Hogeschool Utrecht

Vrije Universiteit Amsterdam

Implementing A Large-Scale Distributed Database on XtreemOS

Author:
Niek Linnenbank

Supervisor:
Guillaume Pierre

Examiner:
Leo van Moergestel

May 28, 2009

1500967

Abstract

Currently the XtreemOS grid does not have a scalable distributed database suitable for cloud computing. This thesis presents the implementation of Apache HBase on XtreemOS. Apache HBase is an open source scalable distributed database modeled after Google BigTable. In XtreemOS the XtreemFS distributed filesystem can be used to provide shared storage to all HBase nodes. Using the XtreemOS DIXI framework, it is possible to submit jobs to be executed on available nodes in the XtreemOS grid. By careful submission of DIXI jobs, it is possible to add and remove HBase region servers from the XtreemOS grid. Experiments show that HBase is scalable on XtreemOS.

Acknowledgements

This project would never have been possible without the help of many people I have met at the VU and in the last several years. First I would like to thank Guillaume for giving me the great opportunity to participate in and learn from the XtreemOS project, the System Programming class and overall his excellent supervising during the project. Next I thank Corina for her work on the VU testbed, which provided me and other students a platform for experiments, development and performance benchmarks. Ofcourse great many thanks to my parents, for supporting me both financially and socially during my studies. Next thanks go to Leo van Moergestel as without his permission letter I would not have been able to register as a pre-master student at the VU, nor could I have followed pre-master courses last September nor taken place in this project. Another thanks to the XtreemOS DIXI developers who helped me during this project to solve problems and bugs.

Special thanks to Pieter Lange and the *Hogeschool Utrecht Linux Kennisgroep* (HULK). Pieter introduced me to the wonderful world of Linux and learned me to use it. During the years of study on the HU, we and several other Linux and open source entheusiasts formed the HULK, which currently provides hosting services HU students and develops publicly available open source projects. Majid Hossainy, system administrator at the Hogeschool Utrecht, made the existence the HULK possible by helping us with organization, supplying hardware and finding available room for us and the servers. The HULK has been a great place to practice computing just for the fun of it and I very much look forward to go to *Hacking At Random 2009* with the HULK.

Contents

List of Figures

Glossary

API	Application Programming Interface, 12, 18, 19, 41, 42, 44
bug	Programming error in a computer program, 8, 25
bugtracker	System for displaying and organizing currently known bugs and previous bugs in computer software, 8
client	Computer program which initiates a request to a (remote) service, 10, 15–17, 19, 28, 32, 33, 35–37, 42
cloud computing	Customers rent and use the cloud provider's infrastructure without knowning the location, operating system and hardware of the machines they use. Cloud providers often adopt the utility computing model for billing customers, 8, 18
cluster computing	Computing model in which a group computers connected by local network are used to perform collaborative computation. Nodes in a cluster have the same hardware and software, and belong to one administrative domain, 12
consistency	Correctness of data presented by a computing system. In order to be consistent, the data must not contradict or conflict itself, 10
database	Computer program for efficient inserting, modifying, deleting and querying of structured data, 8, 10, 16, 18, 19, 21, 25, 40
filesystem	Operating system component responsible for organizing data on storage devices, 8, 12, 14, 15, 17, 27, 28, 40–43

grid computing	Computing model where computing resources of multiple administrative domains are bundled by a network for collaborative computation. Unlike clusters, nodes in a grid may have different hardware, operating systems, network connectivity and security policies, 8, 12, 28, 29, 40, 42
HTTP	HyperText Transfer Protocol, 14, 19
HU	Hogeschool Utrecht, 21
I/O	Input and Output, 8
kernel	Core operating system component responsible for process management, memory management, inter process communcation and optionally more functionality, 15, 28
node	A single computer connected to a network, 8, 10, 12, 27, 28, 32, 38, 40–42, 44
open source	Software licensing model where distribution, modification and copying of source code is permitted, 8
POSIX	Portable Operating System Interface for UniX, 15, 19, 21, 28
source code	Human readable program text, which may be interpreted or compiled into machine language for execution by the computer, 8, 28, 29, 40, 41, 43
throughput	The average rate of data processed, often expressed in bits or bytes per second, 8, 38
URL	Uniform Resource Locator, 27, 28
utility computing	Computing paradigm in which computing resources are seen as public utility services, 8
VU	Vrije Universiteit, 21, 22, 35, 38
XML	Extensible Markup Language, 8, 19, 28, 29

Chapter 1

Introduction

As the number of users in modern computing grows into the millions, several new paradigms have evolved in order to deal with their demands, such as utility computing. In utility computing customers use the computing resources from a service provider and pay only for the amount of resources they use, instead of the full purchase price of the machines they use. The word utility in utility computing is used to make an analogy to public services, such as electricity, gas and water. Utility computing services can be provided in the form of a grid computing or cloud computing infrastructure. Grids consist of typically large numbers of compute nodes connected by a network, specialized for running computational intensive jobs. Nodes in a grid may exist in different administrative domains, can have different hardware, operating systems, networks and security policies. Cloud computing infrastructures are often provided by one administrative domain. In cloud computing customers can develop, deploy and run applications on a cloud provider's infrastructure [2, 3], without knowing details like the location, operating system and hardware of the machines they use. The cloud can be seen as an infinite pool of computational resources from which customers may take or give back based on their needs. Beneficial for customers is that they do not need to purchase expensive hardware, thereby also avoiding installation and maintenance costs. Many cloud providers have adopted the utility computing model for billing customers and may depend on grids to make their resources available for customers.

A grid operating system in current development is XtreemOS [4]. It is an ongoing project funded by the European Commision (2006-2010). The XtreemOS project aims to build a Linux-based [5] grid operating system, to support Virtual Organizations (VO) [6]. VO's are used to define a set of users and resources provided by real organizations. During the lifespan of a VO, users can run their applications, given they are properly authenticated. The XtreemFS [7] distributed filesystem included in XtreemOS enables users to access their files from any resource node in the grid. XtreemFS is capable of replication, parallel I/O and distribution of large amounts of data spread among the available resource nodes. Because XtreemFS supports such a huge storage capacity, users may want to use it for more purposes such as running a highly scalable database suitable for cloud computing on top of XtreemFS. Currently XtreemOS does not have such scalable database.

One could be tempted to think of a traditional relational database as a solution for this problem. This will work for a few nodes, however it does not scale to millions [8]. The reason a relational database cannot scale to such numbers is that they are replicated: each node needs a complete copy to work. Therefore the maximum throughput is dictated by the degree of replication [9]. It is sometimes said that relational databases scale easy vertically but are hard to scale horizontally. This means a relational database running on a single node can achieve greater performance to a certain boundary by upgrading the machine (vertical), which may be financially expensive. By adding more machines relational databases may grow further in terms of performance (horizontal), but only until replication becomes a bottleneck.

Fortunately there exists a class of highly scalable databases, pioneered by Google BigTable [10] and Amazon SimpleDB [11]. These systems have a different design than relational databases which allows them to be highly scalable. They present client applications a sparse, distributed, persistent, multidimensional, sorted map. The map consists of rows and columns, allowing applications to access it using key to value translation. High scalability is achieved because ranges of rows in the map are distributed among a great number of cheap commodity servers, thereby also keeping financial costs low [12]. Unfortunately both BigTable and SimpleDB are commercial products, which prevents XtreemOS as an open source project to redistribute it. There are however open source alternatives to BigTable which can be included in XtreemOS, like Apache HBase [13]. HBase describes itself as an open-source, distributed, column-oriented store modeled after the Google BigTable paper [1].

HBase was initially not designed to run on a grid like XtreemOS. For example, the HBase configuration is a static XML [14] file, which needs to be regenerated containing the address of a master node somewhere in the XtreemOS grid. Additionally, XtreemOS is still a relatively new and experimental system. Unfortunately this means that it contains many (un)known problems and programming errors [15], and many are yet to be discovered. Because XtreemOS is in heavy development, user and developer documentations are often out of date, incomplete or non existent. This eventually requires one to read the XtreemOS source code, which in turn is undocumented in many places, and hard to read. Therefore it is currently a difficult task to implement any new application on XtreemOS.

During this project I have spend a significant amount of time to learn the various components involved in XtreemOS, XtreemFS and HBase. Due to the complex nature of distributed systems and the many different services in XtreemOS it can take a big effort for new developers in this area to understand their concepts. By using the XtreemOS testbed at the Vrije Universiteit [16], experimenting with HBase and eventually reading parts of their sourcecode I was able to learn and meanwhile prepare myself to get started with the actual implementation. I have written a python program called hbase-xos which is able to submit jobs to XtreemOS to run HBase instances on nodes in the grid, using XtreemFS for data storage and configuration. Documentation is available in the hbase-xos manual page and comments inside the sourcecode. I have packaged HBase and the hbase-xos script in RPM format. They will be included in the second public release of XtreemOS. While developing hbase-xos I have encountered several (un)known problems in XtreemOS and reported those which where directly reproducible to the bugtracker or their authors. For this reason I installed a test installation of several XtreemOS nodes in VMware, which I used instead of the Vrije Universiteit testbed to workaround bugs, simulate crashes and try out new versions of XtreemOS components from the Subversion repository. At the Vrije Universiteit testbed I performed benchmarks of HBase which indicate HBase is scalable on XtreemOS using XtreemFS as the distributed shared filesystem.

The rest of this thesis is organized as follows. Section 2 describes related research and work done in this area. Section 3 lists the project as it was originally planned when it started. In section 4 the current implementation of HBase on XtreemOS is presented. Section 5 evaluates the experiments, implementation and project as a whole and section 6 concludes.

Chapter 2

Related Work

Over the years scalable databases has been an active topic in computer science research. Traditional relational databases attempt to provide scalability by replicating the contents of the database on multiple machines, such that more client requests can be processed. A design choice made when using replication is whether each machine stores a fully replicated copy of the database, or a small part [9]. When using full replication read requests from client applications will be fast, as they are spread among the available machines. Write requests however, are much more difficult to scale with a fully replicated database, as for each write performed by a client, the replicated copy on all machines need to be updated. When only small parts of the relational database is replicated and distributed among the machines, it is possible that due to network or hardware failures, the replicated parts of the database become inconsistent.

Recent work towards providing scalability to databases includes the *Ganymed* [17] project. Ganymed achieves both scalability and consistency by using a special transaction scheduling algorithm called *Replicated Snapshot Isolation with Primary Copy* (RSI-PC). With RSI-PC, Ganymed proposes to separate read-only and update transactions to the database, and send them to different replicas. Read transactions are distributed among any of the available replicas. Write transactions are send to a master replica only, instead of all active machines. The RSI-PC scheduling algorithm ensures that temporary inconsistencies between the replicas are hidden from client applications to achieve consistency, and synchronizes the master replica with all other replicas appropriately. Examples of relational databases that use such scheduling algorithm include Oracle and PostgreSQL. Oracle 11g currently supports up to 100 nodes [18]. To implement a scalable database on XtreemOS we need a different design, as real grids may consist of thousands of nodes or more.

Chapter 3

Background

Contents

In this chapter we describe various systems which form the context of our project. First we introduce XtreemOS and several of it's components. Then we describe the XtreemFS distributed filesystem servers and client, and finally we discuss scalable distributed databases.

3.1 XtreemOS

In order to perform computation of increasingly large technical or scientific problems, (super)computers must be able to process and compute more data, and faster. One way for companies to tackle this problem is to simply buy a bigger supercomputer, which can financially be very expensive. Another solution involves combining a group of smaller, less expensive computers (nodes), to perform collaborative computation. Nodes may sometimes be homogeneous: they have the same hardware, operating system, and are on the same (local) network. This is called a computer cluster [19]. In contrast to clusters, grids [20] can consist of nodes with different hardware, operating systems, and may be geographically distributed, connected via a wide area network such as the internet.

XtreemOS is an open source project funded by the European Commission, whichs aims to develop a grid operating system based on Mandriva Linux. It enables users to create Virtual Organizations [21] using a modified Linux kernel. This means that users can share, select and aggregate a wide variaty of geographically distributed resources, such as clusters, storage systems, supercomputers, mobile devices and more. Once formed, a Virtual Organization can be used to perform high demanding computations. For applications, XtreemOS is entirely transparent. Programs can run on a XtreemOS Grid using POSIX [22] or SAGA [23] interfaces, without any modifications.

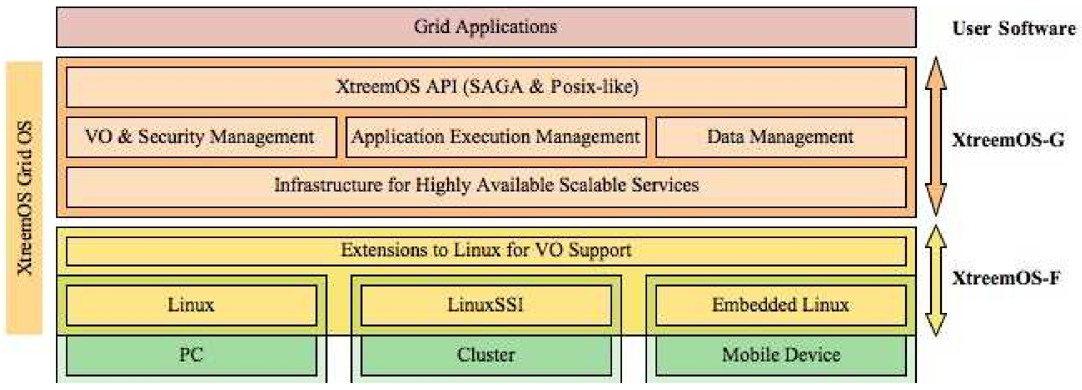


Figure 3.1: Overview of the XtreemOS architecture.

The architecture of XtreemOS is logically divided in two layers: the G-layer and the F-layer. The lowest F-layer contains Linux extensions, to provide virtual organization support within the operating system itself. It contains software support for mapping of virtual organization identities to local user identities, a checkpointing kernel module [24] to save and restart submitted jobs and LinuxSSI cluster integration. The G-layer has all XtreemOS services to enable connectivity between nodes, execution management and a shared distributed filesystem.

3.1.1 Application Execution Management

One of the most important functionalities in XtreemOS is the ability to run programs on the available resource nodes in the grid. Users can do this in XtreemOS by submitting a job which describes what program to run, optional arguments and environment variables to pass to the program and how much resources it requires such as processor power and memory. Application Execution Management is the software package in XtreemOS which supports submitting, scheduling and running jobs. Jobs submitted to the Application Execution Management service must be written in the Job Submission Description Language [25].

3.1.2 Infrastructure for Highly-available and Scalable Services

In order to start executing a program on a resource node which fits the job description best, AEM needs a service to select candidate resource nodes. In XtreemOS there are two services which together provide this functionality. The first is the Resource Selection Service and is responsible to select resources based on static attributes, like the operating system, architecture, processor power and total memory. After the RSS has selected resources, another service called the Application Discovery Service further selects nodes using dynamic attributes such as the free memory available, network traffic and current processor load. RSS and ADS together form the Scalable Resource Discovery System.

3.1.3 Virtual Organization Management

Users and resource nodes can become member of a Virtual Organization in XtreemOS using the VOLifeCycle web application or command-line utilities. Creating, modifying, removing, joining and leaving Virtual Organizations is supported by VOLifeCycle for registered users.

3.1.4 Security Management

To prevent unauthorized individuals to run programs on the grid or access XtreemFS files, XtreemOS uses certificates to authenticate and authorize users. For each Virtual Organization users use a X509 certificate [26] when using services on resource nodes provided by XtreemOS. The Credential Distribution Authority is responsible for the creation new security certificates for users, and the Resource Certification Authority generates certificates to authenticate resource nodes.

3.1.5 Data Management

Most frequently a submitted job will output results after it has completed calculations. This requires a shared filesystem on all XtreemOS resource nodes. In XtreemOS jobs can read and write data to the XtreemFS distributed filesystem which is available on all resource nodes in the XtreemOS grid.

3.1.6 XtreemOS API

Applications submitted to the XtreemOS Application Execution Management service are able to run unmodified using the standard POSIX system interface. Grid applications in XtreemOS can either submit jobs directly to the Application Execution Service or use the SAGA [23] API. SAGA standardizes the interface to the grid, which allows SAGA applications to run under different grids transparently.

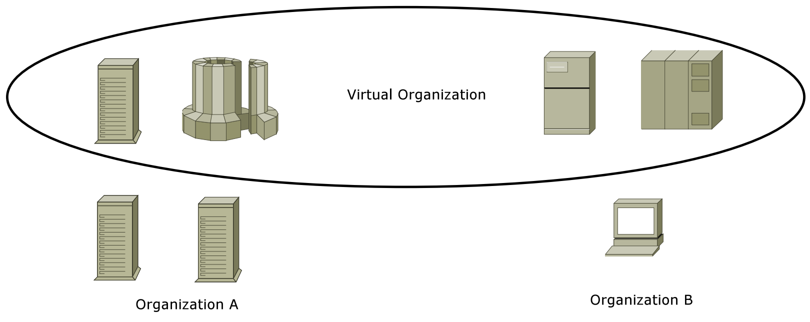


Figure 3.2: Virtual Organizations allows sharing resources between organizations.

3.2 XtreemFS

XtreemFS is a distributed filesystem, which means it is capable of storing files on several computers connected by a network. Files can be striped into smaller pieces and distributed on multiple servers. This enables XtreemFS clients to access the parts in parallel for optimal performance. In XtreemFS users can create different volumes for storing data, for example a volume to mount their home directory. XtreemFS has been especially designed for wide area networks such as the Internet, allowing users to use XtreemFS volumes from any location.

The XtreemFS design modelled after the object-based filesystem architecture [27]. The term object comes from the fact that it splits files into smaller fixed-size parts called objects. In contrast to block-based file systems, the size of such an object can vary from file to file. Objects are stored on storage servers while the metadata of files such as the filename, size and modification time is stored separately on metadata servers. The XtreemFS services are implemented in the Java programming language communicate internally using the JavaScript Object Notation [28] over HTTP.

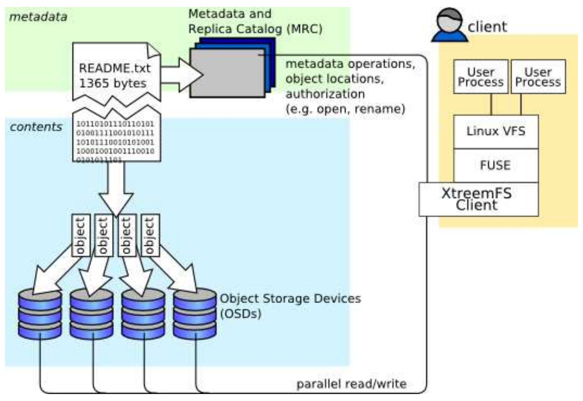


Figure 3.3: Overview of the XtreemFS architecture.

3.2.1 Metadata and Replication Catalog

Metadata of files in XtreemFS is stored on the Metadata and Replication Catalog (MRC). File information such as the name, size and owner are kept in the MRC. There can be multiple instances of the MRC running on different servers across the network. The MRC is also responsible for proper authentication and authorization of users for access to files in XtreemFS, which can be done using client provided UNIX identities or in the form of security certificates.

3.2.2 Object Storage Device

XtreemFS uses a technique called striping to split file content into small objects. Striped objects are distributed among Object Storage Device (OSD) instances available on the network. This allows XtreemFS clients to read and write objects in parallel to increase the overal bandwidth. XtreemFS supports different striping policies which can be specified on a per-file basis.

3.2.3 Directory Service

In XtreemFS the Directory Service is used as a central registry for all services in XtreemFS. MRC instances use the Directory Service to discover OSD servers and users point their XtreemFS client to a Directory Service when to want to mount an XtreemFS volume.

3.2.4 Client

The XtreemFS distribution comes with several client applications for accessing the XtreemFS services, which require a recent Linux 2.6.x kernel, OpenSSL 0.9.8 and *Filesystem In UserSpace* [29] (FUSE) support. With FUSE programmers can develop their own filesystems which will run as a regular user process interacting with the Linux kernel. Standard POSIX functions are translated into FUSE messages by the Linux kernel and send to the filesystem process. Figure 3.4 illustrates how an example "Program A" accesses XtreemFS with the FUSE filesystem loaded in Linux. In the current XtreemFS 1.0 release the following client applications are supported:

- **xtfs_lsvol** Lists available XtreemFS volumes given the address of an active MRC service. Users may use this command to discover what volumes are available.
- **xtfs_mkvol** Creates a new XtreemFS volume, also taking the location of a MRC service as input. It is possible to specify the default striping policies and access controls for the volume.
- **xtfs_rmvol** Removes a XtreemFS volume including all files from the given MRC service. Like all other client applications, users may input a security certificate to authenticate themselves.
- **xtfs_mount** Mounts a XtreemFS volume on the local filesystem. By mounting an XtreemFS volume in a directory on the local filesystem, users can read, write and delete their files on the XtreemFS volume the same way as they would manage their files on the local filesystem.
- **xtfs_umount** Unmounts a XtreemFS volume on the local filesystem. After mounting a XtreemFS volume in a directory on the local filesystem, it can be unmounted to close the connection with the XtreemFS volume.
- **xtfs_stat** Displays information about a file on XtreemFS. Is shows standard POSIX file information like the filename, size, modification timestamp and ownership, and also XtreemFS specific attributes such as striping information and the location of each object inside the file.
- **xtfs_cleanup** Performs consistency checks on an OSD service. This program verifies that each object stored on an OSD has a corresponding entry on a MRC service, and may be removed otherwise.
- **xtfs_send** Invokes remote procedure calls on an XtreemFS service. Users can use this program to debug problems in running XtreemFS services by inspecting the output returned by this program.
- **xtfs_mrcdbtool** Dumps and restores XtreemFS MRC databases. Users can backup and restore the database of running XtreemFS MRC instances in XML format.

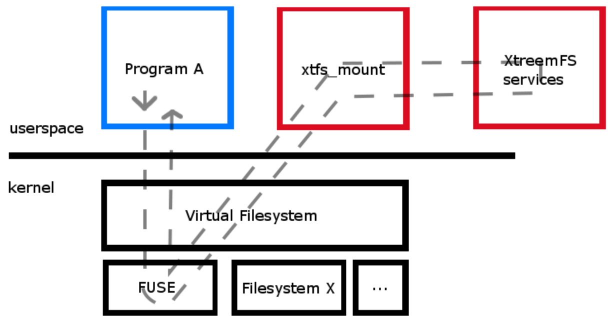


Figure 3.4: The XtreemFS client uses FUSE to mount volumes on the local filesystem.

3.3 Scalable distributed databases

3.3.1 Google BigTable

Millions of users across the globe use the services offered by Google each day, such as Google Search, Google Earth and Google Finance. In order to provide the huge storage capacity and performance demands required by these applications, Google has designed and implemented BigTable [10]. Google describes BigTable as a distributed storage system for managing structured data, which presents applications a sparse, distributed, persistent, multi-dimensional, sorted map. A map is an abstract datatype containing a collection of keys. Each key in a map is associated with one value. In BigTable the map is indexed by a row key, column key, and a timestamp, allowing applications to access BigTable using simple key-value translation. Each value in the BigTable map is an uninterpreted array of bytes.

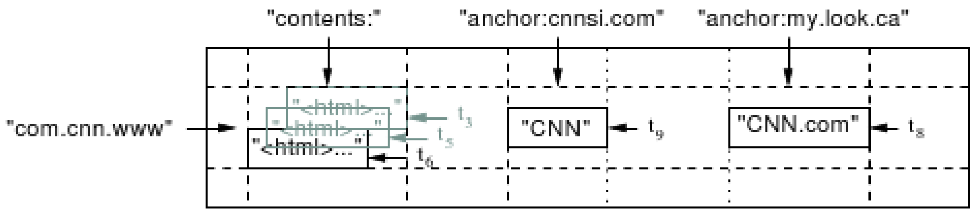


Figure 3.5: An example BigTable which stores Web pages (adapted from [1])

The BigTable map consists of typically large number of rows and columns which may have large empty spaces, therefore BigTable is called a sparse map. Rows inside the BigTable map are divided into distinct ranges of rows called tablets. Each tablet may be stored on multiple tablet servers on the network, hence BigTable is a distributed map. This property enables BigTable to scale to thousands of servers, as each tablet server is responsible for a small range of rows in the map. BigTable persists written data by user applications, meaning the written information will remain stored in BigTable even after the user program terminates. Whenever a user application writes a new value for a given row and column key, BigTable keeps the old value to maintain the history of the record. Applications may request the latest or an older value by specifying a different timestamp when accessing the value, which is the reason why BigTable is called a multi-dimensional map. Another property of BigTable is that it is sorted lexicographically by row keys. Client applications can exploit this to keep the number of tablet server needed for communication low. An example Google uses in their paper to illustrate this is a BigTable for storing webpages. The hostname components of each URL from a webpage fetched by the webcrawler are reversed and the result is used as the row key, for example *com.starwars.www/index.html*. Because BigTable is a sorted map, webpages from the same domain are grouped together.

In BigTable column keys are grouped into sets called column families. Data inside a column family is ideally of the same data type for efficient data compression. Column families must be created in BigTable before any data can be stored under a column key in that family. An example from the Google BigTable paper is a column family *anchor*, to contain all anchors linking to external webpages. The column key in this example is the URL to the referring website and the value is the link text. BigTable uses column families also as the basic unit of access control. Some applications may be allowed to write to a certain set of column families, while others can only read or none.

Before a client application can access BigTable, it first needs to know the location of the tablet servers needed to complete the request. To maintain scalability, BigTable has a three-level tablet hierarchy which are used by clients to find the correct tablet servers they need. The first level is the *root tablet* and contains the location of all tablets of the *METADATA* table. The *METADATA* table in turn describes the location of all known tablets defined by client applications using their row key.

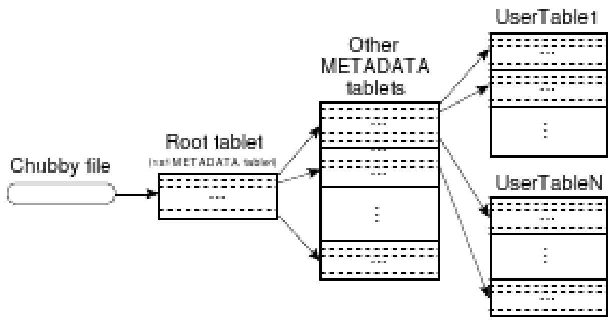


Figure 3.6: Tablet location hierarchy used in BigTable (adapted from [1])

Google uses the *Google File System* [30] (GFS) to provide distributed storage to BigTable for storing data and log files. The GFS is designed for clusters of hundreds to thousands of cheap machines connected by a network. Just like XtreemFS, GFS is an object-based filesystem, meaning that it splits files in fixed-size parts. In Google terminology these parts are called chunks. With GFS, Google assumes that system component failure may happen at any time, at any machine. Therefore chunks are stored on multiple machines to ensure data integrity in the case a machine becomes unavailable. As with XtreemFS, chunks may be accessed in parallel by clients to achieve greater performance.

Since april 2008 Google offers customers access to BigTable via the Google App Engine. Google App Engine allows customers to run their web applications on Google's infrastructure, which can be written in Java or Python. Google App Engine is free of costs for applications up to 500MB of storage, with CPU power supporting about 5 million page views a month. Only if a customer enables billing in Google App Engine, the limits are released and they will pay only for the resources they use. The runtime environments offered by Google App Engine ensures that a web applications runs in a limited sandboxed environment to prevent disturbing or affecting other programs running on the system. Programs running on Google App Engine may use BigTable to read and write data efficiently.

3.3.2 Amazon SimpleDB

Amazon has played a critical role in the development of cloud computing by providing cloud computing services to customers on the internet. The *Amazon Elastic Computing Cloud* (EC2) enables user run applications on the operating system of their choice on the cloud computing infrastructure of Amazon. EC2 can do this because it offers a true 32 or 64-bit *virtual environment* to users. By creating and uploading an Amazon Machine Image to EC2, users can use their own libraries, applications, data, configuration and operating systems on virtual instances running on EC2. In EC2 users are able to start any number of virtual instances by using an online web interface.

EC2 offers complete control to virtual instances for users. They can be booted remotely using the webservice API. While booting users can access the console output of the virtual instances just like a physical machine. There are several virtual instance types available for users to choose from, including standard instance types (small, large and extra large) and High-CPU instances (High-CPU medium, High-CPU extra large). Smaller instance types have less virtual cores, system memory, and storage capacity than larger instances. For example, a standard small instance has 1.7 GB memory, 1 EC2 compute unit and 160 GB storage on a 32-bit platform, while the extra large standard instance has 15 GB memory, 8 EC2 compute units and 1690 GB storage on a 64-bit platform. Virtual instances can be associated with an *Elastic IP address* in EC2. An elastic IP address is a static endpoint which can be assigned to and released from virtual instances at any time. EC2 supports placing virtual instances on multiple geographically distributed locations called *Availability Zones* in Amazon terms. Customers can benefit by choosing nearby Availability Zones for low latency connections and launching more virtual instances abroad to protect applications from failure at a particular location.

The services and resource offered by EC2 are rented by customers on a pay-per-use basis. Customers pay Amazon only for the amount of resources they have actually used. The eventual price paid to Amazon is based on the number of virtual instances used, the type of virtual instances and how many hours they were used. Amazon has defined a price list containing to be paid per hour, per virtual instance type. Optionally customers may choose to reserve instances at the cost of a one-time payment. Customers can define the duration terms of their reservation and are then billed with a significant discount on the per-hour usage rates during that period.

Besides EC2 Amazon offers a scalable storage solution called *Amazon Scalable Storage Service* (S3) for web applications to their customers. With S3 users can read, write and delete objects ranging from 1 byte to 5 gigabyte, using REST or SOAP programming interfaces. Like EC2 customers only pay for the storage they use from Amazon. Prices are based on the total used storage capacity in gigabytes, the number of gigabytes transferred over the internet and the number of S3 API requests used.

On the same infrastructure on which Amazon hosts EC2 and S3, Amazon provides *Amazon SimpleDB*. SimpleDB is a highly scalable distributed database similar to Google BigTable and works closely together with EC2 and S3 in the Amazon cloud computing infrastructure. Collectively EC2, S3 and SimpleDB provide a platform for data storage, processing, and querying to customers on a pay-per-use basis. SimpleDB is accessible for users with Java, C#, Perl, PHP and VB.NET programming interfaces using simple functions for reading, writing and querying data stored in SimpleDB. Amazon charges customers based on the machine hours used to process requests, the total data in gigabytes transferred over the internet, and the total storage capacity used in gigabytes.

3.3.3 Apache HBase

The *Apache Software Foundation* (ASF) is a non-profit organization in the United States of America. It was founded in 1999 by a group which called themselves the “Apache Group”. The group had chosen the name “Apache” in respect to the Native America Indian tribe of Apache. The Apaches live in Northern America since around the 10th century and proved to be fierce warriors during battles fought against the Spaniards and Mexicans. Between 1995 and 1999 the Apache Group wrote the Apache HTTP server, which became and currently is leader of the market. As a computer joke the name Apache is sometimes interpreted as “a patchy server”, meaning the Apache HTTP server was made from a series of patches, but this is not the origin of the name.

Figure 3.7: The purple and red colored feather is the logo the Apache Software Foundation

The Apache Software Foundation develops Hadoop [31], an open source reliable, scalable, distributed computing environment written in Java. With Hadoop users can run applications to process large amounts of data, on clusters of typically thousands of commodity servers. Applications use the Hadoop MapReduce [32] implementation when running on an Hadoop compute cluster. MapReduce divides the work to be done by applications into small parts, and executes those on Hadoop. The Hadoop project is consists of several subprojects, including Hadoop Core for providing MapReduce functionality and the *Hadoop Distributed FileSystem* (HDFS), the Zookeeper coordination system, and Apache HBase as the scalable distributed database for Hadoop.

Just like BigTable provides users a distributed scalable database on top of the Google filesystem, HBase supports BigTable capabilities on an Hadoop cluster using the HDFS distributed filesystem, as described on the HBase project webpage [13]. HBase initially started as a contribution [33] to the Hadoop project, by Michael Stack at a company called Powerset. It was first publicly released in October 2007, included in Hadoop 0.15.0, and since that time it has outgrown into a separate subproject of Hadoop. Like Hadoop, HBase is written in Java and runs on most POSIX compatible operating systems. The HBase architecture is based on the BigTable paper [1] published by Google, and therefore also presents client applications with a map. In HBase terminology, tables are called regions, and tablet servers are called region servers, but it’s concepts remain the same. Although HBase already has several public releases available on the internet, it is currently still an experimental system compared to BigTable, and is under active development by the HBase community. For instance, HBase does not yet have authentication or access control mechanisms implemented.

There are several client programming interfaces available for developing applications which can interact with HBase. HBase supports the default Java programming interface, and also *Thrift* [34] and *REST* API’s. Thrift is a framework for transparent communications between programs written in different programming languages. The HBase distribution comes with Facebook’s Thrift implementation, which allows HBase to communicate with client programs written in C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, Smalltalk and OCaml. REST offers a method to transfer information between applications over the HTTP protocol. The HBase REST interface can be accessed with HTTP requests, in combination with XML tags or JSON.

Chapter 4

Project Plan

Contents

The following sections describe the initial plan of our project. It gives a description of the project as we originally defined and planned it at the very beginning of our project. We present the project goal, deliverables, organization, activities, planning, risc analysis, quality assurance and cost and gain analysis.

4.1 Project goal

The goal of this project is to implement a large scale distributed database for XtreemOS. We will use an existing open source implementation, such as Apache HBase, and modify it to cooperate with XtreemOS, using XtreemFS for storage. Optionally, if enough time remaining, investigate if it is possible to modify the implementation such that it efficiently uses XtreemFS' internal organization, apart from the normal POSIX interface, to further optimize performance.

4.2 Deliverables

Deliverable	Priority
Apache HBase implementation on XtreemOS (using HDFS)	MUST HAVE
Apache HBase implementation on XtreemOS (using XtreemFS)	SHOULD HAVE
Improved HBase performance (using XtreemFS' capabilities)	COULD HAVE
Packaging scripts, for easy installation on XtreemOS	SHOULD HAVE
Detailed technical documentation	SHOULD HAVE
Presentation and demonstration of the system	MUST HAVE
Project thesis	MUST HAVE

4.3 Organization

4.3.1 Location

The project will take place on the *Vrije Universiteit Amsterdam*. The *Vrije Universiteit Amsterdam* was founded on 20 October 1880 by Abraham Kuyper in Amsterdam as a private orthodox protestant university, and initially consisted of three faculties: Theology, Literature and Law [35]. Abraham Kuyper was born in 1837, the son of a preacher. Kuyper was president of the Netherlands during 1901 until 1905, professor theology at the VU and also the first *rectorus magnifici*, which is the university president. Since it's foundation the VU was financed by donations from volunteers, which kept the VU growing from a small university to mid-size during the sixties and eventually large university in the seventies.

The word *Vrije* means freedom in Dutch and emphasizes the independence of the university from the government and church. Therefore the *Vereniging VU-Windesheim* manages the organization of the VU and also for the *Windesheim University of applied sciences.* Although freedom is important for the VU, the education itself at the VU is not free of costs. Dutch students pay a government determined tuition of about 1600 euro's a year, which is the same as other accredited universities in the Netherlands.

The logo of the VU is a griffin, which also symbolizes the values of the VU. It has the body of a lion and the wings of an eagle, and originated from Ancient Greece, Rome and is also the history of Christianity. On the webpage of the VU [36], the griffin is described as:

> *"The griffin represents the values embodied by VU University Amsterdam. The spreading wings represent the quest for knowledge, in complete freedom. The possession of knowledge brings with it responsibilities and these have to be addressed conscientiously. The griffin's feet, planted firmly on the ground, represent VU's commitment to the well-being of society as a whole. And like the griffin, VU University Amsterdam cannot be summed up in a few words."*

Figure 4.1: The logo of the VU is a blue griffin

The current campus of the VU is located at the Boelelaan 1105 in south Amsterdam. There are about 20,000 students, 2,200 faculty members and researchers among which 300 professors. Since it's foundation, the number of faculties on the VU has increased from three to twelve, including Science, Arts, Dentistry, Earth and Life Sciences, Economics and Business Administration, Human Movement Sciences, Philosophy, Phychology and Education, Social Sciences and the original Law, Literature and Theology faculties. Also see appendix A for a complete organogram of the VU.

In this project all activities will be taken place at the faculty of Sciences at the VU Amsterdam, including research, systems development, deployment and documentation. The VU also supplies the required hardware for performing experiments and development.

4.3.2 Members

The project consists of several members which are listed below. As XtreemOS is an international project, it includes people from the *Vrije Universiteit* but also from abroad. The list below only contains those people from XtreemOS who have played a role in the project.

Niek Linnenbank <nieklinnenbank@gmail.com>
Student

Responsible for performing research, building implementations, documenting and presenting all results during the project. Available from 09:00 - 17:00, Monday until Friday, either by e-mail or at the *Vrije Universiteit Amsterdam.*

Guillaume Pierre <gpierre@cs.vu.nl>
Project Supervisor

Guides and supervises Niek, makes decisions regarding all implementations, and is available for answering questions by email during the project. Every one or two weeks Niek and Guillaume have an appointment for project status.

Leo van Moergestel <leo.vanmoergestel@hu.nl>
Examiner

As the representative of the *Hogeschool Utrecht*, Leo will visit the *Vrije Universiteit Amsterdam* at least 2 times to examine the status of the project. Leo is also end-responsible for the final grade of the project.

Corina Stratan <cstratan@cs.vu.nl>
XtreemOS developer

Corina works at the *Vrije Universiteit Amsterdam* on the Resource Selection Service for the XtreemOS project, and is system administrator of the XtreemOS testbed at the *Vrije Universiteit Amsterdam.* She provides supports for users of the testbed and can review Niek's work.

Toni Cortes <toni.cortes@bsc.es>
Ramon Nou <rnou@ac.upc.edu>
Jacobo Giralt <jacobo.giralt@bsc.es>
XtreemOS developers

Toni, Ramon and Jacobo are developing the Application Execution Management service in XtreemOS. They are available for questions and reporting problems concerning AEM in this project.

Zhou Wei <zhouw@few.vu.nl>
HBase specialist

As an HBase specialist, Zhou is available by email during the project, for any technical questions about Apache HBase.

4.4 Activities

- Gather all required documents and software packages.
- Understand XtreemFS, XtreemOS, and Apache HBase internals.
- Run Apache HBase on top of XtreemOS (using Apache's HDFS).
- Investigate how to replace HDFS with XtreemFS on Apache HBase.
- Implement Apache HBase on XtreemOS, using XtreemFS as storage.
- Optimize performance by exploiting XtreemFS' internal organization.
- Debugging, testing and packaging.
- Technical documentation.
- Write the project thesis.

4.5 Planning

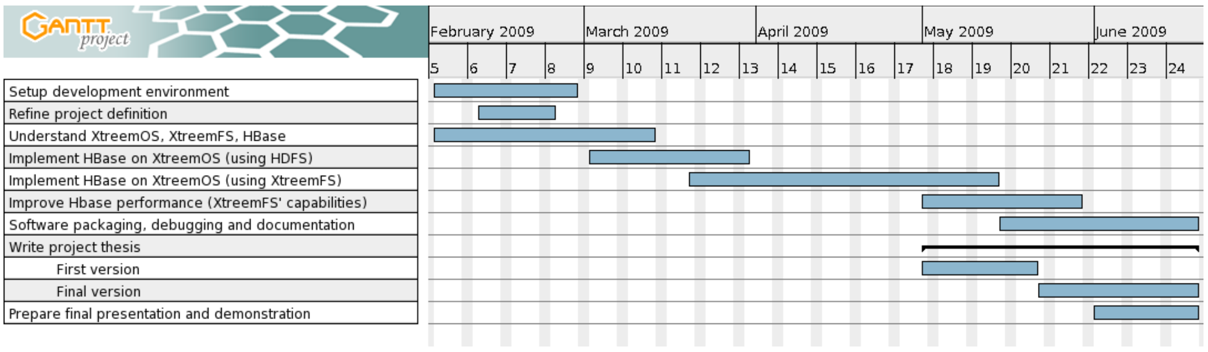


Figure 4.2: Outline of planned activities in the project

4.6 Risk analysis

The following is a list of possible problems which may or may not occur during the project. For each situation, a solution is provided in order to resolve or (even better) avoid it:

Situation	Solution
There are not enough XtreemOS nodes available to do experiments on	Reserve nodes early
Any of the project members is (too) busy to schedule an appointment	Plan the appointment earlier
XtreemOS, XtreemFS or HBase contains bugs	Report the bugs at the appropriate mailing list
Hardware failures of any of the development machines	Make backups often on separate storage
HBase does not support a feature required by XtreemOS	Ask HBase developers for help, or implement it ourselves

4.7 Quality assurance

In order to assure quality of all the project deliverables, Guillaume, Zhou and XtreemOS developers can review each deliverable, either remotely via the internet, or at the *Vrije Universiteit Amsterdam*. For the quality of each deliverable, it is important that it is reviewed by a person which has sufficient technical knowledge about the implementation of the deliverable.

4.8 Costs and gains

The following is a list of costs and gains for this project:

Hardware must be available, running XtreemOS.	COST
Some time and energy of each project member.	COST
Distributed scalable database implementation on XtreemOS.	GAIN
Experience and knowledge of distributed, scalable databases.	GAIN
Bugreports (and possible fixes) in any software used.	GAIN

Chapter 5

Implementation

Contents

In this section we describe the current implementation of HBase on XtreemOS. First we present the design decisions made for the implementation, and describe the programs we wrote. Then we describe how we deployed our implementation of HBase on XtreemOS and finally we present the performance results we measured.

5.1 Design

5.1.1 HBase requirements

As described in section 3.3.2, HBase is a Java program designed to run on many cheap Linux servers using a Java Virtual Machine of at minimum version 1.6.x. In a default HBase installation, each server requires the following:

- **Java 1.6.x**, preferably the Sun JVM although HBase should in theory work with other compatible JVM's.
- **Hadoop 0.19.x**, required in a default installation for providing the *Hadoop Distributed FileSystem* to HBase servers.
- **OpenSSH**, HBase needs the ssh command-line client to remotely start and stop the HBase master and region servers.
- **NTP**, the system time on each node in an HBase cluster is assumed to be in basic alignment. A *Network Time Protocol* client may be used to synchronize time on the nodes.
- **Configuration**, at minimum each HBase region server needs to know the location of the HBase master server, and the path to the shared filesystem between all HBase servers. This can be configured using the *hbase.master* and *hbase.root* configuration keys respectively. Additionally, in a default HBase setup the location of all region and master servers are stored in a textfile to automate SSH commands for starting and stopping HBase.

5.1.2 Java Virtual Machine

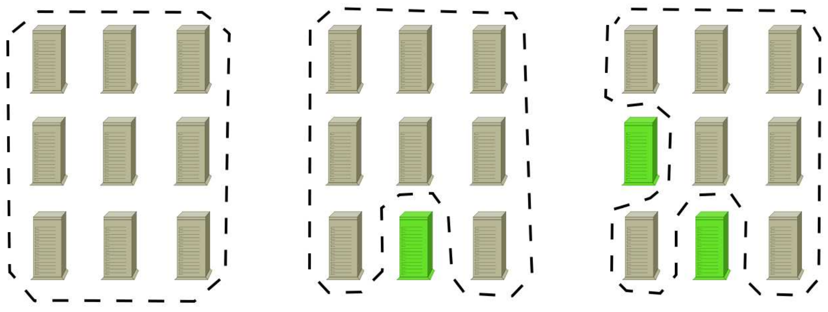
To provide a Java Virtual Machine of version 1.6.x or greater, we use *OpenJDK* in XtreemOS. OpenJDK is the open source edition of the Sun Java Virtual Machine. OpenJDK 1.7.0 is included in the latest current release of XtreemOS, version 1.1, and will also be included in the upcoming XtreemOS 2.0 release planned for June 2009.

5.1.3 Shared filesystem

In order to successfully run a HBase installation, each HBase region server needs to have access to a filesystem which is shared among all HBase region servers. HBase used various built-in filesystem modules provided by Hadoop which can be configured as the shared filesystem using the *hbase.root* configuration key, including the default HDFS, FTP, Amazon's S3 and the local filesystem. Each supported filesystem can be configured using the corresponding URL in the *hbase.root* configuration key, for example to use HDFS one would configure HBase as illustrated in figure 5.1.

```
<property>
  <name>hbase.rootdir</name>
  <value>hdfs://my.host.com:9000/hbase</value>
  <description>The directory shared by region servers.
    Should be fully-qualified to include the filesystem to use.
    E.g: hdfs://NAMENODE_SERVER:PORT/HBASE_ROOTDIR
  </description>
</property>
```

Figure 5.1: Example HBase configuration when using HDFS as shared filesystem



(a) Initially reserve all nodes.

(b) DIXI schedules the program on one node from the reservation. Now we reserve all remaining nodes.

(c) Keep reserving and submitting until enough programs running.

Figure 5.2: Scheduling jobs in DIXI on different resource nodes using reservations

To make using HBase easier on XtreemOS, we should replace the default HDFS with XtreemFS to provide a shared storage among HBase region nodes, as XtreemFS is already be available as shared storage among all XtreemOS resource nodes. As described in section 3.2.3, the XtreemFS client is built on top of the FUSE kernel module. This allows applications, including HBase running in the OpenJDK Java Virtual Machine, to access XtreemFS with regular POSIX functions on the local filesystem. Fortunately, HBase supports accessing the local filesystem using the *file://* URL in the *hbase.rootdir* configuration key, which we have used to point to the user's XtreemFS home directory. This way we where able to run HBase region servers using XtreemFS as shared storage. Therefore, to replace the Hadoop Distributed FileSystem (HDFS) with XtreemFS to provide a shared storage among all HBase region nodes, we did not need to modify the HBase source code.

5.1.4 Remote execution

In a regular HBase installation, the *bin/start-hbase.sh* and *bin/stop-hbase.sh* scripts are used to start and stop HBase using remote SSH commands, respectively. The location of each HBase region is written in the configuration file *conf/regionserver*, which is read by these scripts in order to connect to the right SSH daemon. Although XtreemOS supports executing SSH commands using XtreemOS certificates as authentication method, it is much harder in XtreemOS as a grid to maintain such a list, as resource nodes may enter and leave the grid at any time. The *DIstributed XtreemOS Infrastructure* (DIXI) was especially designed to execute programs on the XtreemOS grid, using the Application Execution Management (AEM) services as described in section 3.1. The DIXI Java API enables submission and monitoring of jobs, using JSDL [25] to describe jobs.

For running the HBase region servers on XtreemOS it is important that DIXI schedules them on different resource nodes, as otherwise there would be no performance gains if multiple HBase region servers are launched multiple times on the same machine(s), or worse it could overload them. Initially we hoped DIXI would be able to provide us the ability to submit one JSDL job, which would start multiple HBase region servers on multiple, unique resources nodes. Currently DIXI is able to process jobs which start multiple processes, however it cannot guarantee yet they are ran on different resource nodes. However, the latest DIXI version supports reservation of resource nodes. By reserving XtreemOS resource nodes, an application can supply DIXI with a number of machines of interest, and the time at which the application wants to use them. The application is then able to run a program on any of the reserved machines. It is then possible to only reserve resource nodes, which the application knows it has not yet started an instance of it's program, as illustrated in figure 5.2. Our current HBase implementation uses this method to run HBase on different XtreemOS resource nodes.

5.1.5 Time synchronization

HBase region servers are expected to have their system time synchronized to the system time of the HBase master server. On the HBase overview page [37] the developers suggest to install a *Network Time Protocol* client on each machine to synchronize their system times with an NTP server on the internet. In a grid environment such as XtreemOS it may not be guaranteed that each node has it's system time synchronized, and that all nodes reside in the same timezone. Our current implementation of HBase on XtreemOS does not deal with this problem and assumes that system times are correctly synchronized.

5.1.6 Configuration

HBase is configured using two XML files, *conf/hbase-default.xml* containing the default configuration and *conf/hbase-site* for user configuration. Configuration for HBase region servers is expected to be in a known location, and should be available for each HBase server. In a grid environment such as XtreemOS, no assumptions can be made on the contents of the local filesystem in each resource node, such as the location and access permissions of configuration files, as the resource nodes which HBase is scheduled on may change each time it is restarted with a new DIXI job. Therefore we cannot place HBase configuration on the local filesystem of XtreemOS resource nodes. Fortunately the XtreemFS user home directory provides a shared storage on which guaranteed to be available on all resource nodes. In our implementation we use XtreemFS to store the HBase configuration for both the HBase master and region servers. An advantage is that the user can configure all HBase servers using one configuration file, instead of configuring each region server individually as in the default HBase distribution. Administrators of XtreemOS resource nodes can override the HBase user configuration when appropriate. The *conf/hbase-default.xml* file on the XtreemFS is a symbolic link to the *conf/hbase-default.xml* configuration file in the default HBase installation directory, which is */usr/share/hbase*, to allow configuration overrides per XtreemOS resource node. The *conf/hbase-env.sh* file on the user's XtreemFS home directory, containing HBase environment variables, also reads the *conf/hbase-env.sh* if available to also allow overrides of environment variables. For example, HBase can be configured to consume a maximum amount of system memory with the *HBASE_HEAPSIZE* environment variable in the *conf/hbase-env.sh* file. Administrators of XtreemOS resource nodes can set this value to an acceptable level based on the amount of system memory available on the resource node.

5.1.7 Automating administration

To automate the process of submitting DIXI jobs to start and stop HBase on XtreemOS, we have written four python scripts. Python was chosen in preference to the *Bash Shell*, because our implementation needs to be able modify the HBase XML configuration files, especially the *hbase.master* configuration key. This configuration key contains the location of the currently active HBase master and needs to be overwritten each time the HBase master is started on the XtreemOS grid, as the resource node chosen by DIXI to run the HBase master may change. Shell scripts do not support an easy way to modify XML files. The Java programming language would be a good candidate, as the DIXI programming interface is written in Java aswell. However creating symbolic links in Java requires on to write a *Java Native Interface* module as it is not supported in the standard Java API. JNI modules are architecture specific, and this means we would have to create a different HBase package for each supported XtreemOS architecture, which should be avoided if possible. Moreover, code written in Python is often shorter, easier to read than Java programs, and does not require any compilation step. This avoids placing DIXI *Java Archive* files on the XtreemOS Subversion repository for HBase, which would otherwise become outdated in time. Figure 5.3 illustrates the interaction between the HBase python scripts, DIXI and XtreemFS.

5.2 User Programs

5.2.1 hbase-xos

Users can start and stop HBase on an XtreemOS grid using the *hbase-xos* program. It supports the actions *start*, *stop*, *restart*, *status* and *setup*. *setup* must be used by the user to initialize HBase configuration in the user's XtreemFS home directory before starting HBase on XtreemOS. *start*, *stop* and *restart* are used to start, stop and restart HBase on XtreemOS respectively. *status* may be used to output the current status of HBase, whether it is running and optionally on which resource nodes. *hbase-xos* optionally accepts command-line arguments to print out help information, enable verbose debugging output, override the location of the HBase configuration and data, the number of HBase regions to be started or stopped and the location of the XtreemOS user certificate to authenticate to DIXI. Also see appendix C for the complete source code of *hbase-xos* version 0.0.1.

5.2.2 hbase-xos-master

When submitting a job to DIXI to start the HBase master, *hbase-xos* constructs a JSDL to start the *hbase-xos-master* script. Once started on a XtreemOS resource node, it is responsible for overwriting the *hbase.master* configuration variable with it's own IP address. After updating the HBase configuration, it invokes an HBase master server. Although possible, this script should not be called directly by users.

5.2.3 hbase-xos-region

After *hbase-xos* has submitted a job to run the HBase master, it starts submitting JSDL jobs to run the *hbase-xos-region* program. It is used to start an HBase region server and like *hbase-xos-master* it should not be directly invoked by users.

5.2.4 hbase-xos-setup

To initialze the user's HBase configuration, *hbase-xos* submits a job to run *hbase-xos-setup* on any of the XtreemOS resource nodes. It creates initial XML files and points symbolic links to the appropriate files. Per default *hbase-xos-setup* initializes HBase configuration in *.hbase* in the users XtreemFS home directory.

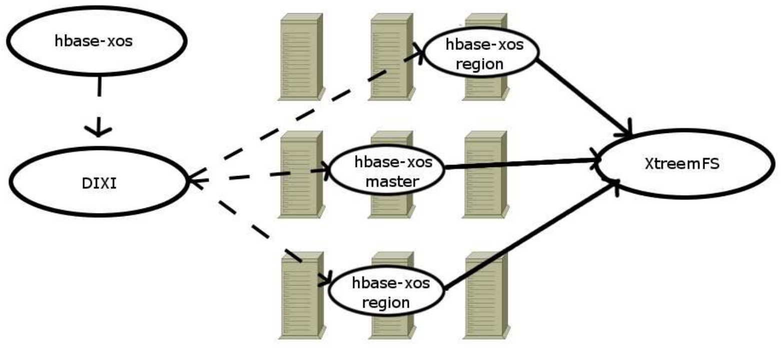


Figure 5.3: hbase-xos interacts with DIXI to submit HBase jobs to XtreemOS.

5.3 Deployment

5.3.1 RPM package

To make management of system programs easier for administrators, computer scientists have introduced the concept of a package manager. A package manager is a program which can configure, install and uninstall programs inside an package archive on an operating system automatically. Most modern package managers have the capability to keep track of installed programs and can determine and resolve dependencies needed when installing new programs. For example, when a system administrator decides to install program X on his machine the package manager fetches it's list of currently installed programs. The package manager knows that program X requires another program Y to function properly and verifies that it is installed. If not it should either install program Y aswell, or abort the whole installation process. Packages mostly have a special archive format which the package manager can read, in order to read various types of metadata about a package, such as it's name, maintainer, dependencies, architecture, release date and description. Package managers may install or update packages from several types of media, such as CD-ROM, DVD, HTTP, FTP or RSYNC. Popular examples of package management systems used today are the Debian Advanced Package Tool, Gentoo Portage, RPM and the FreeBSD ports collection.

To automate installation of programs on XtreemOS, the *RPM Package Manager* (RPM) was chosen. RPM is a package management system originally developed by Redhat for Redhat Linux and supports installation, configuration, uninstallation, verification, querying and updating software packages. RPM is capable of checking dependencies between packages, but it cannot automatically resolve them without help from an external program. Therefore several Linux distributions including Mandriva have build small scripts on top of RPM which are able to automatically detect and resolve dependencies by downloading the required packages from the internet and feeding them as input to RPM. The Mandriva distribution comes with several URPM scripts that are able configure remote RPM repositories, automatically download dependencies and update the entire system to the latest Mandriva release from a remote RPM repository. As XtreemOS is based on Mandriva, administrators can install pre-packaged RPM archives containing executable programs, libraries, configuration and documentation using Mandriva's URPM tools. For easy installation of HBase we have created a RPM package "hbase" for the default HBase installation and a RPM package "hbase-xos" containing the python scripts for managing HBase on XtreemOS.

5.3.2 Documentation

The *hbase-xos* program is documented in a regular UNIX manual page. UNIX is a computer operating system designed by Ken Thompson and Dennis Ritchie at Bell Labs. The philosopy of UNIX is to have many small programs that each can perform a simple task. For example, there are programs to create and delete files, list directory contents and query the current time of day. In a typical UNIX system the output of one program can be used as input for another program, thereby combining several simple programs to perform more complex tasks. Ken Thompson and Dennis Ritchie documented the parameters and workings of each program in a separate manual page. For easy searching the manual pages are divided in several sections, including User Commands, System Calls, Library Functions, Device and Special Files, File Formats and Conventions, Games, System Administration tools and Daemons and Miscellanea. Many UNIX-like operating systems have adopted the concept of manual pages to document their system, including Linux. On XtreemOS users can simply type "man hbase-xos" to view the manual page of the *hbase-xos* program, which describes it's workings and syntax, or then may use the *–help* command-line argument to view a brief description of it's syntax. Also see appendix B for the complete *hbase-xos* manual page.

5.4 Performance Evaluation

In this section we present the performance of HBase on XtreemOS we measured. For the performance tests we used three nodes from the VU testbed: *node004*, *node005* and *node007*. Each node had the following hardware and software installed:

- **Processor:** dual core 996.928 Mhz pentium III Coppermine
- **Memory:** 1 gigabyte RAM, 3 gigabyte swap
- **Disk:** 20 gigabyte Seagate ATA disk (ST320414A)
- **Network:** Intel Pro 100Mbit ethernet card
- **XtreemOS:** version 0.1
- **Kernel:** 2.6.20-0.5mdvsmp
- **Java:** OpenJDK 1.7.0
- **HBase:** version 0.19.2
- **XtreemFS:** version 1.0 RC1
- **Hadoop:** version 0.19.1

To put load on the HBase nodes we used the *PerformanceEvaluation* [38] program, which is included in the official HBase distribution. The PerformanceEvaluation program supports 5 different tests: random reads and writes, sequential reads and writes and scanning. It starts a given number of HBase clients as a local process or a MapReduce job. A similar number of HBase region servers should be started for each test, to spread the load generated by the clients among the available HBase region servers. Each client reads or writes per default 1 gigabyte of data to the HBase nodes. PerformanceEvaluation outputs the amount of time in milliseconds it took to complete the transactions for all clients. Using this information we calculate the average, aggregate rows processed by HBase per second with the following formulae:

$$Rows\ per\ Second = Total\ Rows\ /\ (Total\ Milliseconds\ /\ 1000)$$

For our tests we configured PerformanceEvaluation to read and write 512 megabyte of data, and used 64 megabyte regions in HBase. We first ran each test with the the latest XtreemFS public release available, version 1.0 RC1. Initially an older version of XtreemFS was installed on our testbed, that is version 0.10.1. With XtreemFS 0.10.1, we failed to complete the *PerformanceEvaluation* tests with more than one region server, as HBase reported *ChecksumException* errors on region splits. Region splits are critical for these performance tests, as it enables HBase to split the map in smaller pieces and distribute it among available region servers to spread the overall load. For comparison with the default Hadoop Distributed FileSystem (HDFS) included in HBase, we used version HDFS 0.19.1.

Each node ran an XtreemFS MRC and OSD service, or a HDFS Datanode respectively. *Node004* ran the HBase master, and *node007* the PerformanceEvaluation test, and all nodes ran an HBase region server as required by the tests. The XtreemFS and HDFS services where restarted on each test, to prevent caching to pollute the performance measurements. For the same reason all HBase master and region servers where restarted before each test was ran. The results of each test are plotted in the following sections. Also see appendix D for all tests and options supported by the PerformanceEvaluation program.

5.4.1 Sequential Read

Like BigTable, clients access HBase via a distributed map as described in section 3.3 and 3.3.2. Clients may read or write values from HBase given a row key, column key and version timestamp. Row keys are ordered lexicographically in HBase and are split in smaller ranges of rows called regions in HBase terminology. The regions are distributed among region servers. The *Sequential Read* test performs a single read operation on each row in HBase in lexicographical order, as illustrated in figure 5.5. Figure 5.4 displays the amount of 1000-byte rows read per second. When we analyze these results, XtreemFS seems to be almost twice as fast as HDFS on Sequential Reads. The straight increasing lines show that both filesystems scale very well.

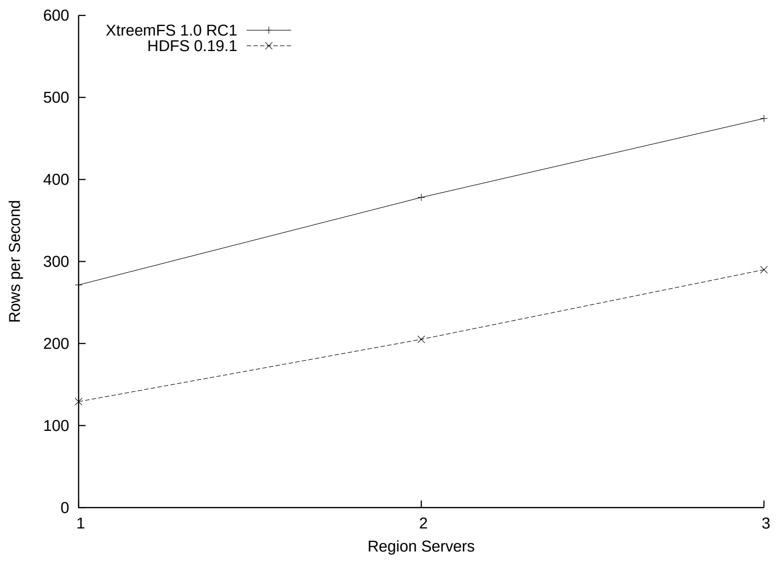


Figure 5.4: Performance results of the Sequential Read test

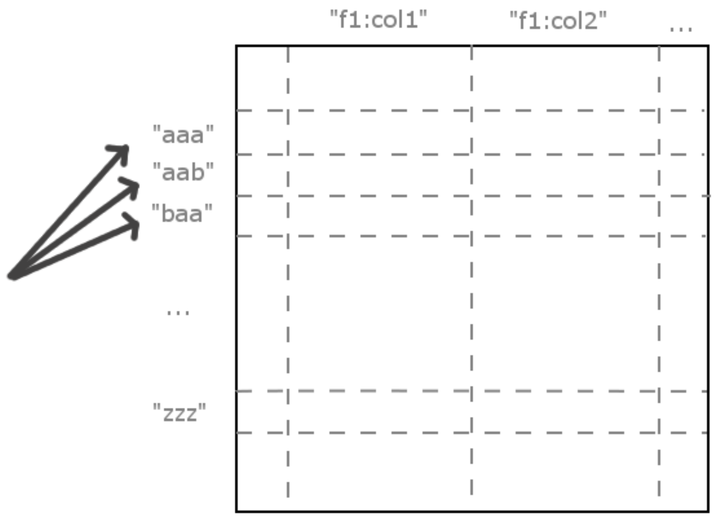


Figure 5.5: The Sequential Read test performs a read on each row key in order

5.4.2 Sequential Write

The *Sequential Write* performs the same test as the Sequential Read, except that it writes a new value to each row instead of reading, as figure 5.7 illustrates. Our performance measurements of Sequential Write are presented in figure 5.6. For Sequential Writes HDFS appears to be twice as fast as XtreemFS. The rows per second barely increase when adding region servers with XtreemFS, while HDFS shows an increasing line. However with three region servers, throughput seems to decrease with HDFS.

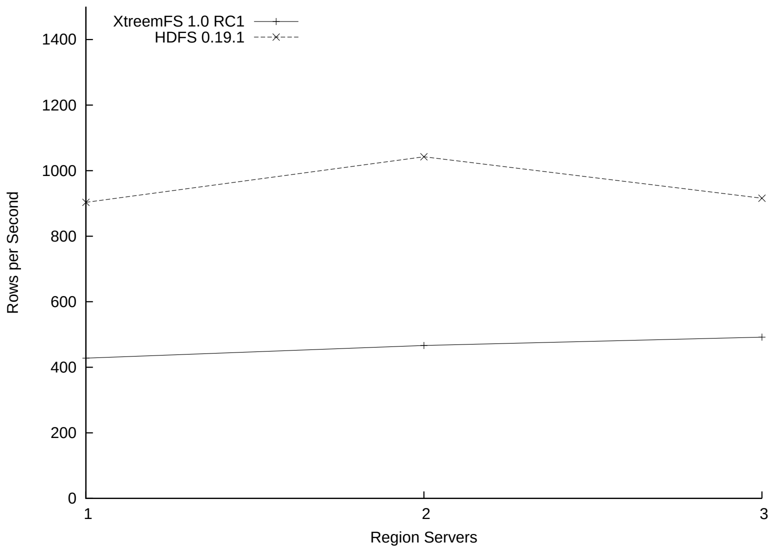


Figure 5.6: Performance results of the Sequential Write test

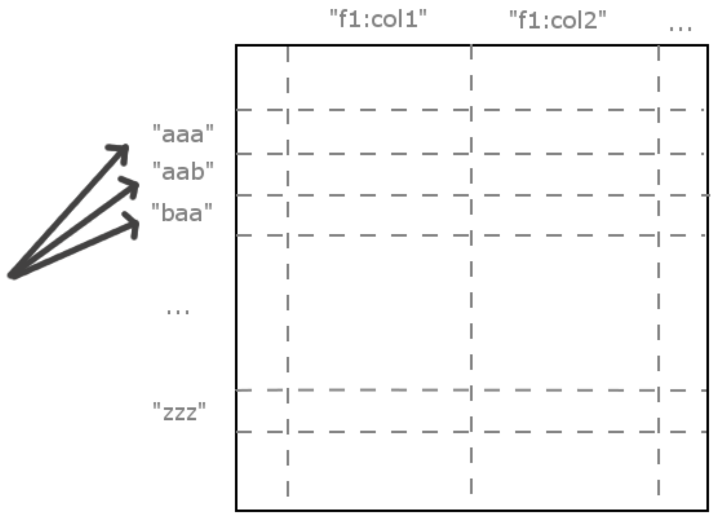


Figure 5.7: The Sequential Write test performs a write on each row key in order

5.4.3 Random Read

In practice, HBase clients may not read or write row keys in order. Therefore the *Random Read* test measures the performance of single read operations on all row keys in random order, as illustrated in figure 5.9. Figure 5.8 displays the performance results we measured on the VU testbed. As in the Sequential Read test, these results show that XtreemFS is faster on Random Reads than HDFS, and both scale very well when adding region servers.

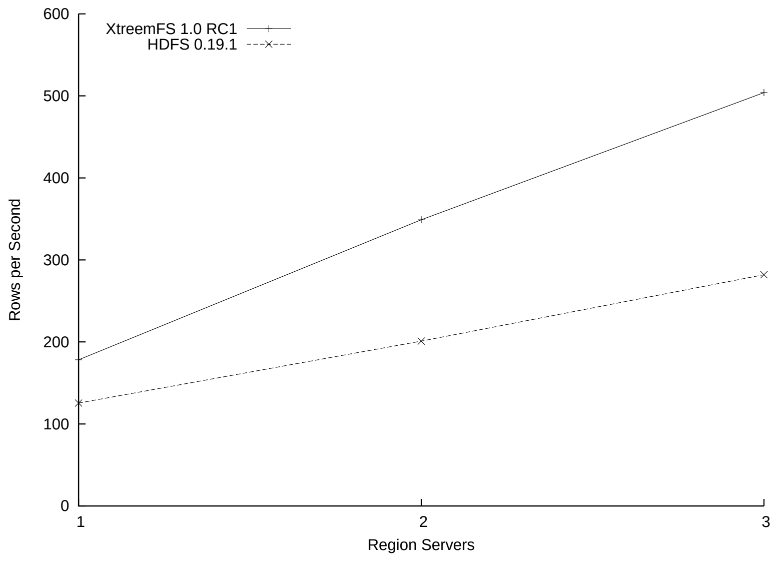


Figure 5.8: Performance results of the Random Read test

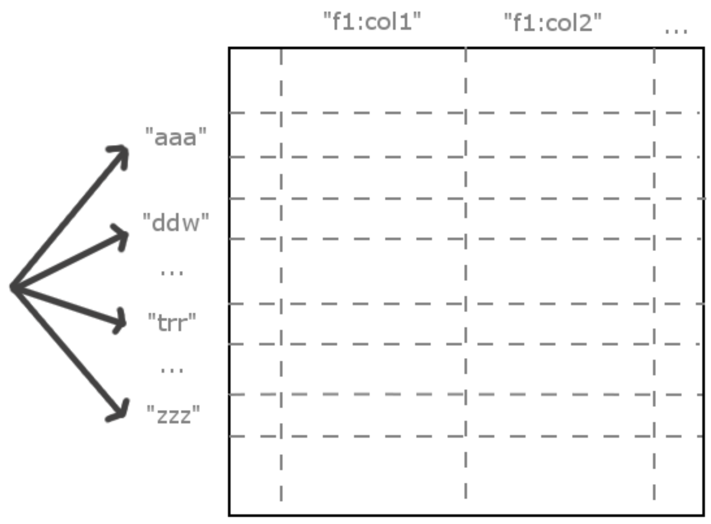


Figure 5.9: Random Read requests a read on each row key in random order

5.4.4 Random Write

Likes reads, HBase clients may not write to row keys in the order as they appear in the map, as illustrated in figure 5.11. *Random Write* measures HBase performance when writing to row keys in a random order. Figure 5.10 presents the results of the Random Write test. Like in Sequential Write HDFS is much faster than XtreemFS. XtreemFS shows a slight performance increase with three region servers, but a performance drop with two region servers. The decreasing line of HDFS shows that it does not scale very well with Random Writes.

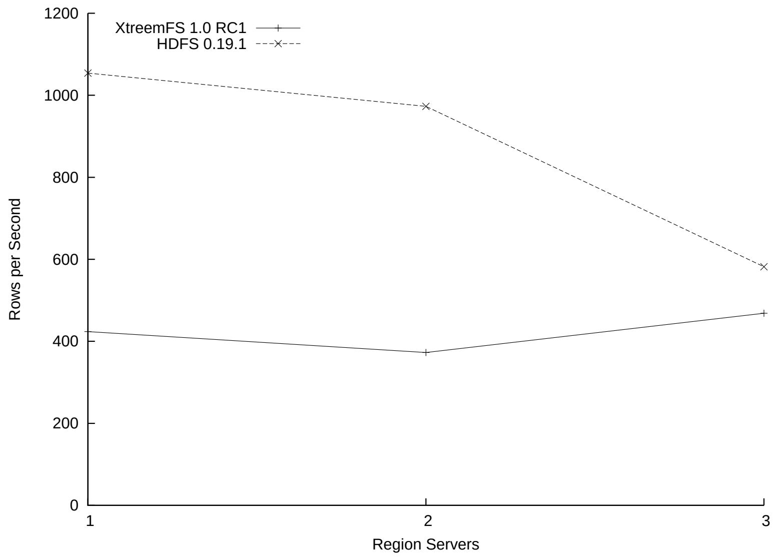


Figure 5.10: Performance results of the Random Write test

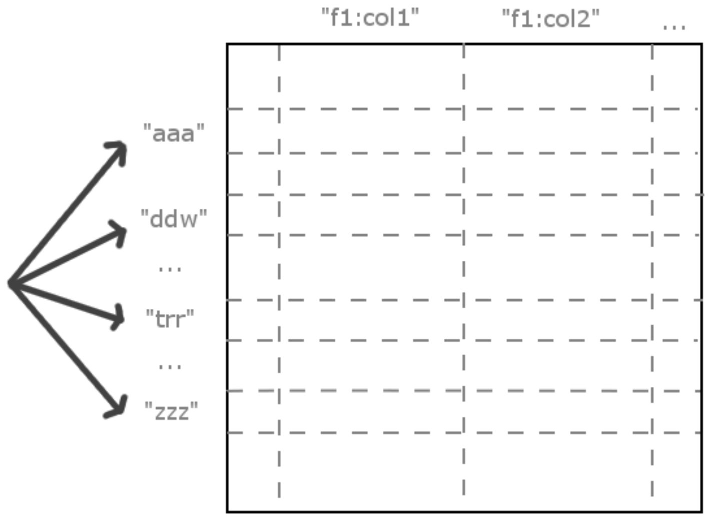


Figure 5.11: Random Write requests a write on each row key in random order

5.4.5 Scans

HBase supports scanning of row ranges. With a scan, HBase clients can read a range of rows a once, as illustrated in figure 5.13. Scanning reduces the communication overhead between the HBase client and region servers. Our scan performance measurements are presented in figure 5.12. For a single node setup, XtreemFS is faster than HDFS. However HDFS scans twice the rate of XtreemFS with more than one region server. HDFS shows better performance increases when adding region servers.

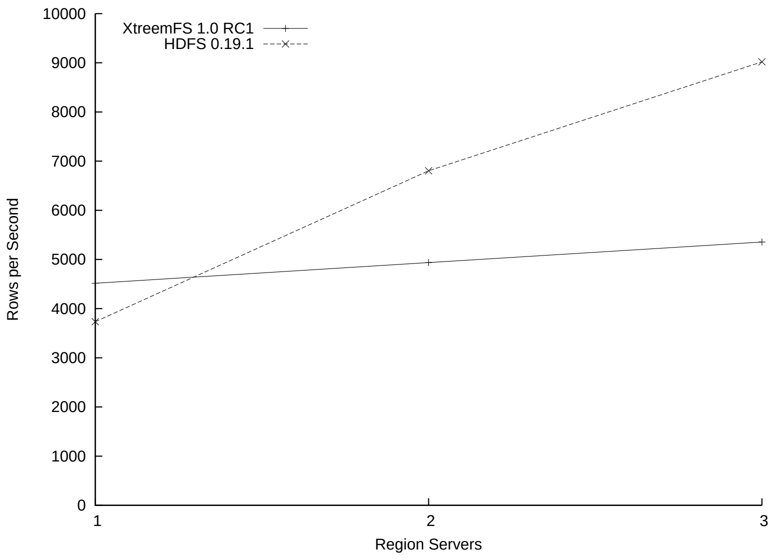


Figure 5.12: Performance results of the Scan test

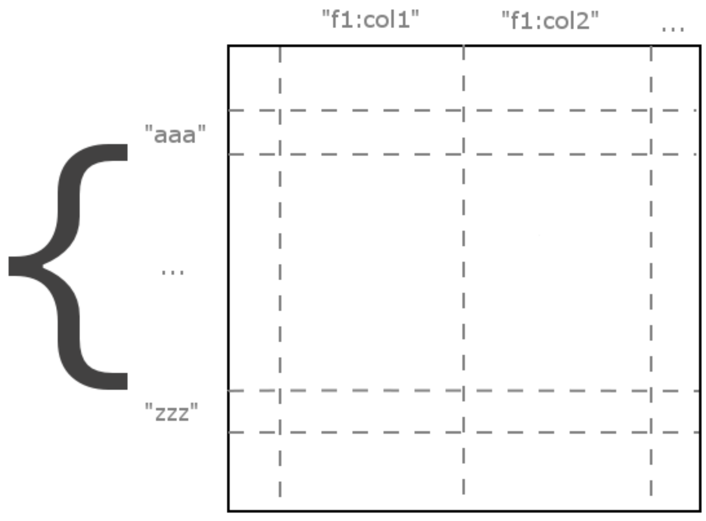


Figure 5.13: Scans request a read of a range of row keys

5.4.6 Discussion

In section 5.4 we presented the performance results measured on the VU testbed. Before conducting the tests, we hoped that HBase on XtreemFS would give us near liniar performance increase in terms of throughput as we added more region servers. The results of XtreemFS for the Random Read, Sequential Read, Sequential Write and Scan tests showed that throughput indeed increases with a straight, liniar line. Random Read appears to be the slowest operation, but manifests the best performance increase with more region servers. Scans are by far the fastest of all tests and show reasonable performance gains when adding region servers. Remarkable is the Random Write test, which is the only graph that does not show a straight line for XtreemFS. Two region servers seem to give a slower throughput than one region server, but with three region servers throughput does increase. We presume that XtreemFS may be inefficient in write operations under the circumstances of the Random Write test.

The performance results presented in the Google BigTable paper [1] show similar results that we measured in section 5.4. BigTable appears to also be fastest on the Scan test, and slowest with Random Reads. Google concludes that Random Reads show the worst scaling, but in our tests Random Reads appear to scale the best for XtreemFS. In our tests we only had few available nodes to benchmark HBase, where Google measured BigTable performance using 500 region servers. HBase might show similar Random Read scalability results when also benchmarked with 500 nodes. Compared to HDFS, XtreemFS is slow on writes but fast on reads, although writes stay the fastest on both filesystems. With both HDFS and XtreemFS, HBase shows good read scalability in our performance measurement results, but less scalability for write operations.

Although our performance measurements show a performance increase when adding more region servers, we did not have the resources in this project to measure with more than three nodes. The performance of HBase on XtreemFS may be different with hundreds or thousands of nodes.

Chapter 6

Project Evaluation

Contents

This section evaluates the various aspects of the project, by describing the learnings we have gained from the project, which experiments have been performed, difficulties and problems regarding the implementation and XtreemOS components encountered and how we have solved them, and finally we evaluate the project management.

6.1 Learnings

6.1.1 XtreemOS

Before this project, I had close to zero knowledge and experience with Apache HBase, XtreemOS and XtreemFS. Thanks to this project I now have a better understanding of highly scalable databases such as HBase, distributed filesystems like XtreemFS and grids as XtreemOS in general. It is also the first existing open source project on which I have made an contribution. Although I worked on a few of my own open source projects, XtreemOS is a very different environment. When creating a hobbist project one is free to design, implement and deploy the entire system in any way the author wants. In a large international project like XtreemOS this is impossible, as each developer works on a small piece of the whole puzzle. Developers need to communicate with each other to ensure that the different components cooperate.

In order to develop new functionality for an existing system, one first needs to understand it. Therefore I have spend most of February 2009 to learn the basics of XtreemOS, XtreemFS and HBase. XtreemOS is a system consisting of several layers, each consisting of multiple components and are used on multiple machines on the network. Their distributed naturealone already makes learning XtreemOS difficult for new developers. Additionally XtreemOS is a system in current development, in which user and developer documentations are often out of date, or do not yet exist. Therefore I learned how to use the XtreemOS API by reading the source codes available on the public Subversion repository, which takes a significantly longer time than reading user or developer documentation. The best way I have encountered to learn XtreemOS during this project was when I installed a few test virtual machines in VMWare on my personal workstation. By installing XtreemOS one is forced to understand each component that is configured and started. At the current state of XtreemOS the system does not work out-of-the-box, meaning that additional configuration is required after installation Configuring the components often results in unexplained errors or behaviour, forcing one to understand even better what each component should do.

6.1.2 XtreemFS

In my experience XtreemFS proved easy to learn. On the XtreemFS homepage [7] there is detailed information available about it's internals, configuration and user interfaces. XtreemFS requires minimal configuration in a standard installation on a few nodes and thanks to the FUSE client it allows transparent access to existing programs. During this project I had very few problems with XtreemFS, except during the benchmark experiments of HBase which is evaluated in section 5.4. Version 0.10.0 seemed to cause *ChecksumException* errors in HBase, and random disconnects on mounted volumes. Before using XtreemFS in this project, I had used several distributed filesystems before, including NFS and SSHFS, but I never used or heard of object-based filesystems. Object-based filesystems like XtreemFS offer many advantages over existing distributed filesystems like NFS, such as parallel transfers and replication. I believe distributed object-based filesystems like XtreemFS offer a good solution to provide scalability to end-users.

6.1.3 HBase

Although I used databases in many of my school projects at the *Hogeschool Utrecht* such as MySQL, SQLite and Oracle, I never needed more scalability for those projects. Therefore I did not need to put them in clustered mode to scale further, or even think of alternatives such as HBase. Learning the concepts of BigTable and HBase has broadened my knowledge of databases, scalability and distributed systems in general. During this project I have experimented with HBase on several Linux test installations and XtreemOS and in my experience it is easy for a new user to get started with HBase. Installation, configuration and basic functionality of HBase is very well documented on their Apache project page [13] which was the primary source of information I used regarding HBase.

6.1.4 English

In this project we have had several meetings to review the project status, decide about design and implementation steps and to discuss this thesis. Thanks to the conversations I had with Guillaume, Corina and several other people at the VU, I had the opportunity to practise my English speech skills. Writing the project documents, technical documentation and this thesis helped me improve my English writings aswell.

6.2 Development evaluation

6.2.1 Default HBase on XtreemOS

During the project we have performed several experiments and testings to examine the functionalities and workings of HBase and the XtreemOS API. By experimenting with HBase and XtreemOS, we have incrementally implemented the *hbase-xos* Python scripts and RPM package. First we tried to run HBase placed in */usr/local/hbase* on all XtreemOS nodes without any modifications as a regular UNIX user, which worked without problems. Then we looked for a way to override the default configuration and data paths in HBase without modifying the source code. To achieve this we used the *HBASE_CONF_DIR* environment variable to point to the users home directory, and placed HBase in */usr/share/hbase*.

6.2.2 HBase on top of XtreemFS

Before we started the project, we had little knowledge of HBase and XtreemFS system internals and anticipated that it would take a large part of the project to modify HBase to use XtreemFS as the distributed filesystem. As described in section 5.1.2, we managed to implement HBase on top of XtreemFS without any modifications, by using the *file://* URL. Thus, it took us exactly two days to fullfill this requirement versus half the project.

6.2.3 Running as the VO-user

Now that we had a working HBase implementation on top of XtreemOS, we attempted to invoke HBase as a Virtual Organization user. In XtreemOS the OpenSSH server allows remote logins and has been modified to perform authentication using XtreemOS certificates. This enables users inside a specific Virtual Organization to log in with *ssh-xos* to remote systems inside the VO using their XtreemOS certificate. When trying to launch and run HBase on the *ssh-xos* command line, we discovered a very poor system response. It looked like each time an operation was performed on the XtreemFS filesystem on the *ssh-xos* shell, the system hung for about two seconds. It took us about a week to find out that the *Account Mapping Service* was the cause of the problem. The Account Mapping Service in XtreemOS performs a translation from global user identifiers to local system accounts. It uses a database to maintain this mapping, which was used in a very inefficient way. This issue prevented us to normally run HBase as the Virtual Organization user, as HBase uses XtreemFS intensively. To workaround the problem, we installed a few XtreemOS VMware machines, and modified to Account Mapping Service code to always return the same mapping to temporarily avoid the performance problem. Several weeks later the developers provided us with a partly working solution to the problem, but as of this writing it has still broken the AMS daemon on the *node004* server at the VU testbed.

6.2.4 Invoking HBase using DIXI jobs

The core functionality of XtreemOS is to enable submission and execution of programs on the grid. As described in section 3.1, DIXI accepts JSDL jobs descriptions to run programs on the AEM service in XtreemOS. Currently, DIXI does not support the full JSDL 1.0 standard. Unfortunately there is currently no developer documentation in DIXI. Therefore there is no other place in XtreemOS than the DIXI source code to find which JSDL features are supported by DIXI. The DIXI source code consists of thousand of lines of Java and small bits of C code, which takes a long time to read and understand. For example, it took us several weeks to discover that DIXI did not correctly implement the <*Argument*> tag of JSDL 1.0, but thanks to the quick response of the DIXI developers this problem is now solved.

The majority of time invested was invested in exploring the current functionality and features supported by DIXI. By experimenting with JSDL jobs, reading the DIXI source code and sending questions to the DIXI developers we did achieve to successfully run HBase on multiple nodes in the XtreemOS grid by using reservations, as described in section 5.1.3. However if DIXI would have supported submitting one job to start a program multiple nodes, and could guarantee that each program ran on a different machine, this would have been the preferred solution. That way *hbase-xos* would be more simple, and DIXI could make the reservations transparently if needed.

6.2.5 Implementing hbase-xos

To automate the process of submitting HBase jobs to the XtreemOS grid, we wrote several Python programs as described in section 5.1.7. The Python programming language is very well documented in the publicly available Python documentation page [39], several example programs on the internet and also in books [40] written about Python. We experimented with DIXI on XtreemOS as we wrote the *hbase-xos* Python program.

6.3 XtreemOS experiences

6.3.1 Installation

During this project, we have installed XtreemOS several times using different CD-ROM images. Around february we have used XtreemOS 0.1, which was the first public release. Although it contained all XtreemOS components, all configuration needed to be done manually. This means that creating configuration files, XtreemOS certificates, Virtual Organizations and adding resource nodes was a time expensive activity, regardless of the many programming errors still in the system. In the 0.1 release there was no way to specify during setup which type of XtreemOS node was being installed.

XtreemOS version 1.1 was internally released around April. It contained improvements for each individual component, but also made installation of XtreemOS slightly easier, as administrators now had the option to install a different type of XtreemOS node: core, resource or client. Dialog boxes where now used to initially configure the AEM services, and several configuration files where pre-generated, as well as some test XtreemOS certificates. Although version 1.1 has made improvements to make installing XtreemOS easier, it still takes an average administrator at least a day to figure out how it should be configured.

6.3.2 Functionality

As XtreemOS is a project in current development, there is functionality which has not yet been implemented. During experiments with DIXI, as described in section 6.1.4, we encountered and reported missing functionality which where relevant to the DIXI developers, such as support for reliable multiple job submission and the JSDL <Argument> tag. In general XtreemOS is ready to install a simple grid, although much work has to be done to make installation and configuration easier, and stabilizing the system as a whole.

6.3.3 Documentation

There is an XtreemOS user guide [41] available on the in XtreemOS homepage, but it has not been updated since december 2008. Many components have changed since that time, including their installation steps and configuration files. DIXI does not have a developer guide available yet, but has some comments inside the source codes which allow automatic generation of JavaDoc documentation. These source code comments have helped us during the project to learn how to use the DIXI API. Except for the XtreemFS client commands which have been listed in section 3.2.3, none of the XtreemOS commands have manual pages. This required us to read the user guide or their source code to understand how to use them.

6.3.4 Source code

The source code of XtreemOS is available on the public Subversion repository. The latests and previous versions of all XtreemOS components can be downloaded using a recent Subversion client. In our experience, the source code of XtreemOS currently often lacks comments, is sometimes poorly indented and is overall very big, which makes it hard to understand it at a first glance.

6.4 Project management

As we described in section 4.1, the goal of the project was to implement a scalable database on XtreemOS. Our implementation presented in section 5 fullfills this that goal. There are some possible improvements and limitations, but the project goal has been achieved. According to our original planning, we would have spend most of march and april modifying HBase to use XtreemFS, but as we noted in section 6.1.4 we spend that time experimenting with DIXI. The rest of the original planning roughly reflects the actual time spend, except that we packaged HBase mid-April, and started writing this thesis early-April.

Before starting the project, we agreed with several deliverables. The first one was the implementation of HBase using HDFS as distributed filesystem on XtreemOS. As section 6.1.4 describes, we could run HBase using HDFS on XtreemOS without problems. Because XtreemFS has an FUSE client, we where able to use XtreemFS as distributed filesystem for HBase within a few days, fulfilling the second requirement. Therefore, modifying the HBase source code to improve XtreemFS performance would be pointless with FUSE. Packaging scripts have been implemented in RPM archives, however when planning this project we did not expect they would be included in the next public XtreemOS release. Technical documentation has been provided by a manual page describing the syntax and arguments of the *hbase-xos* program, and source comments.

Chapter 7

Conclusion

In this project we implemented HBase on XtreemOS. Although XtreemOS is in current development, running a scalable distributed database like HBase on an XtreemOS grid is possible. We packaged HBase for the second public XtreemOS release in June 2009. Performance measurements on the VU testbed show that HBase on top of XtreemFS is scalable. Our implementation of HBase on XtreemOS is functional, but there are some known limitations and improvements to be made:

- **XOSAGA**: the XOSAGA API was designed to make programming grid applications easier, and also supports a Python API. XOSAGA can simplify the implementation of *hbase-xos*. The *hbase-xos* program would then no longer need to generate JSDL files, import DIXI Java classes, perform reservations or know about DIXI in the first place. However like DIXI, XOSAGA is under active development. During the last weeks of May, developers released version 0.2.0 of the XOSAGA python API. According to the developers that version should support running HBase jobs on different XtreemOS resource nodes, internally also using DIXI reservations.

- **Time Synchronization**: We did not deal with the time synchronization between HBase servers, as this is much harder in grids than it would be in clusters. In grids, nodes may span multiple administrative domains and thus also multiple time zones. Although the HBase website [37] suggest that time synchronization is needed, perhaps future improvements in HBase may remove this requirement.

- **Authentication**: Since the HBase is developers are currently focussed on improving the overall performance of HBase, there is no access control or authentication mechanism implemented yet in HBase. Whenever this functionality becomes available in HBase, XtreemOS could benefit by using XtreemOS certificates for authentication.

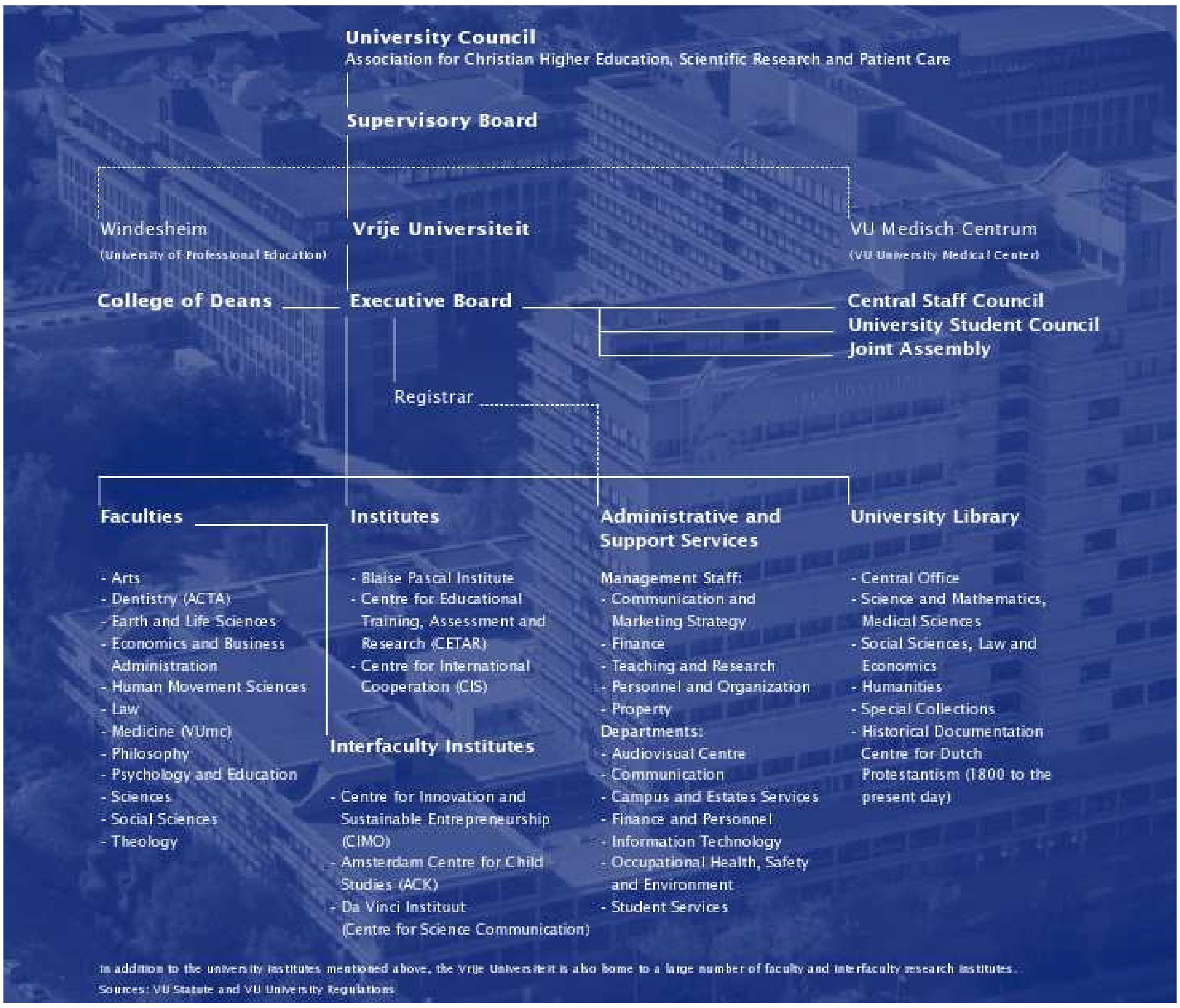
- **RPM dependencies**: We have packaged default HBase distribution in XtreemOS, but several of it's dependencies are not. As a temporary solution, we packaged those dependencies inside our RPM. Properly packaging a program is a time consuming task, therefore we did not have the time in this project to package all HBase dependencies.

- **HBase instances**: HBase consumes a significant amount of system resources. For this reason, our implementation supports only one HBase instance running at a XtreemOS resource at the same time. This means only one Virtual Organization user may run HBase on the same resource node.

Nowdays there is active research and development going on in grid computing, including the XtreemOS project. Having a cloud computing infrastructure on XtreemOS can offer several advantages and benefits to users. By implementing a scalable database, we contributed towards the realization of a cloud computing platform on XtreemOS.

Appendix A

Vrije Universiteit Organigram


University Council
Association for Christian Higher Education, Scientific Research and Patient Care
Supervisory Board
Windesheim
(University of Professional Education)
Vrije Universiteit
VU Medisch Centrum
(VU University Medical Center)
College of Deans
Executive Board
Central Staff Council
University Student Council
Joint Assembly
Registrar
Faculties
- Arts
- Dentistry (ACTA)
- Earth and Life Sciences
- Economics and Business Administration
- Human Movement Sciences
- Law
- Medicine (VUmc)
- Philosophy
- Psychology and Education
- Sciences
- Social Sciences
- Theology
Institutes
- Blaise Pascal Institute
- Centre for Educational Training, Assessment and Research (CETAR)
- Centre for International Cooperation (CIS)
Interfaculty Institutes
- Centre for Innovation and Sustainable Entrepreneurship (CIMO)
- Amsterdam Centre for Child Studies (ACK)
- Da Vinci Instituut (Centre for Science Communication)
Administrative and Support Services
Management Staff:
- Communication and Marketing Strategy
- Finance
- Teaching and Research
- Personnel and Organization
- Property
Departments:
- Audiovisual Centre
- Communication
- Campus and Estates Services
- Finance and Personnel
- Information Technology
- Occupational Health, Safety and Environment
- Student Services
University Library
- Central Office
- Science and Mathematics, Medical Sciences
- Social Sciences, Law and Economics
- Humanities
- Special Collections
- Historical Documentation Centre for Dutch Protestantism (1800 to the present day)

In addition to the university institutes mentioned above, the Vrije Universiteit is also home to a large number of faculty and interfaculty research institutes.
Sources: VU Statute and VU University Regulations

Appendix B

HBase-Xos Manual Page

```
HBASE-XOS(1)                                                     HBASE-XOS(1)



NAME
       hbase-xos - Run Apache Hbase on an XtreemOS grid


SYNOPSIS
       hbase-xos [OPTIONS] {start|stop|restart|status|setup}


DESCRIPTION
       hbase-xos  is  a  python script which can be used to run Hbase on XtreemOS. It can
       start, stop and restart Hbase on any available nodes on the XtreemOS grid, by sub-
       mitting  jobs using DIXI.  Hbase configuration and data is typically stored in the
       user's XtreemFS automount in ~/.hbase and may be initialized using the setup  com-
       mandline argument.  hbase-xos automatically updates the hbase.master configuration
       value in ~/.hbase/conf/hbase-site.xml with the hostname of the  node  running  the
       Hbase master.


OPTIONS
       -h --help
              Display a help screen and quit.


       -v --verbose
              Instructs  hbase-xos  to  output debug messages during execution. Use twice
              for extra verbosity.


       -H --hbase-homedir=DIR
              Specifies  the  Hbase  installation  directory,  which  contains  binaries,
              libraries  and  configuration  required  to run Hbase. The default value is
              /usr/share/hbase.
       -u --hbase-userdir=DIR
              Sets the user's Hbase configuration and data directory to the given  value.
              The  given  directory  must  be  shared amoung all Hbase nodes, e.g. on the
              XtreemFS automounted home directory.  The default value is ~/.hbase


       -r --regions=NUM
              Use the given number of region server nodes on the grid. This is a required
              argument for the start and stop actions.


       -c --certificate=FILE
              Sets  the  XtreemOS  certificate to be used for job submission. The default
              value is ~/.xos/truststore/certs/user.crt


ENVIRONMENT
       The environment variable HBASE_CONF_DIR must point to the user's  hbase  directory
       to  use  the normal hbase(1) command. This way it can determine which node is cur-
       rently configured as the Hbase master.


EXAMPLES
       First initialize the user hbase directory:

             client$ hbase-xos setup

       Then, start Hbase with 20 region servers on the grid from an XtreemOS client node.

             client$ hbase-xos -r 20 start

       Verify that they have started correctly:

             client$ hbase-xos -v status

       Launch the Hbase shell to administer the database as the VO user on the ssh-xos(1)
       shell:

             vouser$ hbase shell

       And finally, stop all nodes again:

             client$ hbase-xos stop


SEE ALSO
       python(1) xsub(1) xconsole_dixi(1)


BUGS
       The XtreemOS bugtracker is available on http://gforge.inria.fr/tracker/?group_id=411.


AUTHOR
       Niek Linnenbank <nieklinnenbank@gmail.com>


April 2009                        XtreemOS manual                        HBASE-XOS(1)
```

Appendix C

HBase-Xos Program

```
#!/usr/bin/env jython
#
# This script runs a Hbase master, and a given number
# of region servers on an XtreemOS grid. A possible improvement
# for this script may be to use the python XOSAGA API, if it supports
# submitting XtreemOS jobs on different nodes.
#

# Import standard python modules.
import sys
import os
import getopt
import socket
import tempfile
import time

# We require these JAR files.
sys.path.append('/usr/share/java/xati.jar')
sys.path.append('/usr/share/java/DIXIMain.jar')
sys.path.append('/usr/share/java/DIXIMainServices.jar')
sys.path.append('/usr/share/java/aem-node.jar')
sys.path.append('/usr/share/java/bcpg.jar')
sys.path.append('/usr/share/java/bctsp.jar')
sys.path.append('/usr/share/java/bcel.jar')
sys.path.append('/usr/share/java/bcprov.jar')
sys.path.append('/usr/share/java/log4j.jar')
sys.path.append('/usr/share/java/mina-core.jar')
sys.path.append('/usr/share/java/slf4j-api.jar')
sys.path.append('/usr/share/java/slf4j-simple.jar')

# Import them in our execution environment.
from eu.xtreemos.xati.API import XJobMng
from eu.xtreemos.xati.API import XResMng
from eu.xtreemos.xati.API import XReservationManager
from eu.xtreemos.system.communication.net import CommunicationAddress
from eu.xtreemos.xosd.utilities.security import Utils
from eu.xtreemos.xosd.metrics import MetricsDesc
from eu.xtreemos.xosd.metrics import MetricScope
from eu.xtreemos.xosd.metrics import MetricType
from eu.xtreemos.xosd.utilities.jobinfo import JobInfoList
from eu.xtreemos.xosd.utilities.metrics import InfoLevel
from eu.xtreemos.xosd.utilities.metrics import TypeOfInfo
from eu.xtreemos.xosd.localallocmgr.attributes import *
from eu.xtreemos.xosd.reservationmanager.base import *
from eu.xtreemos.xosd.localallocmgr.basic import *
from eu.xtreemos.xosd.localallocmgr.frontend import *
from eu.xtreemos.xosd.localallocmgr.frontend.utils import *
from eu.xtreemos.xosd.utilities.security import UserCertificateUtility
from org.bouncycastle.jce.provider import BouncyCastleProvider
from org.apache.log4j import Logger
from java.util import ArrayList
from java.util import GregorianCalendar
from java.lang import Thread
from java.lang import String
from java.security import Security
from java.security.cert import X509Certificate

# Hbase installation directory.
hbase_homedir = "/usr/share/hbase"

# Hbase user configuration and data directory
hbase_userdir = None

# Path to the user's XtreemOS certificate
cert_path = os.getenv("HOME") + "/.xos/truststore/certs/user.crt"

# XtreemOS X509 certificate.
cert = None

# Number of Hbase region servers to start.
hbase_num_regions = 0

# Output verbose messages. The higher the more verbose.
hbase_verbose = 0

# Initialize bouncycastle.
bc = BouncyCastleProvider()
Security.addProvider(bc)

#
# Reserves the given nodes.
#
def reserveNodes(nodes, cert, startTime, endTime):

    CPU0 = "CPU0"
    requests = ArrayList()

    # Loop all nodes.
    for node in nodes:

        # Instantiate a new request.
        request  = ReservationRequest()
        request.nodeAddress  = node
        request.localRequest = Request()

        # Fill in the timeslots and attributes.
        ttelm = TTElmFactory.createBasic(startTime, endTime, SharingValues.MUTUAL)
        TTElmFactory.addOwnerInfo(ttelm, OwnersInfo("hbase-xos", "hbase-xos"))
        TTElmFactory.addAttribute(ttelm, CurrentAmount(1))
        elmreq = TTElmRequestAdd(CPU0, ttelm)
        request.localRequest.add(elmreq)
        requests.add(request)

    # Output debug trace.
    if hbase_verbose >= 2:
        print "Invoking: XReservationManager.createReservationExplicit(requests, cert)"

    # Now send the requests to the reservation manager.
    reservationId = XReservationManager.createReservationExplicit(requests, cert)
    if reservationId == None:
        print 'Failed to create reservation for nodes: ' + str(nodes)
        sys.exit(1)

    # Success.
    return reservationId

#
# Execute a job on the given resource nodes.
#
def performJob(jsdl, nodes, cert, serverType):

    # Fill in timestamps.
    startTime = GregorianCalendar()
    startTime.add(GregorianCalendar.SECOND, 3)
    endTime   = startTime.clone()
    endTime.add(GregorianCalendar.MINUTE, 2)

    if hbase_verbose >= 1:
        print str(nodes),

    # Reserve the node.
    resID = reserveNodes(nodes, cert, startTime, endTime)
    jobID = XJobMng.createJob(jsdl, False, resID, cert)

    # Wait for the timeslot to arrive.
    check = GregorianCalendar();
    while (check.before( startTime) ):
        check = GregorianCalendar()

        if hbase_verbose >= 1:
            print ".",
        Thread.sleep(1000)

    # Output debug marker.
    if hbase_verbose == 1:
        print "!",
    elif hbase_verbose >= 2:
        print "Invoking: XJobMng.runJob(" + jobID + "," + \
               resID + ",cert)",
    else:
        print ".",

    # Run the job now!
    if XJobMng.runJob(jobID, resID, cert) == -1:
        print 'Failed to run jobID ' + jobID
        sys.exit(1)
    # Set a job metric, containing the type of HBase server: master or region.
    Thread.sleep(2000)
    XJobMng.addJobMetric(jobID, MetricsDesc("hbase", MetricType.string_t, \
                                "hbase server", MetricScope.job, False, None), cert)
    XJobMng.setMetricValue(jobID, "hbase", None, None, serverType, cert);

    # Free reservations.
    XReservationManager.releaseReservation(resID, cert)

    # Return the node on which we run.
    info = XJobMng.getJobInfo(jobID, TypeOfInfo.BASIC.val(), \
                                     InfoLevel.PROCESS.val(), None, cert)
    return JobInfoList(info).getJobResources(jobID).get(0)

#
# Prints usage information.
#
def usage():
    print "usage: hbase-xos [OPTIONS] {start|stop|restart|status|setup}"
    print "Run Apache Hbase on an XtreemOS grid."
    print ""
    print "-h, --help               Print help information"
    print "-v, --verbose            Output debug messages"
    print "-H, --hbase-homedir=DIR  Hbase installation directory"
    print "-u, --hbase-userdir=DIR  Hbase user configuration/data directory"
    print "-r, --regions=NUM        Number of Hbase region servers"
    print "-c, --certificate        XtreemOS certificate to use"

#
# Generate a JSDL for job submission via xsub.
#
def generateJSDL(prog,args,out,err):

    return ("<JobDefinition xmlns=\"http://schemas.ggf.org/jsdl/2005/11/jsdl\">\n"
            "   <JobDescription>\n"
            "       <JobIdentification>\n"
            "           <Description>Runs " + prog + " on an XtreemOS grid</Description>\n"
            "           <JobProject>" + prog + "</JobProject>\n"
            "       </JobIdentification>\n"
            "       <Application>\n"
            "           <POSIXApplication xmlns=\"http://schemas.ggf.org/jsdl/2005/11/jsdl-posix\">\n"
            "                <Executable>" + prog + "</Executable>\n"
            "                <Output>" + out + "</Output>\n"
            "                <Error>"  + err + "</Error>\n"
            "                <Argument>" + args + "</Argument>\n"
            "           </POSIXApplication>\n"
            "       </Application>\n"
            "       <Resources>\n"
            "           <TotalResourceCount>\n"
            "               <Exact>1</Exact>\n"
            "           </TotalResourceCount>\n"
            "       </Resources>\n"
            "    </JobDescription>\n"
            "</JobDefinition>\n")
```

```
#
# Get an ArrayList<String> of jobs running
# an HBase server of the given type (master or region).
#
def getActiveJobs(type):

    jobIds = XJobMng.getJobsUser("", cert);
    list   = ArrayList()

    # Do we have any jobs?
    if jobIds == None:
        return ret

    # Walk them all.
    for job in jobIds:

        try:
            if hbase_verbose:
                print "Job: " + job

            # Retrieve job information in XML format.
            info = XJobMng.getJobInfo(job, TypeOfInfo.USER_METRICS.val() | \
                                      TypeOfInfo.BASIC.val(), InfoLevel.PROCESS.val(), \
                                      None, cert)

            if len(info) > 0:
                infoList = JobInfoList(info)

                # Find out if this job is an HBase instance of the given type.
                if infoList.getMetricValue(job, "hbase").getValue() == type and \
                   infoList.getMetricValue(job, "jobStatus").getValue() != "Done":
                    if hbase_verbose:
                        print "Job matches HBase for " + type + " : " + job
                    list.add(job)
        except:
            pass

    return list

#
# Get a list of currently active master servers.
# Note: currently, there can be only one master in Hbase,
#       but this may change in the future.
#
def getActiveMasters():

    list    = ArrayList()
    jobInfo = None

    for masterJob in getActiveJobs("master"):

        info = XJobMng.getJobInfo(masterJob, TypeOfInfo.BASIC.val(), \
                                  InfoLevel.PROCESS.val(), None, cert)
        jobInfo = JobInfoList(info)
        list.add(jobInfo.getJobResources(masterJob).get(0))

    return list
```

```
#
# Retrieve the list of currently active region servers.
#
def getActiveRegions():

    list    = ArrayList()
    jobInfo = None

    for regionJob in getActiveJobs("region"):

        info = XJobMng.getJobInfo(regionJob, TypeOfInfo.BASIC.val(), \
                                 InfoLevel.PROCESS.val(), None, cert)
        jobInfo = JobInfoList(info)
        list.add(jobInfo.getJobResources(regionJob).get(0))

    return list

#
# Get a list of active XtreemOS resource nodes, regardless
# whether they run HBase already or not.
#
def getActiveResources(jsdl):

    # Let them know what we are doing.
    if hbase_verbose >= 2:
        print "Invoking: XResMng.getResources(jsdl,cert,0)"

    if jsdl == None:
        nodes = XResMng.getResources(generateJSDL("dummy", "", "", ""), \
                                 cert, 0)
    else:
        nodes = XResMng.getResources(jsdl, cert, 0)

    # We need at *least* one resource node for anything in hbase-xos.
    if nodes.size() <= 0:
        print "No active resource nodes available"
        sys.exit(1)
    else:
        return nodes

#
# Start Hbase using DIXI.
#
def start():

    # We need the number of region servers to start.
    if hbase_num_regions == 0:
        usage()
        sys.exit(1)

    # Generate the job descriptions for xsub.
    m = generateJSDL(hbase_homedir + "/bin/hbase-xos-master",
                     hbase_userdir + " " + hbase_homedir,
                     hbase_userdir + "/hbase-master.txt",
                     hbase_userdir + "/hbase-master.err")

    s = generateJSDL(hbase_homedir + "/bin/hbase-xos-region",
                     hbase_userdir + " " + hbase_homedir,
                     hbase_userdir + "/hbase-region.txt",
                     hbase_userdir + "/hbase-region.err")

    # Retrieve the list of nodes.
    nodes      = getActiveResources(s)
    regions    = getActiveRegions()
    masters    = getActiveMasters()
    available  = ArrayList()
    numStarted = 0

    # Nodes without region servers are available.
    for node in nodes:
        if not regions.contains(node):
            available.add(node)

    # Verbosely output a list of nodes.
    if hbase_verbose >= 1:
        print 'List of all nodes: '       + str(nodes)
        print 'List of active masters: '  + str(masters)
        print 'List of active regions: '  + str(regions)
        print 'List of available nodes: ' + str(available)

    # Even more verbose: output the JSDL content.
    if hbase_verbose >= 2:
        print 'Master JSDL:'
        print m
        print 'Region JSDL:'
        print s

    # Verify we have enough available nodes.
    if nodes.size() - regions.size() < hbase_num_regions:
        print "Not enough resources (" + \
               str(hbase_num_regions - (nodes.size() - regions.size())) + \
              " more needed)"
        sys.exit(1)

    print "Starting Hbase:",

    # Submit master job.
    if masters.size() == 0:
        performJob(m, available, cert, "master")
        time.sleep(2)

    # Submit region job(s).
    for node in nodes:
        if numStarted < hbase_num_regions:
            available.remove(performJob(s, available, cert, "region"))
            numStarted += 1
        else:
            break

    print "OK"
```

```
#
# Stops running Hbase instances.
#
def stop():
    print "Stopping Hbase:",

    # Fetch list of active servers.
    regions = getActiveJobs("region")
    masters = getActiveJobs("master")

    # Set the number of region servers to stop.
    if hbase_num_regions == 0:
        num_regions = regions.size()
    else:
        num_regions = hbase_num_regions

    # Kill region servers.
    for region in regions:

        # Restrict the number of region servers to kill.
        if num_regions == 0:
            break

        # Send kill signal.
        XJobMng.sendEvent(region, 9, 0, None, cert)

        if hbase_verbose >= 1:
            print region,
        else:
            print ".",

        num_regions -= 1

    # Only kill all masters if we have no regions anymore.
    if getActiveRegions().size() > 0:
        print "OK"
        return

    # Kill all master servers.
    for master in masters:

        if hbase_verbose >= 1:
            print master,
        else:
            print ".",

        # Send kill signal.
        XJobMng.sendEvent(master, 9, 0, None, cert)

    print "OK"
```

```
#
# Stops and then starts Hbase again.
#
def restart():
    stop()
    start()

#
# Prints out the current status of Hbase.
#
def status():

    print "Status Hbase:",

    # Retrieve active hbase nodes.
    masters = getActiveMasters()
    regions = getActiveRegions()

    print str(masters.size()) + " masters",

    # Output list of masters in verbose mode.
    if hbase_verbose >= 1:
        print "(",
        for node in masters:
            print node.host.getCanonicalHostName(),
        print ")",

    print str(regions.size()) + " regions",

    # Output a list of region servers in verbose mode.
    if hbase_verbose >= 1:
        print "(",
        for node in regions:
            print node.host.getCanonicalHostName(),
        print ")"
    else:
        print ""

#
# Initializes the users Hbase directory.
#
def setup():

    print "Initializing HBase:",

    # Generate an JSDL first.
    jsdl = generateJSDL("hbase-xos-setup", \
                hbase_userdir + " " + hbase_homedir,"/tmp/hbase_setup.txt","/tmp/hbase_setup.out")

    # Invoke hbase-xos-setup on the HBase user directory.
    performJob(jsdl, getActiveResources(None), cert, "setup")
    print "OK"
```

```
#
# Program entry point
#
def main(argv):

    global hbase_homedir, hbase_userdir
    global hbase_verbose, hbase_num_regions
    global cert_path,     cert

    # Supported operations using this script.
    actions = {
        "start"   : start,
        "stop"    : stop,
        "restart" : restart,
        "status"  : status,
        "setup"   : setup
    }

    # Attempt to parse command-line arguments.
    try:
        opts, args = getopt.getopt(argv, "hvH:u:r:c:", [
                                   "help",
                                   "verbose",
                                   "hbase-homedir=",
                                   "hbase-userdir=",
                                   "regions=",
                                   "certificate="])
    # Catch errors.
    except getopt.GetoptError:
        usage()
        sys.exit(1)

    # Loop parsed arguments.
    for opt, arg in opts:

        if opt in ("-h", "--help"):
            usage()
            sys.exit(0)

        elif opt in ("-v", "--verbose"):
            hbase_verbose += 1

        elif opt in ("-H", "--hbase-homedir"):
            hbase_homedir = arg

        elif opt in ("-u", "--hbase-userdir"):
            hbase_userdir = arg

        elif opt in ("-r", "--regions"):
            hbase_num_regions = int(arg)

        elif opt in ("-c", "--certificate"):
            cert_path = arg
    #
    # Obtain the user's XtreemOS certificate.
    #
    cert = Utils.readX509Certificate(cert_path, " ")

    #
    # Set the user's HBase home directory, if not explicitely set.
    #
    if hbase_userdir == None:
        hbase_userdir  = "/home/" + UserCertificateUtility.getGlobalUserIdentity(cert)
        hbase_userdir += "/.hbase/"

    # Invoke the appropriate function.
    actions.get("" . join(args), usage)()

    # All done.
    sys.exit(0)

if __name__ == "__main__":
    main(sys.argv[1:])
```

Appendix D

HBase Performance Evaluation Program

```
$ ./bin/hbase org.apache.hadoop.hbase.PerformanceEvaluation
Usage: java org.apache.hadoop.hbase.PerformanceEvaluation [--master=HOST:PORT] \
  [--miniCluster] [--nomapred] [--rows=ROWS] <command> <nclients>

Options:
 master          Specify host and port of HBase cluster master. If not present,
                 address is read from configuration
 miniCluster     Run the test on an HBaseMiniCluster
 nomapred        Run multiple clients using threads (rather than use mapreduce)
 rows            Rows each client runs. Default: One million

Command:
 randomRead      Run random read test
 randomReadMem   Run random read test where table is in memory
 randomWrite     Run random write test
 sequentialRead  Run sequential read test
 sequentialWrite Run sequential write test
 scan            Run scan test

Args:
 nclients        Integer. Required. Total number of clients (and HRegionServers)
                 running: 1 <= value <= 500
Examples:
 To run a single evaluation client:
 $ bin/hbase org.apache.hadoop.hbase.PerformanceEvaluation sequentialWrite 1
```

Bibliography

[1] Fay Chang, Jeffery Dean, Sanjay Ghemwat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *Seventh Symposium on Operating System Design and Implementation*, November 2006.

[2] John M. Willis. Cloud vendors a to z (revised). `http://www.johnmwillis.com/cloud-computing/cloud-vendors-a-to-z-revised/`, May 2009.

[3] Richard Martin and J. Nicholas Hoover. Guide to cloud computing. `http://www.informationweek.com/news/services/hosted_apps/showArticle.jhtml?articleID=208700713\&pgno=1\&queryText=\&isPrev=`, May 2009.

[4] XTreemOS.eu. Xtreemos. `http://www.xtreemos.eu/`, October 2008.

[5] Linus Torvalds. Linux operating system kernel. `http://www.kernel.org`, February 2009.

[6] Massimo Coppola, Yvon Jgou, Brian Matthews, Christine Morin, Luis Pablo Prieto, Óscar David Sánchez, Erica Y Yang, and Haiyan Yu. Virtual organization support within a grid-wide operating system. *IEEE Internet Computing*, 12(2), 2008.

[7] XTreemfs.org. Xtreemfs. `http://www.xtreemfs.org/`, October 2008.

[8] Richard Jones. Anti-rdbms: A list of distributed key-value stores. `http://www.metabrew.com/article/anti-rdbms-a-list-of-distributed-key-value-stores/`, May 2009.

[9] Matthias Nicola and Matthias Jarke. Performance modeling of distributed and replicated databases. *IEEE Transactions on Knowledge and Data Engineering*, 12(4), July/August 2000.

[10] Google Research. Google bigtable. `http://labs.google.com/papers/bigtable.html`, October 2008.

[11] Amazon.com. Amazon simpledb. `http://aws.amazon.com/simpledb/`, October 2008.

[12] Jonathan Gray. Hadoop and hbase vs rdbms. `http://www.docstoc.com/docs/2996433/Hadoop-and-HBase-vs-RDBMS`, May 2009.

[13] apache.org. Apache hbase. `http://hadoop.apache.org/hbase/`, October 2008.

[14] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and Franois Yergeau. Extensible markup language (xml) 1.0 $fifthedition$. `http://www.w3.org/TR/2008/REC-xml-20081126/`, November 2008.

[15] XtreemOS project. Xtreemos bugtracker. `http://gforge.inria.fr/tracker/?group_id=411`, May 2009.

[16] Vrije Universiteit Amsterdam. Vrije universiteit amsterdam. May 2009.

[17] Christian Plattner and Gustavo Alonso. Ganymed: scalable replication for transactional web applications. In *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 155–174, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[18] www.oracle.org. Oracle real applications clusters data sheet. Technical report, Oracle Enterprise, May 2009.

[19] Andrew S. Tanenbaum and Maarten van Steen. Distributed systems principles and paradigms. pages 17–18. Pearson, 2007.

[20] Ian Foster. What is the grid: A three checkpoint list. July 2002.

[21] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. 2001.

[22] The Open Group. Ieee std 1003.1: Posix.1-2008. `http://www.opengroup.org/onlinepubs/9699919799/`, December 2008.

[23] Tom Goodale, Shantenu Jha, Hartmut Kaiser, Thile Kielmann, Pascal Kleijer, Andre Merzky, John Shalf, and Christopher Smith. A simple api for grid applications (saga). Number 090. January 2008.

[24] Pascal Le Metayer. Design and implementation of basic checkpoint/restart mechanisms in linux (d2.1.3). December 2007.

[25] Ali Anjomshoaa, Fred Brisard, Michel Drescher, Donal Fellows, An Ly, Stephen McGough, Darren Pulsipher, and Andreas Savva. Job description submission language 1.0. Number 056. Global Grid Forum, November 2005.

[26] R. Housley, W. Ford, W. Polk, and D. Solo. Internet x.509 public key infrastructure certificate and crl profile. Internet Society, January 1999.

[27] Mike Mesnier, Gregory R. Ganger, and Erik Riedel. Object based storage. *IEEE Communications Magazine 41*, 41(8), August 2008.

[28] Douglas Crockford. The application/json media type for javascript object notation (json). The Internet Society, July 2006.

[29] FUSE development team. Filesystem in userspace. `http://fuse.sourceforge.net/`, February 2009.

[30] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, New York, NY, USA, 2003. ACM Press.

[31] apache.org. Apache hadoop. `http://hadoop.apache.org/`, October 2008.

[32] Jeffery Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI '04*, December 2004.

[33] Jim Kellerman. Hbase: Structured storage of sparse data for hadoop, May 2009.

[34] Mark Slee, Aditya Agarwal, and Marc Kwiatkowski. Thrift: Scalable cross-language services implementation. Facebook, april 2007.

[35] A.Th. van Deursen. *Een hoeksteen in het verzuild bestel.* Uitgeverij Bert Bakker, Amsterdam, 2005.

[36] Vrije Universiteit Amsterdam. Vrije universiteit amsterdam: About the griffin. `http://www.vu.nl/en/about-vu-amsterdam/mission-and-profile/the-griffin/index.asp`, may 2009.

[37] The Apache Software Foundation. Hbase 0.19.2 overview. `http://hadoop.apache.org/hbase/docs/current/api/overview-summary.html`, May 2009.

[38] Michael Stack. Testing hbase performance and scalability. `http://wiki.apache.org/hadoop/Hbase/PerformanceEvaluation`, May 2009.

[39] The Python Project. Python v2.6.2 documentation. `http://docs.python.org/index.html`, May 2009.

[40] Mark Lutz. *Learning Python*. O'Reilly Media Inc., Sebastopol, CA, 2008.

[41] Óscar David Sánchez. Xtreemos user and administrators guide. Technical report, XtreemOS Project, December 2008.