.
- Parent: lists all upper level hierarchy of the competence.  
Column name is "Parent", qualifiers are upper hierarchy name list and value is the level of parents.
- Child: lists all lower level hierarchy of the competence.  
Column name is "Child", qualifiers are lower hierarchy name list and value is the level of child in hierarchy.

- Employee: lists all employees under the competence. Column name is "Employee", qualifier is employee\_Id and value is rating of competence rated by employee.

RowKey/Timestamp	Child	Employee	Info	Parent
<a href="#">Business processes</a> Show 1 Timestamp	English: L3 Finnish: L3 German: L3 ITOMS: L2 OMT: L3 Swedish: L3		Level: 1	
<a href="#">Certificates</a> Show 1 Timestamp	Finnish: L3 Sw engineering_SEI/SAPP: L3		Level: 2	General: 1
<a href="#">Communication</a> Show 1 Timestamp	Communication & co-operation: L3 On-line communications: L3		Level: 2	General: 1
<a href="#">Communication &amp; co-operation</a> Show 1 Timestamp		13297: 2	Level: 3	Communication: L2 General: L1
<a href="#">Customer Interaction Management</a> Show 1 Timestamp	Genesys: L3		Level: 2	Technologies & Platforms: 1
<a href="#">Databases</a> Show 1 Timestamp	MS Access: L3 MS-SQL server: L3		Level: 2	Technologies & Platforms: 1
<a href="#">English</a> Show 1 Timestamp		13297: 2 17684: 4	Level: 3	General: L1 Language: L2
<a href="#">Experience in cross border projects</a> Show 1 Timestamp		13297: 3	Level: 3	General: L1 International: L2
<a href="#">Financial Services</a> Show 1 Timestamp	Life Insurance: L3 Life Insurance_Claims Handling: L3 Life Insurance_United Linked insurances: L3 Pension Insurance: L3		Level: 2	Industry: 1
<a href="#">Finnish</a> Show 1 Timestamp		13297: 5 17684: 5 61023: 5	Level: 3	General: L1 Language: L2
<a href="#">General</a> Show 1 Timestamp	Certificates: L2 Communication: L2 Communication & co-operation: L3 English: L3 Experience in cross border projects: L3 Finnish: L3 German: L3		Level: 1	

Figure 20. HBase Competence Table

Figure 20 shows the Competence table in HBase.

### 5.4.3 Project Table

The Project table contains a list of projects, general information of the project and a list of employees to that project. Project\_Id is taken as the row key. Table 6 gives the schema of Project table.

Table 6. Project Table Schema

Row	Column Families	
	Info	Member:
<Project_ID>	Info: start_date=value Info: End_date=value	Member: <Employee_ID>=role
66476	Info:start_date=1.1.2009 Info:End_date=10.1.2015	Member: 16115=Consultant(A-) Member: 147968=Developer(D+) Member: 77421=Consultant(A-)

The Column family has two columns as summarized in table 6:

- Info: contains project information such as project start date, project end date.  
Column name is "Info", qualifiers are "ProjStartDate" and "ProjEndDate" and values are start date of project, end date of project respectively.
- Member: lists all employees of the project.  
Column name is "Member", qualifier is list of employee\_ids in the project and value is type of role in the project.

### Loading of Project Table

The snippet of the source code for loading the project data to the Project table is given in the next page. The implementation is done in the reduce part of the MapReduce framework.

```

/**
 * Processes the map and writes the data to the HBase.
 */
public static class ReduceToTable extends
TableReducer<ImmutableBytesWritable, MapWritable, Text> {
    HTable table ;
    @Override
    protected void reduce(ImmutableBytesWritable key,
        Iterable<MapWritable> values, Context context) throws
IOException,
        InterruptedException {
        table = new HTable(ProjectConstants.tableName);
        Put put = new Put(key.get());
        long timeStamp = System.currentTimeMillis();

        for (MapWritable value : values) {
            Iterator<Writable> iterator =
value.keySet().iterator();

            String pro_startDate = "";
            String pro_endDate = "";
            String employee_id = "";
            String role = "";

            while (iterator.hasNext()) {
                Writable mapValue = iterator.next();
                int next = ((IntWritable) mapValue).get();
                switch (next) {
                    case
ProjectConstants.PROJECTSTARTDATEINDEX:
                        pro_startDate = ((Text)
value.get(new IntWritable(
                            ProjectConstants.PROJECTSTARTDATEINDEX))).toString();

                        break;
                    case
ProjectConstants.PROJECTENDDATEINDEX:
                        pro_endDate = ((Text) value.get(new
IntWritable(
                            ProjectConstants.PROJECTENDDATEINDEX))).toString();

                        break;
                    case ProjectConstants.EMPLOYEEIDINDEX:
                        employee_id = ((Text) value.get(new
IntWritable(
                            ProjectConstants.EMPLOYEEIDINDEX))).toString();

                        break;
                    case ProjectConstants.ROLEINDEX:
                        role = ((Text) value.get(new
IntWritable(
                            ProjectConstants.ROLEINDEX))).toString();

```

```

                break;
            }
        }

        if(put!=null){
            if(put.isEmpty()){

                put.add(Bytes.toBytes(ProjectConstants.columnName[ProjectConstants.FAMILYNAMEINFO]),

                    Bytes.toBytes(ProjectConstants.qualifiers[ProjectConstants.QUALIFIERPROJECTSTARTDATE]),

                    System.currentTimeMillis(),

                    Bytes.toBytes(pro_startDate));

                put.add(Bytes.toBytes(ProjectConstants.columnName[ProjectConstants.FAMILYNAMEINFO]),

                    Bytes.toBytes(ProjectConstants.qualifiers[ProjectConstants.QUALIFIERPROJECTENDDATE]),

                    System.currentTimeMillis(),

                    Bytes.toBytes(pro_endDate));

                put.add(Bytes.toBytes(ProjectConstants.columnName[ProjectConstants.FAMILYNAMEMEMBER]),

                    Bytes.toBytes(employee_id),

                    System.currentTimeMillis(),

                    Bytes.toBytes(role));

                table.put(put);

            }else {
                timeStamp = timeStamp + 11;
                if(!employee_id.isEmpty()){
                    put = new Put(key.get());

                    put.add(Bytes.toBytes(ProjectConstants.columnName[ProjectConstants.FAMILYNAMEMEMBER]),

                        Bytes.toBytes(employee_id),

                        timeStamp,

                        Bytes.toBytes(role));

                    table.put(put);
                }
            }
        }
    }
}

```

Figure 21 shows Project table created in HBase based on the above code snippet.

# HBase Explorer

[Home](#)
[CloudInstance List](#)
[New CloudInstance](#)

SCAN from *LocalHost*

Table	<input type="text" value="Project"/>	RowKey	<input type="text"/>
Versions	<input type="text" value="100"/>	Rows	<input type="text" value="100"/>

 Scan

RowKey/Timestamp	Info	Member
<b>100002</b> Show 1 Timestamp	ProjEndDate: 28.2.2011 ProjStartDate: 1.7.2010	<b>145468</b> : Consultant(A-)
<b>100003</b> Show 1 Timestamp	ProjEndDate: 31.12.2010 ProjStartDate: 1.7.2010	<b>23365</b> : Developer(D+) <b>72622</b> : Developer(D+)
<b>100004</b> Show 1 Timestamp	ProjEndDate: 30.9.2010 ProjStartDate: 1.7.2010	<b>17122</b> : Senior Consultant(A)
<b>100005</b> Show 1 Timestamp	ProjEndDate: 28.2.2011 ProjStartDate: 1.7.2010	<b>108197</b> : Developer(D+) <b>97576</b> : Developer(D+)
<b>100006</b> Show 1 Timestamp	ProjEndDate: 30.9.2010 ProjStartDate: 1.7.2010	<b>76323</b> : Project Manager
<b>100014</b> Show 1 Timestamp	ProjEndDate: 31.3.2011 ProjStartDate: 1.6.2010	<b>16497</b> : Consultant(A-) <b>16968</b> : Consultant(A-)

Figure 21. Project Table

Based on these table schemas Employee, Competence and Project tables were created in HBase using Java. Then data was loaded to these tables using competence data and project data files.

```

Version: 0.20.6, r965666, Mon Jul 19 16:54:48 PDT 2010
hbase(main):001:0> list
Competence
Employee
Project
3 row(s) in 0.6560 seconds
hbase(main):002:0> █

```

The screenshot shows the HBase Explorer interface. At the top, there is a navigation bar with 'Home', 'Hbase Source List', and 'New Hbase Source'. Below this is a 'Show HbaseSource' section with the following details:

- Name: LocalHost: Id: 1
- Quorum Port: 2,181
- Quorum Servers: 127.0.0.1
- Master Uri: http://localhost

The main area displays a grid of table information. Each table entry includes a 'Disable Table' button, '1 Region(s)', 'Clone', and 'Statistics' (V, A, B, C, M, BF). The tables listed are:

- Competence**: Child, Employee, Info, Parent
- Employee**: Competence, Info, Project
- Level1group**: Level2, Level3
- Project**: Info, Member

A 'Scan MASTER New Table' button is visible on the right side of the grid.

Figure 22. List of HBase Tables

Figure 22 shows a list of tables in HBase using HBase explore and putty.

## 5.5 Using RESTful Services

Basic database queries following CRUD pattern were implemented as REST Services. A RESTful Web service was developed on Java platform with the JAX-RS reference implementation Jersey. Eclipse was used as a development environment and Tomcat as servlet container. The actual versions of these products were Java 1.6, Tomcat 6.0 and JAX-RS 1.1. (Jersey 1.4).

In REST architecture, everything is a resource. A resource is accessed via common interface based on the HTTP standard methods. REST server provides access to the resources and a REST client accesses and modifies the REST resources. Server must support the HTTP common operations (GET, POST, PUT and DELETE). Resources are identified by global IDs (which are typically URIs). [33]

It typically defines the base URI for the services, the MIME-types its supports (XML, JSON, text). Here are the URIs and HTTP methods that correspond to the resources in the application:

Resource	URI Path	HTTP Methods
ProfileService	/profiles	GET, POST, POST, DELETE

### 5.5.1 Server

Server `com.tieto.reloud.hbase.restservices.server` contains the following Java classes as shown in figure 23:

- ProfileService  
This class registers itself as CRUD resources via `@PUT`, `@GET`, `@POST` and `@DELETE` annotation respectively. Via the `@Produces` annotation it defines that it delivers MIME type "text" code. It also defines via the "Path" annotation that its service should be available under the URL "profiles".
- HBaseHandler  
This class acts as handler to access the table from HBase. This class process HBase table.
- ProfileDataObject  
Creates profile in object form.
- ProfileDataSet  
Creates a list of profiles from Row\_Id

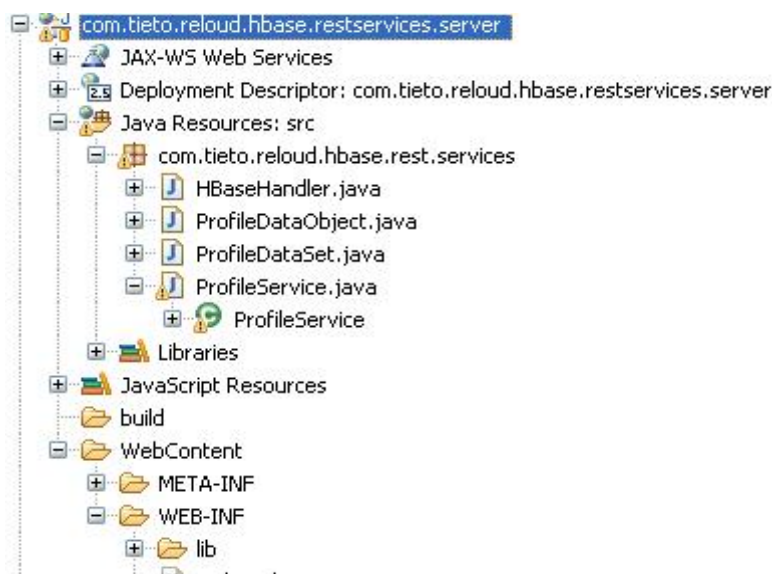


Figure 23. REST server classes



Here is a snippet of the source code from the ProfileService class:

```

/**
 * Rest web-service for profiles' CRUD operations. The services are accessed
 * with http://baseURI/restservice/profiles for getting all profiles' IDs,
 * create or update a profile or via
 * http://baseURI/restservice/profiles/profileID for getting and a deleting a
 * profile
 */

@Path("/profiles")
@Produces(MediaType.TEXT_XML)
@Consumes(MediaType.TEXT_XML)
public class ProfileService {

    private ProfileDataSet profiles;
    private ProfileDataObject profile;
    @Context
    private final UriInfo uriInfo;
    @Context
    private final Request request;
    String id;

    public ProfileService(UriInfo uriInfo, Request request, String id) {
        this.uriInfo = uriInfo;
        this.request = request;
        this.id = id;
    }
    /**
     * @return {@link Response} instance with {@link ProfileDataSet}
     * instance or 404 in case of failure to fetch any profiles' IDs
     */
    @GET
    public Response getAllProfiles() {
        Response res;
        profiles = HBaseHandler.getInstance().getAllProfiles();
        if (!profiles.isEmpty()) {
            res = Response.ok().entity(profiles).build();
        } else {
            res = Response.status(Status.NOT_FOUND)
                .entity("Couldn't find any profile!").build();
        }
        return res;
    }
    /**
     * @return {@link Response} instance with {@link ProfileDataObject}
     * instance or 404 in case of failure to fetch any profiles' IDs
     */
    @GET
    @Path("/{profileID}")
    public Response getProfile(@PathParam("profileID") int id) {
        Response res;
        profile = HBaseHandler.getInstance().getProfile(id);
        if (profile.getID() != 0) {
            res = Response.ok().entity(profile).build();
        } else {
            res = Response.status(Status.NOT_FOUND)

```

```

        .entity("Target profile doesn't exist!").build();
    }
    return res;
}

/**
 * @return {@link Response} instance with 200 if profile created
 *         successfully or 403 otherwise
 */
@POST
public Response postProfile(ProfileDataObject object) {
    Response res;
    boolean success = HBaseHandler.getInstance().createProfile(object);
    if (success) {
        res = Response.ok().build();
    } else {
        res = Response.status(Status.FORBIDDEN)
            .entity("Target profile already exist!").build();
    }
    return res;
}

/**
 * @return {@link Response} instance with 200 if profile updated
 *         successfully or 403 otherwise
 */
@PUT
public Response updateProfile(ProfileDataObject object) {
    Response res;
    boolean success = HBaseHandler.getInstance().updateProfile(object);
    if (success) {
        res = Response.ok().build();
    } else {
        res = Response.status(Status.FORBIDDEN).build();
    }
    return res;
}

/**
 * @return {@link Response} instance with 200 if profile deleted
 *         successfully or 403 otherwise
 */
@DELETE
@Path("/{profileID}")
public Response deleteProfile(@PathParam("profileID") int id) {
    Response res;
    boolean success = HBaseHandler.getInstance().deleteProfile(id);
    if (success) {
        res = Response.ok().build();
    } else {
        res = Response.status(Status.FORBIDDEN)
            .entity("Target profile does not exist!").build();
    }
    return res;
}
}
}

```

## ProfileService Methods

The `ProfileService` class has five methods for processing the resources in RESTful way:

- `getAllProfiles` retrieves all profiles from the HBase table;
- `getProfile` retrieves all information of a particular Id;
- `postProfile` is for adding new profile into the table;
- `updateProfile` is for updating existing profile from the table;
- `deleteProfile` is for deleting existing profile from the table.

The `@Produces` annotation specifies the MIME type. Here, the annotation specifies that it returns a text:

```
@Produces(MediaType.TEXT_XML)
```

### 5.5.2 Client

The Client has been created with namespace `com.tieto.reloud.hbase.restservices.client` and includes the following Java classes as shown in figure 24:

- `ProfileRestClient`  
It is client to the server to access resources and return response code.
- `ProfileDataObject`  
Creates profile in object form.
- `ProfileDataSet`  
Creates a list of profiles from Row\_Id

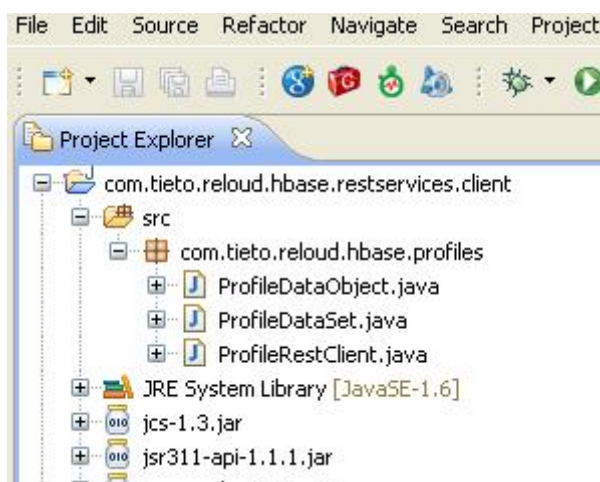
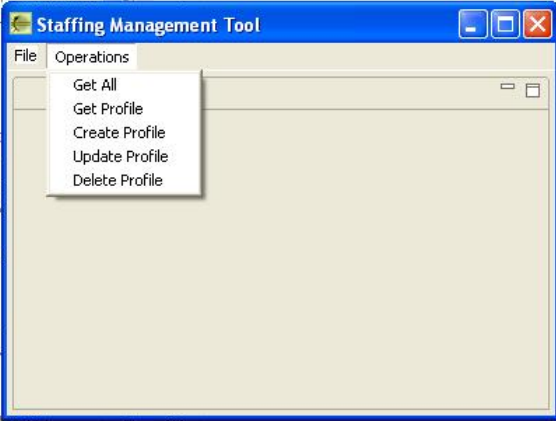
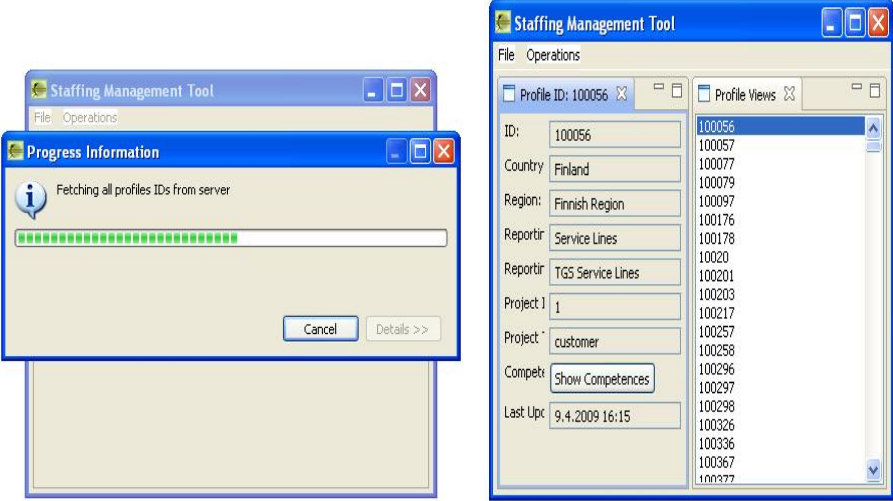
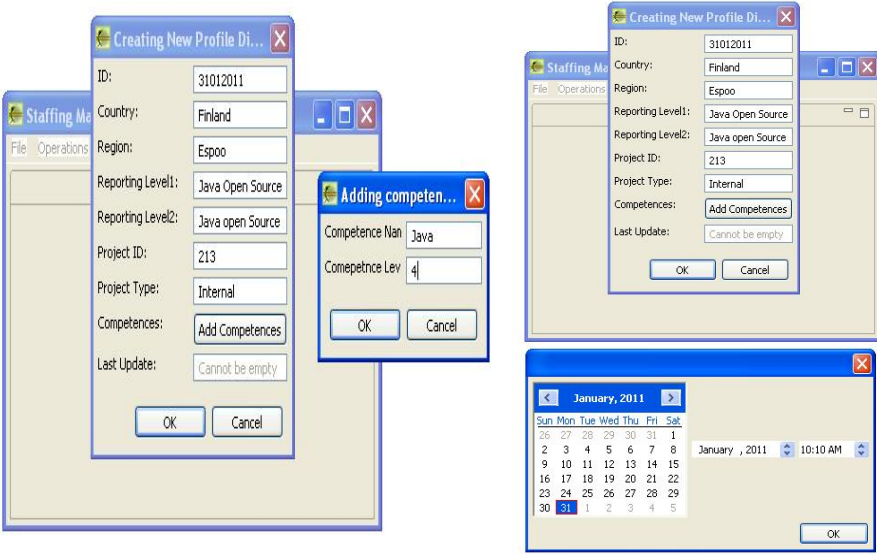
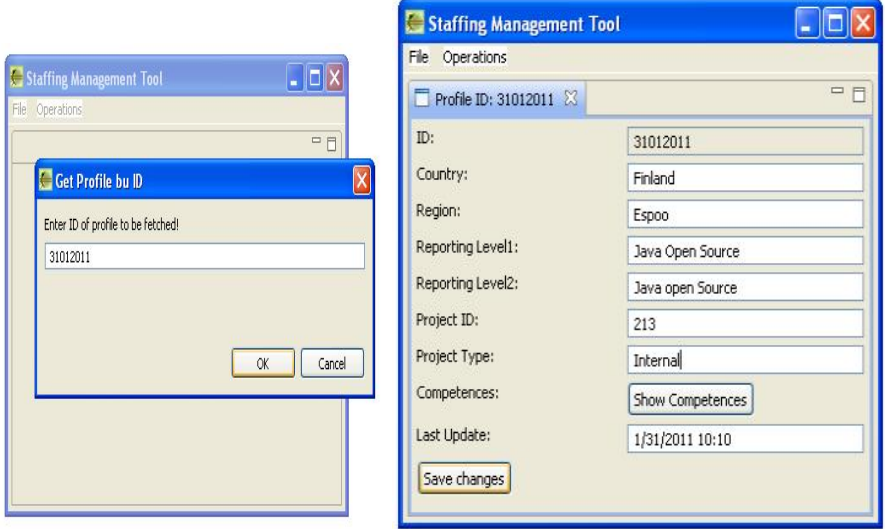
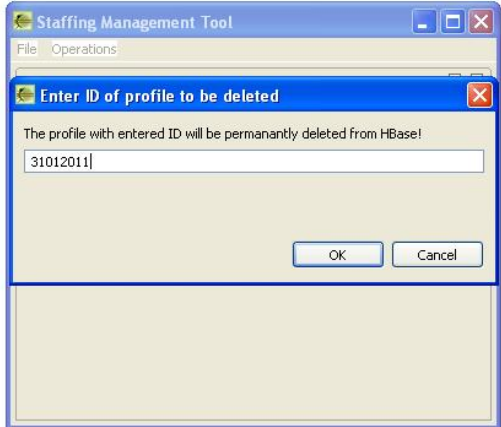


Figure 24. REST Client classes

Table 7 shows the CRUD operations implemented using REST.

Table 7. GUI CRUD operations

1	GUI	
2	Get All Profile	
3	Create new Profile	

4	Update id 31012011	 <p>The screenshot shows two windows from the 'Staffing Management Tool'. The background window is the main profile details form, titled 'Profile ID: 31012011'. It contains fields for ID (31012011), Country (Finland), Region (Espoo), Reporting Level1 (Java Open Source), Reporting Level2 (Java open Source), Project ID (213), Project Type (Internal), Competences (Show Competences), and Last Update (1/31/2011 10:10). A 'Save changes' button is at the bottom. Overlaid on top is a smaller dialog box titled 'Get Profile by ID' with the text 'Enter ID of profile to be fetched!' and a text input field containing '31012011'. It has 'OK' and 'Cancel' buttons.</p>
5	Delete Id 31012011	 <p>The screenshot shows the 'Staffing Management Tool' with a dialog box titled 'Enter ID of profile to be deleted'. The dialog contains the text 'The profile with entered ID will be permanently deleted from HBase!' and a text input field with '31012011'. It has 'OK' and 'Cancel' buttons.</p>

As summarized in table 7, the graphical user interface of the CRUD operations can get all profiles, create a new profile, update an existing profile and delete an existing profile.

## 5.6 Data Mining

### 5.6.1 Competence Data Analysis

Total number of employee in data set is 15480. Total number of competences in dataset is 3516. Total number of competences of all employees in dataset is 361579. The above data set concludes that on an average every employee has 23 competences and the dimension of the feature vectors (one entry for every possible competence) is

3516, so the data has the potential of being quite sparse. Figure 25 shows frequency of the occurrences of competences.

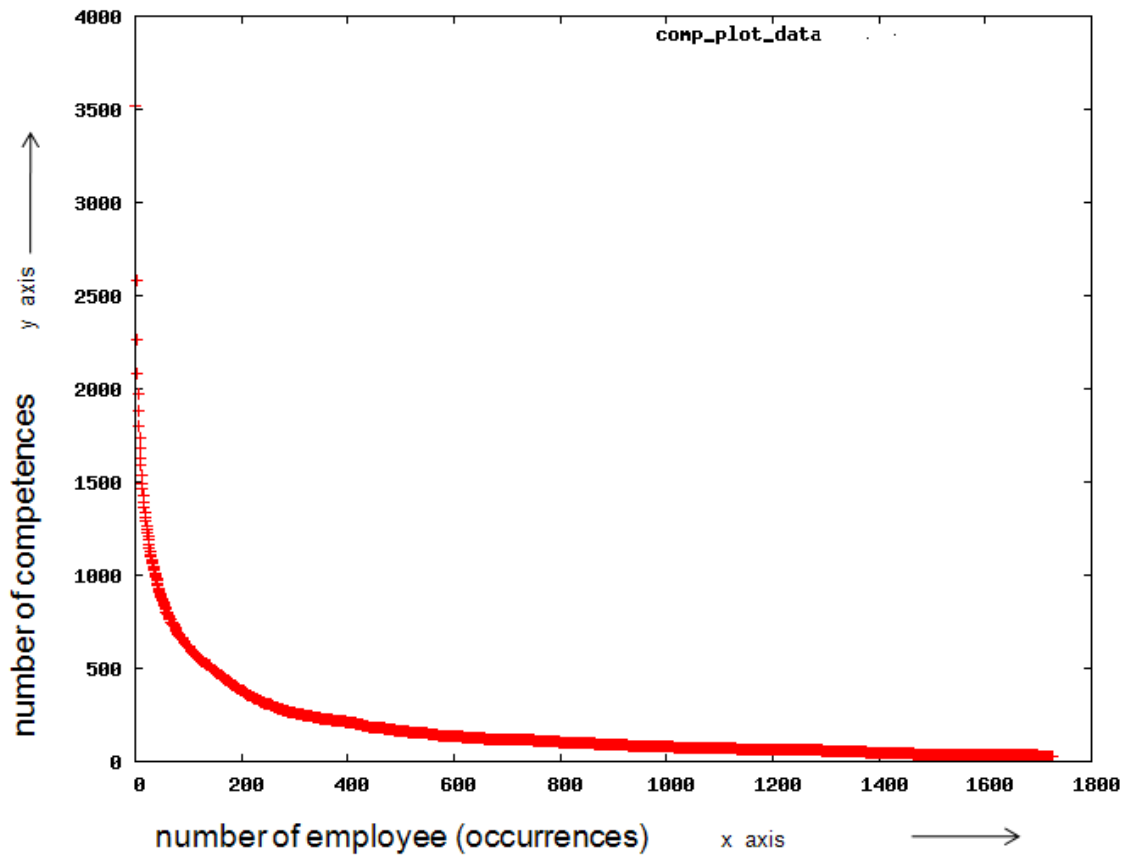


Figure 25. Competence plot

From figure 25 it is clear that the majority of competences are quite rare. From the plot, value  $(x, y) = (1, 3516)$  means that 3516 competences occur at least once in the data set.

So, if a threshold is set of 20 occurrences then the rarest 2/3 of all competences can be eliminated from the data set, which could help quite a bit in clustering.

Figure 26 shows the distances of all employees with respect to the one employee id.

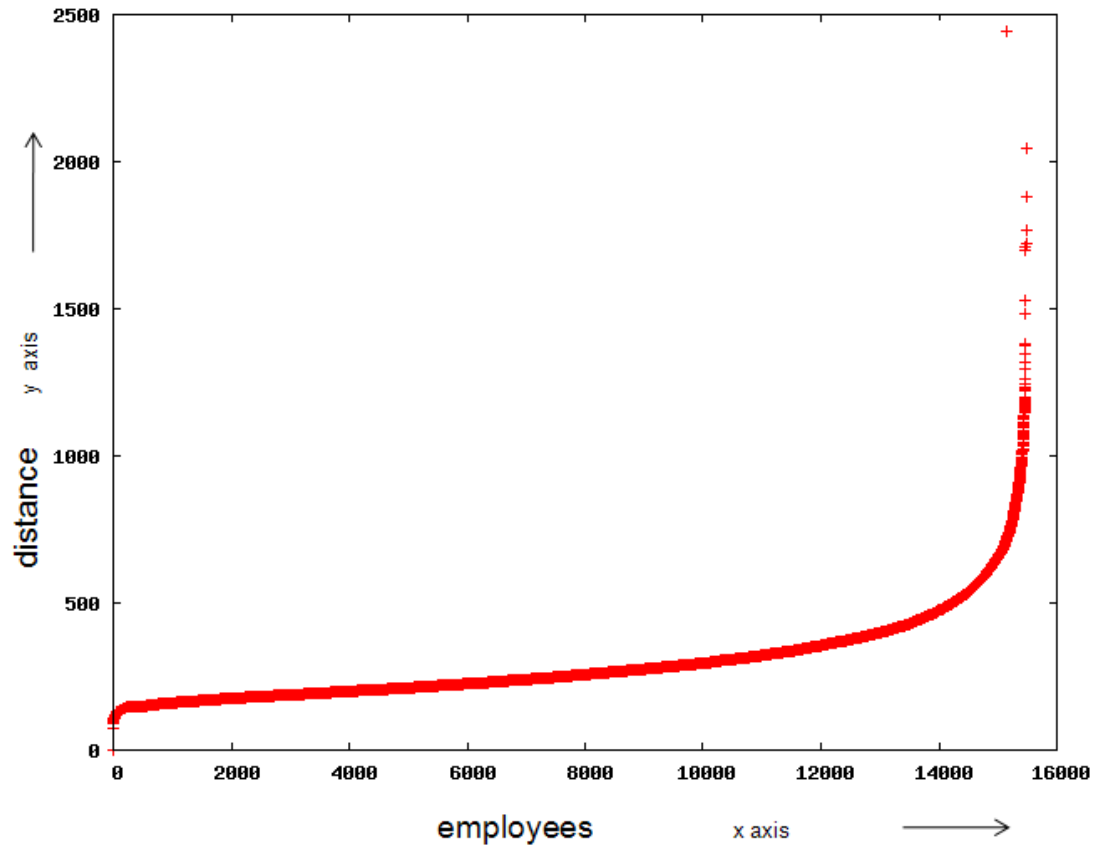


Figure 26. Distances of all employees with respect to the one id

Figure 26 shows that distances in high-dimensional space are not really informative for these "competence points". The search for employee needs to be restricted to relevant regions in the dataset. A mixture of hard and soft constraints could help the search for relevant employee.

Figure 27 shows the distribution of the number of competences over employee population.

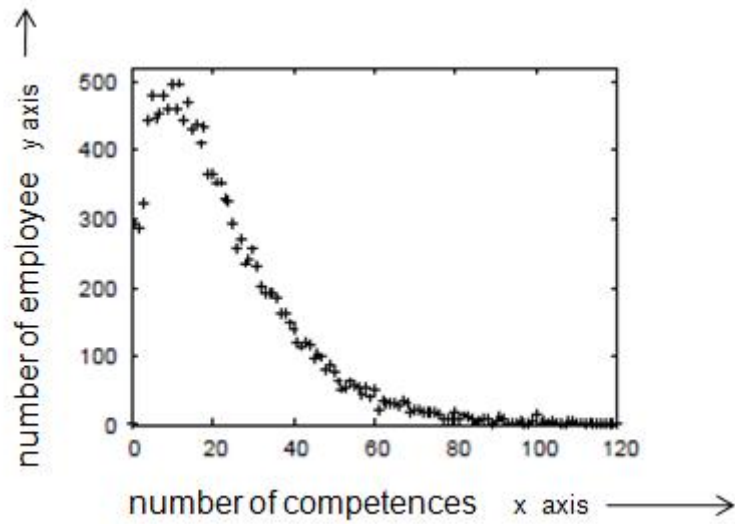


Figure 27. Number of competence over employee population

Figure 27 illustrates that median is 18, average is 23.35 and maximum is 266.

### 5.6.2 Project Data Analysis

There are around 7000 employees assigned to various projects in Tieto. Figure 28 shows the cumulative distribution for the total number of projects a person has been assigned to (not necessarily at the same time of course).



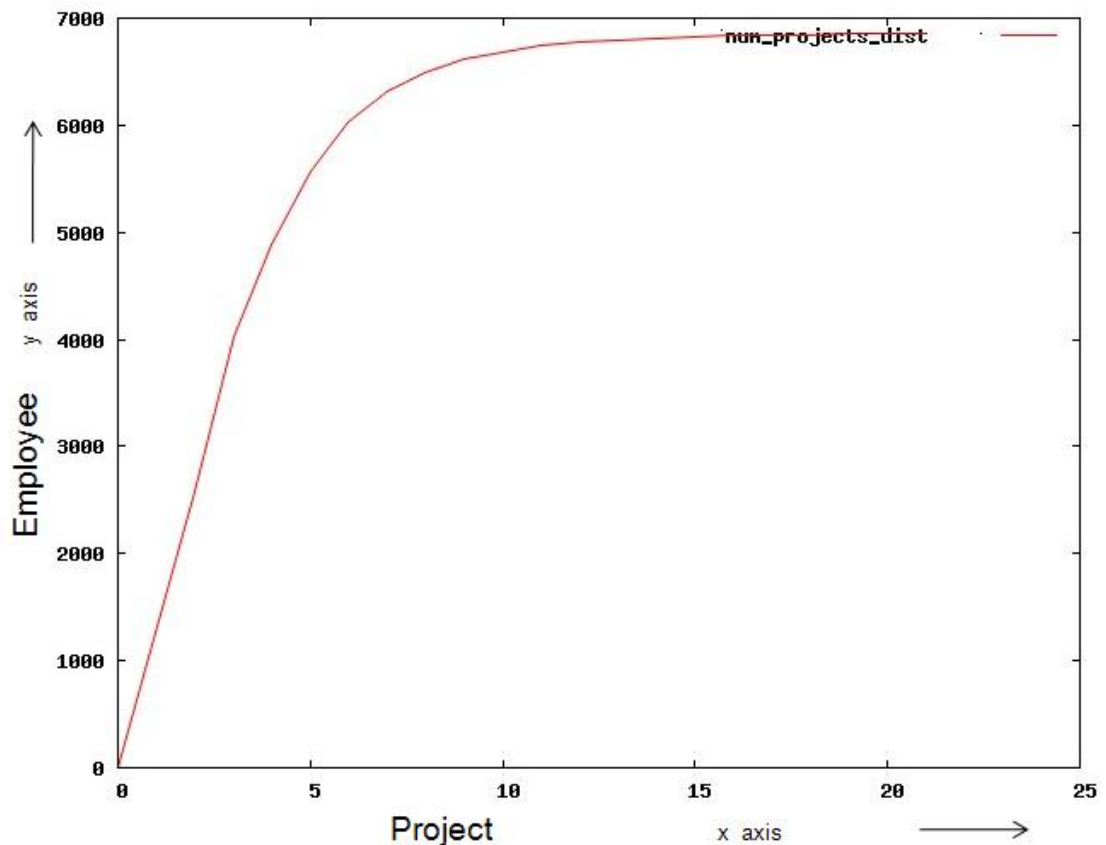


Figure 28. Distribution of number of projects a person has been assigned to

As shown in figure 28, around 85% of the employees have been assigned to less than five projects in total but there are a few people who were assigned to more than twenty projects. About 80% of the employee ids in the project data set are also in the competence data set. The employees that are assigned to 20+ projects are likely line managers or other formal owners of the project.

So, search results are only limited to people with some sort of developer role. Seeing 20+ projects for any person means that he does not have anything real to do with the project.

These are the different roles in the dataset:

- Absence
- Consultant(A-)
- Delivery Owner
- Developer(D+)
- Implementation(D)

- Management Consultant(A+)
- Planning, Customer Support(C)
- PRM Project Manager(B)
- Program Manager
- Project Assistant
- Project Leadership(B+)
- Project Manager
- SAP Developer(D+ +)
- Senior Consultant(A)
- Subcontractor(S1)
- Subcontractor(S2)
- Subcontractor(S3)
- System Specialist(D-)

The following roles are considered while processing the data:

- Consultant(A-)
- Developer(D+)
- Implementation(D)
- Management Consultant(A+)
- Senior Consultant(A)
- Subcontractor(S1)
- Subcontractor(S2)
- Subcontractor(S3)

If one could somehow identify "replacement events" in the project database (person A leaves, person B joins directly afterwards with the same role). This implies that A and B have similar competences then one could use this information to fine-tune the distance metric for KNN and also for the recommendation engine.

### 5.6.3 Recommendation

#### Identifying replacement events

The search is restricted to the roles mentioned above and also when there are potential replacements where person B replaces a "large" number of other employees in the same project (these assignment dates seem to be clustered) then all these replacements are rejected and also when they exceed a certain number (here: max 2 replacements). There is an indegree limit for replacements: when in the same project the same person was potentially replaced by a "large" number of other employees, all of these are rejected and if the number is large than the limit (here: again 2).

By this method replacement event comes out to be 130. For every one of these (tuple (A, B, project)) one can then determine the relative position of B within the neighborhood of A, i.e. by sorting the whole population with respect to distance to A and checking at which position B is. Here the intuition would be that B would tend to be somewhere close to A in the competence space.

Finding replacement event also makes use of the competence tree (comparing competence prefixes). Further, it is normalized in order to try to avoid problems when either A or B has a lot of competence records, which the other one has not.

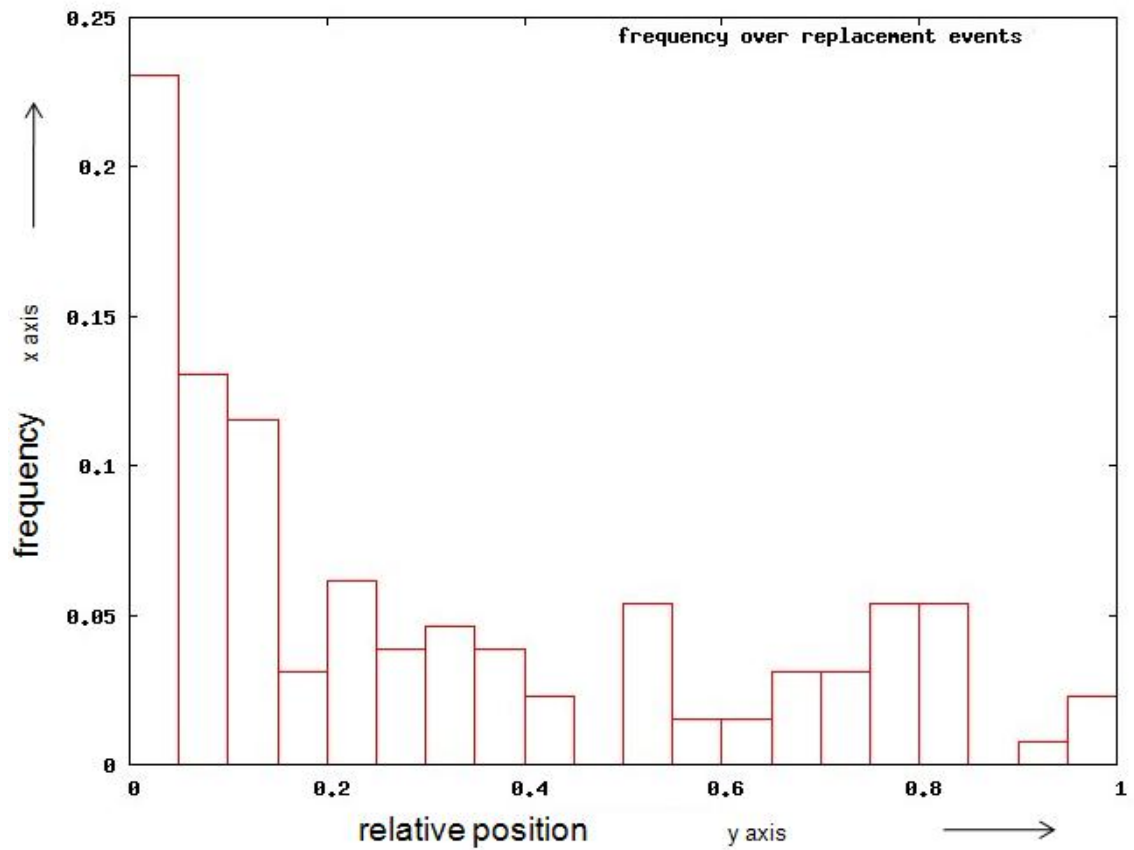


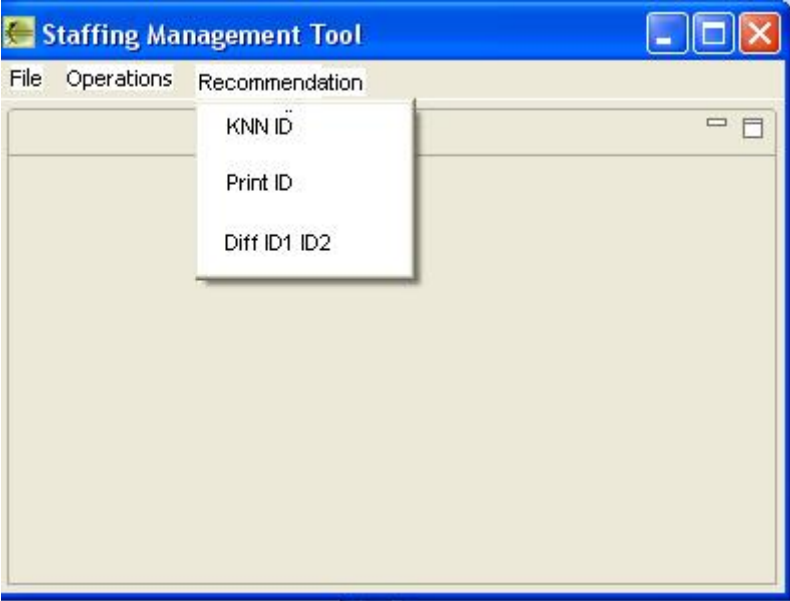
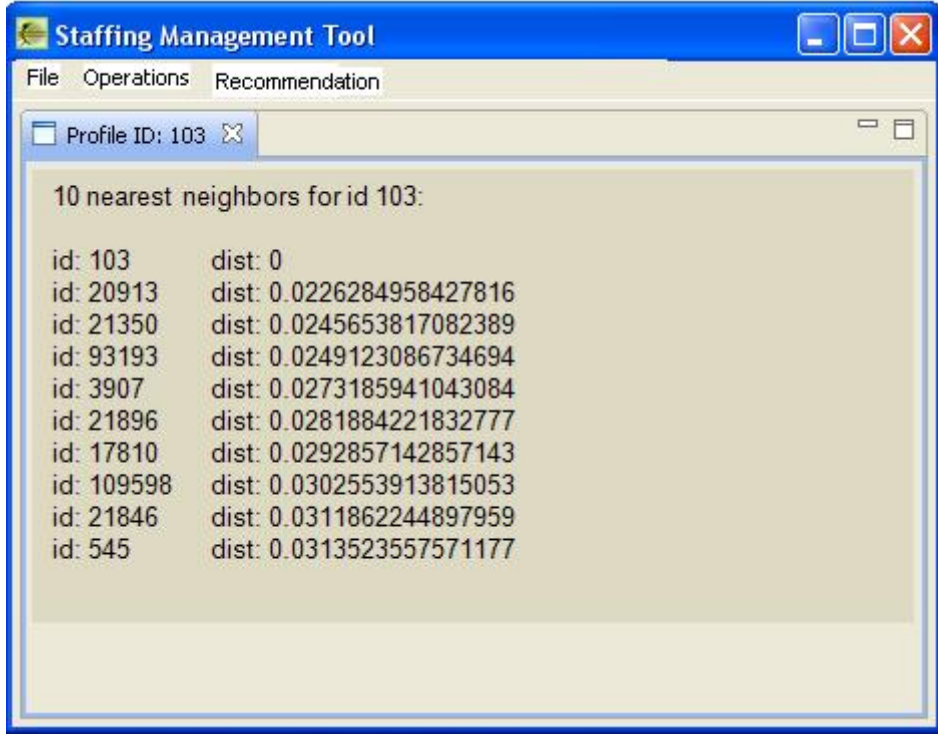
Figure 29. Relative distance histogram

Figure 29 shows the relative position of B to A (1.0 means the employee id furthest away from A, something very close to zero means very close compared to the other employee in the competence database).

Basically the plot shows that around 23% of the 130 replacement cases, the relative distance of B to A was below 0.05. So, the project data and the competence data are meaningful for comparisons.

Table 8 shows recommendation results based on KNN queries to the competence data. Along with KNN queries it also shows the list of competences and differences between two employees' competences.

Table 8. Recommendation Results

Descriptions	GUI Results
<p>Recommendation operations:</p>	 <p>The screenshot shows the 'Staffing Management Tool' window with a menu open. The menu items are 'KNN ID', 'Print ID', and 'Diff ID1 ID2'.</p>
<p>"KNN ID" operation computes k nearest neighbors of ID (k=10 by default)</p>	 <p>The screenshot shows the 'Staffing Management Tool' window with a tab titled 'Profile ID: 103'. The main area displays the following text:</p> <pre> 10 nearest neighbors for id 103:  id: 103      dist: 0 id: 20913   dist: 0.0226284958427816 id: 21350   dist: 0.0245653817082389 id: 93193   dist: 0.0249123086734694 id: 3907    dist: 0.0273185941043084 id: 21896   dist: 0.0281884221832777 id: 17810   dist: 0.0292857142857143 id: 109598  dist: 0.0302553913815053 id: 21846   dist: 0.0311862244897959 id: 545     dist: 0.0313523557571177 </pre>

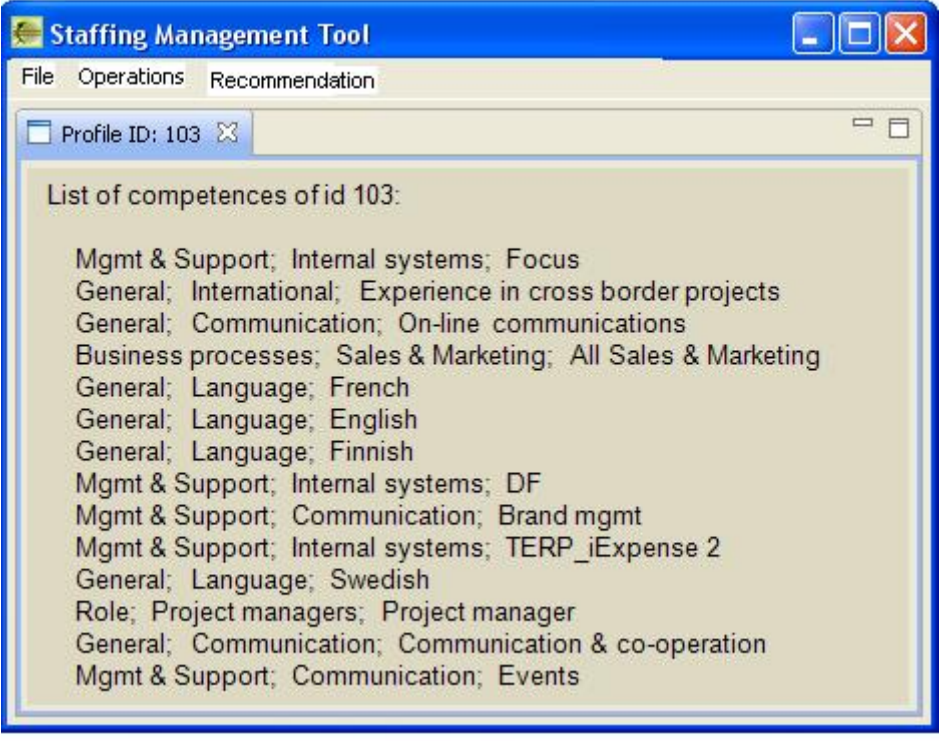
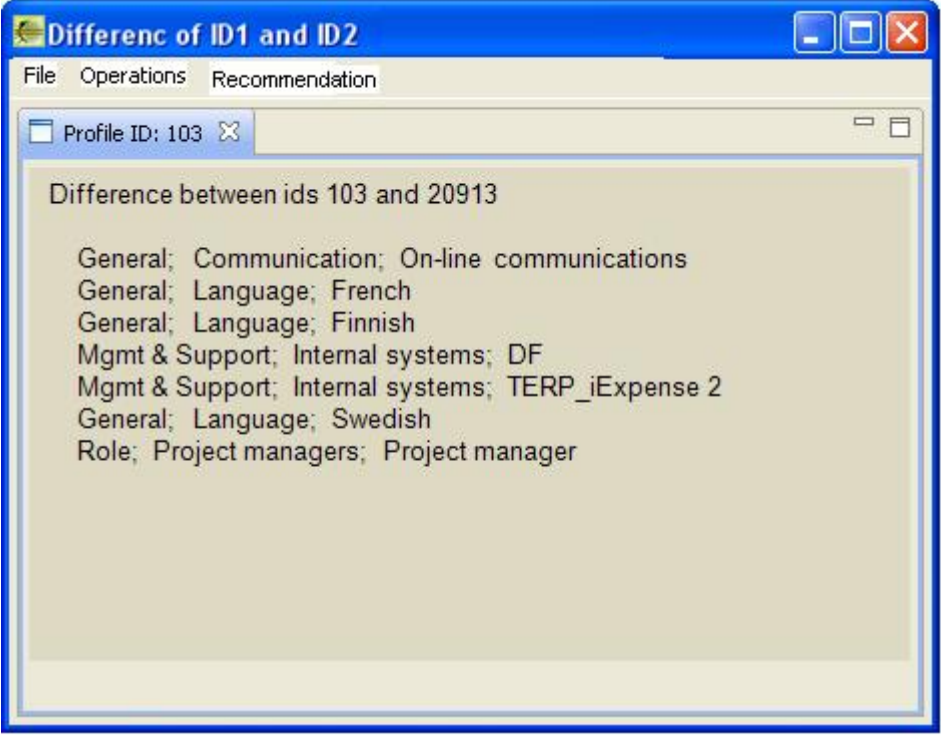
<p>“Print ID” operation prints competences for employee_ID</p>	 <p>The screenshot shows a window titled "Staffing Management Tool" with a menu bar containing "File", "Operations", and "Recommendation". Below the menu is a tab labeled "Profile ID: 103". The main content area displays the following text:</p> <pre>List of competences of id 103:  Mgmt &amp; Support; Internal systems; Focus General; International; Experience in cross border projects General; Communication; On-line communications Business processes; Sales &amp; Marketing; All Sales &amp; Marketing General; Language; French General; Language; English General; Language; Finnish Mgmt &amp; Support; Internal systems; DF Mgmt &amp; Support; Communication; Brand mgmt Mgmt &amp; Support; Internal systems; TERP_iExpense 2 General; Language; Swedish Role; Project managers; Project manager General; Communication; Communication &amp; co-operation Mgmt &amp; Support; Communication; Events</pre>
<p>“Diff ID1 ID2” operation shows differences between employee ID1 and ID2</p>	 <p>The screenshot shows a window titled "Differenc of ID1 and ID2" with a menu bar containing "File", "Operations", and "Recommendation". Below the menu is a tab labeled "Profile ID: 103". The main content area displays the following text:</p> <pre>Difference between ids 103 and 20913  General; Communication; On-line communications General; Language; French General; Language; Finnish Mgmt &amp; Support; Internal systems; DF Mgmt &amp; Support; Internal systems; TERP_iExpense 2 General; Language; Swedish Role; Project managers; Project manager</pre>

Table 8 illustrates the GUI implementation of recommendation and search results.

## 6 Conclusion

Hadoop is an open-source cloud computing implementation and provides all its necessary features such as scalability, reliability and fault tolerance. As one of the goals, this project explored the Hadoop and its architecture and how HDFS supports cloud computing features to deliver cloud services.

HBase is an open source, non-relational, distributed, column-oriented database model [34] also known as Hadoop database. It provides random, realtime read/write access to Big Data. HBase hosts very large tables - billions of rows X millions of columns – on top clusters of commodity hardware. HBase provides Bigtable-like capabilities on top of Hadoop. [35]

The main goal of the project was to build recommendation and search system for staffing using Cloud based technology Hadoop. So, Hadoop was installed on a four node cluster, and HBase was installed on top of Hadoop to provide scalable data storage.

In a competitive market, a software development organization's major goal is to maximize value creation for a given investment [31]. Therefore, resource management is a very crucial factor in enterprise-level value maximization. An efficient staffing system is required to balance between project technical needs and organizational constraints.

At Tieto Company, Recommendation and Search System is developed to fulfill the present demand of team-based work structures where all the jobs assigned to employees, not only with respect to fulfill all the assigned tasks but also considering candidates fitness with the team members in terms of interpersonal compatibility.

The system recommends a list of persons that can replace the person leaving the project. The recommendation system takes less time compared to the old search system which was based on the non-scalable Oracle database. Even a simple search query was taking much time to find from 15000 employees and 3000 competences. Now the new developed system is based on Hadoop. Data processing is done in the

Hadoop MapReduce framework which supports parallel processing. Thus, fetching any kind of search result from the HBase database is quite fast. Finding a person to replace the leaving person from the project is based on the distance-based algorithm KNN. First all the competences are clustered into groups and then the distance between the competences of the leaving employee and other available employees have been calculated in a required cluster. Those who are nearer to the leaving employee would be in recommendation list for replacement.

In this project, recommendation is given for finding a replacement for a leaving person from a project. The system also finds persons as per competences requirements. The recommendation system can be extended to recommend the formation of a new team. If a complete list of competence requirements for a project is given, then the system should suggest the team.

Current implementation is based on declared competences (the competences that are saved to the database by the employee) and project data (the information of the employee's current project and history of his/her previous project he/she was assigned to). In the case an employee missed to update his/her new competences to the database, then the competence data itself is not accurate. Then a recommendation based on this inaccurate data would result in inferior recommendation. In future work, this missing data of competence of an employee can be manipulated based on other employee's competence list, assuming employees working in the same project will also share the same competence. Also then a recommendation could be done with the latest manipulated competence data.



## References

- 1            10 reasons Why Cloud Computing is the Wave of the Future, April 29, 2009, by Gilberto J. Perera [online]  
URL: <http://laptoplogic.com/resources/10-reasons-why-cloud-computing-is-the-wave-of-the-future> [Accessed Dec, 2010]
- 2            Why Cloud Computing [online]  
URL: <http://www.appistry.com/cloud-info-center> [Accessed Dec, 2010]
- 3            Virtual Trade Show: 10 Best Practices For Cloud Computing, By Jennifer Bosavage, CRN, May. 12, 2010 [online]  
URL: <http://www.crn.com/news/networking/224701700/virtual-trade-show-10-best-practices-for-cloud-computing.htm> [Accessed Dec, 2010]
- 4            Wikipedia on cloud computing [online]  
URL: [http://en.wikipedia.org/wiki/Cloud\\_computing](http://en.wikipedia.org/wiki/Cloud_computing) [Accessed Dec, 2010]
- 5            The Cloud Pyramid: Cloud Computing Glossary [online]  
URL: <http://pyramid.gogrid.com/glossary/> [Accessed Dec, 2010]
- 6            Cloud Computing Bootcamp by Marina Fisher, Nov 29, 2008 [online]  
URL: [http://blogs.sun.com/marinafisher/entry/cloud\\_computing](http://blogs.sun.com/marinafisher/entry/cloud_computing) [Accessed Dec, 2010]
- 7            Luis M. Vaquero Luis Rodero-Merino Juan Caceres Maik Lindner. A Break in the Cloud: Towards a Cloud Definition. January 2009
- 8            Distributed Coordination with ZooKeeper [online]  
URL: <http://www.igvita.com/2010/04/30/distributed-coordination-with-ZooKeeper/> [Accessed Sept, 2010]
- 9            Apache Hadoop ZooKeeper [online]  
URL:  
<http://hadoop.apache.org/ZooKeeper/docs/current/ZooKeeperOver.html>  
[Accessed Sept, 2010]
- 10          K Nearest Neighbor Algorithm Implementation and Overview By saharbiz, 4 Feb 2009 [online]  
URL: [http://www.codeproject.com/KB/recipes/K\\_nearest\\_neighbor.aspx](http://www.codeproject.com/KB/recipes/K_nearest_neighbor.aspx) [Accessed Jan, 2011]
- 11          Classification Algorithms on Text Documents Fall, 2001, YANCONG ZHOU and HYUK CHO [online]  
URL:  
<http://www.cs.utexas.edu/users/hyukcho/classificationAlgorithm.html>  
[Accessed Jan, 2011]
- 12          Research and Improvement of Personalized Recommendation Algorithm Based on Collaborative Filtering, Lijuan Zheng, Yaling Wang, Jiangang Qi,

Dan Liu, IJCSNS International Journal of Computer Science and Network Security, VOL.7 No.7, July 2007

- 13 RESTful Webservices with Java (Jersey / JAX-RS) – Tutorial by Lars Vogel, 30.12.2010 [online]  
URL: <http://www.vogella.de/articles/REST/article.html> [Accessed Feb, 2011]
- 14 Apache Hadoop HDFS Users Guid [online]  
URL: [http://hadoop.apache.org/hdfs/docs/current/hdfs\\_user\\_guide.html](http://hadoop.apache.org/hdfs/docs/current/hdfs_user_guide.html) [Accessed Sept, 2010]
- 15 Apache Hadoop HDFS Architecture Guide. [online]  
URL: [http://hadoop.apache.org/hdfs/docs/current/hdfs\\_design.html](http://hadoop.apache.org/hdfs/docs/current/hdfs_design.html) [Accessed Sept, 2010]
- 16 T. White, Hadoop: The Definitive Guide. O'Reilly Media, [online]  
URL: <http://oreilly.com/catalog/0636920010388/preview> [Accessed Sept, 2010]
- 17 Jason Venner, Pro Hadoop book
- 18 The Hadoop Distributed File System Shvachko, Konstantin; Kuang, Hairong; Radia, Sanjay; Chansler, Robert; Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on Digital Object Identifier: 10.1109/MSST.2010.5496972 Publication Year: 2010 IEEE Conferences [online]  
URL: <http://storageconference.org/2010/Papers/MSST/Shvachko.pdf> [Accessed Sept, 2010]
- 19 The Hadoop distributed filesystem: Balancing portability and performance by Shafer, J.; Rixner, S.; Cox, A.L.; Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on Digital Object Identifier: 10.1109/ISPASS.2010.5452045 Publication Year: 2010 IEEE Conferences
- 20 HDFS scalability: the limits to growth by Konstantin V. Shvachko [online]  
URL: <http://www.usenix.org/publications/login/2010-04/openpdfs/shvachko.pdf> [Accessed Sept, 2010]
- 21 Cloudera [online]  
URL: [http://www.vmware.com/appliances/directory/uploaded\\_files/What%20is%20Hadoop.pdf](http://www.vmware.com/appliances/directory/uploaded_files/What%20is%20Hadoop.pdf) [Accessed Sept, 2010]
- 22 Apache Hadoop. [online]  
URL: <http://wiki.apache.org/hadoop/HBase/HBaseArchitecture#intro> [Accessed Sept, 2010]
- 23 BigTable Model with Cassandra and HBase [online]

- URL: <http://horicky.blogspot.com/2010/10/bigtable-model-with-cassandra-and-HBase.html> [Accessed Sept, 2010]
- 24 Toad World, Could DB [online]  
URL: [http://wiki.toadforcloud.com/index.php/Cassandra\\_Column\\_Families](http://wiki.toadforcloud.com/index.php/Cassandra_Column_Families)  
[Accessed Sept, 2010]
- 25 Use HBase and Bigtable to create and mine a semantic Web by Gabriel Mateescu [online]  
URL: [http://www.ibm.com/developerworks/opensource/library/os-HBase/#data\\_model](http://www.ibm.com/developerworks/opensource/library/os-HBase/#data_model) [Accessed Sept, 2010]
- 26 Matching Impedance: When to use HBase By bryan | Published: March 11, 2008 [online]  
URL: <http://blog.rapleaf.com/dev/2008/03/11/matching-impedance-when-to-use-HBase/> [Accessed Sept, 2010]
- 27 HBase Available in CDH2 by Chad Metcalf, September 29, 2009 [online]  
URL: <http://www.cloudera.com/blog/2009/09/HBase-available-in-cdh2/>  
[Accessed Sept, 2010]
- 28 Hadoop/HBase vs RDBMS by Jonathan Gray [online]  
URL: <http://www.docstoc.com/docs/2996433/Hadoop-and-HBase-vs-RDBMS> [Accessed Sept, 2010]
- 29 Hadoop HBase-0.20.2 performance evaluation Carstoiu, D.; Cernian, A.; Olteanu, A.; New Trends in Information Science and Service Science (NISS), 2010 4th International Conference on Publication Year: 2010, Page(s): 84 – 87 IEEE Conferences
- 30 Competence Mining for Team Formation and Virtual Community Recommendation by Sérgio Rodrigues, Jonice Oliveira, Jano M. de Souza [online]  
URL:  
[http://www.sergiorodrigues.net/aulas/downloads/projetofinal/paper\\_SergioRodrigues\\_CSCWD.pdf](http://www.sergiorodrigues.net/aulas/downloads/projetofinal/paper_SergioRodrigues_CSCWD.pdf) [Accessed Jan, 2011]
- 31 Staffing a software project: A constraint satisfaction and optimization-based approach by Ahilton Barreto, Márcio de O. Barros, Cláudia M.L.Werner [online]  
URL:  
[http://homepages.dcc.ufmg.br/~colares/arquivos/@papers/@sbse/@aloc\\_recurso/coppe\\_ufrj2007.pdf](http://homepages.dcc.ufmg.br/~colares/arquivos/@papers/@sbse/@aloc_recurso/coppe_ufrj2007.pdf) [Accessed Jan, 2011]
- 32 Decision support for team staffing: An automated relational recommendation approach by Jochen Malinowski, Tim Weitzel, Tobias Keim [online]  
URL:  
[http://www.sciencedirect.com/science?\\_ob=ArticleURL&\\_udi=B6V8S-4NR185V-6&\\_user=10&\\_coverDate=06%2F30%2F2008&\\_rdoc=1&\\_fmt=high&\\_ori](http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6V8S-4NR185V-6&_user=10&_coverDate=06%2F30%2F2008&_rdoc=1&_fmt=high&_ori)

g=search&\_origin=search&\_sort=d&\_docanchor=&view=c&\_searchStrId=1612675937&\_rerunOrigin=google&\_acct=C000050221&\_version=1&\_urlVersion=0&\_userid=10&md5=c6e2c8edf7463273ffb5556ac02fe2c3&searchtype=a [Accessed Jan, 2011]

- 33      Implementing RESTful Web Services in Java by Jakub Podlesak and Paul Sandoz [online]  
URL: [http://blogs.sun.com/enterprisetechtips/entry/implementing\\_restful\\_web\\_services\\_in](http://blogs.sun.com/enterprisetechtips/entry/implementing_restful_web_services_in) [Accessed Jan, 2011]
- 34      HBase [online]  
URL: <http://en.wikipedia.org/wiki/HBase> [Accessed Sept, 2010]
- 35      Apache HBase [online]  
URL: <http://hbase.apache.org/> [Accessed Sept, 2010]