Project no. IST-033576

XtreemOS

Integrated Project
BUILDING AND PROMOTING A LINUX-BASED OPERATING SYSTEM TO SUPPORT VIRTUAL ORGANIZATIONS FOR NEXT GENERATION GRIDS

Evaluation Report
D4.2.6

Due date of deliverable: November 30th, 2009
Actual submission date: December 23rd, 2009

Start date of project: June 1st 2006

Type: Deliverable
WP number: WP2.2, WP2.3, WP3.1, WP3.2, WP3.3, WP3.4, WP3.5, WP3.6, WP4.2
Task number: T2.2.13, T2.3.8, T3.1.5, T3.2.7, T3.3.14, T3.4.10, T3.5.7, T3.6.7, T4.2.4, T4.2.5

Responsible institution: SAP
Editor & and editor's address: Bernd Scheuermann
SAP Research, CEC Karlsruhe
Vincenz-Prießnitz-Str. 1
76131 Karlsruhe, Germany

Version 3.0 / Last edited by Bernd Scheuermann / December 23rd, 2009

Project co-funded by the European Commission within the Sixth Framework Programme		
Dissemination Level		
PU	Public	√
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Revision history:

Version	Date	Authors	Institution	Section affected, comments
1.0	28/04/09	Bernd Scheuermann	SAP	Initial version
1.1	31/05/09	WP4.2	all teams	Test plans included
1.2	30/06/09	Maik Jorra, Barry McLarnon, Samuel Kortas, Roman Talyansky, Enric Tejedor	BSC, EDF, SAP, ZIB	Application descriptions added
1.5	04/12/09	Bernd Scheuermann	SAP	Evaluation of Installation and Configuration added
1.8	10/12/09	WP4.2	all teams	test results from WP4.2 consolidated
1.9	10/12/09	Alvaro Arenas, Matej Artac, Eugenio Cesario, Mathijs den Burger, Aleš Černivec, Jörg Domaschka, Noé Gallego, Ian Johnson, Thilo Kielmann, Mathijs den Burger, Ramon Nou, Marko Novak, Marko Obrovac, Guillaume Pierre, Alvaro Reol, Santiago Prietro	BSC, CNR, INRIA, STFC, TID, ULM, VUA, XLAB	test results from SP2 and SP3 consolidated
2.0	11/12/09	Bernd Scheuermann	SAP	submitted to internal review
2.1	18/12/09	Bernd Scheuermann	SAP	revised according to internal reviewers' comments
2.2	21/12/09	Samuel Kortas	EDF	Further test results included
2.3	21/12/09	Marjan Sterk, Bojan Blazica	XLAB	Further test results included
3.0	23/12/09	Bernd Scheuermann, Roman Talyansky	SAP	final consolidation and polishing

Reviewers:
Franz Hauck (ULM), Ian Johnson (STFC), Yvon Jégou (INRIA), Peter Linnell (INRIA)

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
T2.2.13	Performance evaluation	KER*, INRIA, XLAB, UDUS.
T2.3.8	Integration testing	TID*, INRIA
T3.1.5	Performance evaluation	VUA*
T3.2.7	Performance evaluation	ULM, VUA*
T3.3.14	Performance evaluation	BSC*, XLAB, INRIA, UDUS
T3.4.10	XtreemFS Testing, Performance, Compatibility and Maintenance	BSC,CNR*,ZIB
T3.5.7	Integration	STFC*, XLAB, INRIA, ICT, SAP, ULM
T3.6.7	Integration and testing	TID*, BSC
T4.2.4	Implementing and porting applications to XtreemOS	BSC, EADS, EDF, SAP*, TID, UDUS, VUA, XLAB, ZIB
T4.2.5	XtreemOS experiments and evaluation	BSC, EADS, EDF, SAP*, TID, UDUS, VUA, XLAB, ZIB

°This task list may not be equivalent to the list of partners contributing as authors to the deliverable
*Task leader

Executive Summary

The experimental evaluation of XtreemOS is an essential supporting activity for the project as it identifies critical bugs and gives feedback on performance, scalability, stability, correctness and usability of the system. This feedback provides guidance to developers, distribution packaging and also to project management. For the first time, an evaluation report from WP4.2 is used to collect and present experimentation results from the consortium in a common document. Contributors are WP4.2 as well as the development work packages from SP2 and SP3. The emphasis of the evaluation by WP4.2 is on testing from the end-user perspective using applications from the scientific and business domain. The tests carried out by SP2 and SP3 concentrate on lower-level performance benchmarking.

The evaluation is sub-divided into four different test categories: evaluation of installation and configuration, evaluation of XtreemOS components, comparison with other Grid solutions and usability evaluation of the XtreemOS MD flavor. The evaluation of installation and configuration includes a long term survey carried out by WP4.2. The purpose is to record and track end-user experience with multiple releases of XtreemOS starting with XtreemOS 1.0, three intermediate internal releases and finally XtreemOS 2.0. The second category covers the evaluation of a wide range of XtreemOS components including node-level VO support, checkpointing and restart, LinuxSSI, DIXI message bus, XtreemOS API, Distributed Servers, Virtual Nodes, application execution management, data management, security services and the mobile device flavor. Experiments for this category were performed by SP2, SP3 and WP4.2. Furthermore, a theoretical comparison of XtreemOS with other Grid solutions is complemented by a description of experimental comparisons between XtreemOS and Globus. Finally, WP4.2 evaluated the MD flavor of XtreemOS in terms of assessing the usability of the MD installer and of the JobMA application.

All evaluations are documented giving detailed specifications which shall account for comprehensive and reproducible test setups. Test results are analyzed and corresponding feedback is provided to XtreemOS developers. The deliverable is concluded with a summary of all test results, an assessment of the fulfillment status of the requirements and an outlook on forthcoming evaluation activities.

Contents

Chapter 1

Introduction

WP4.2 provides a range of applications from the scientific and business domain. Using these applications, experimental tests are carried out to evaluate XtreemOS features from the end-user perspective and to provide feedback to developers, packaging and project management. This deliverable presents the specifications and the results of the experimental evaluation which is structured into four different categories:

1. evaluation of installation and configuration,
2. evaluation of XtreemOS components,
3. comparison with other Grid solutions and
4. usability evaluation of the XtreemOS mobile device (MD) flavor.

The evaluation of installation and configuration includes a long term survey conducted among XtreemOS end-users starting with XtreemOS 1.0 (the first public release), continuing with three intermediate internal releases and finally the second public release XtreemOS 2.0 (which is the latest version at the time of writing this deliverable). The survey aims at recording the degree of end-user satisfaction with the install CDs of the various XtreemOS releases and to monitor the change of results as the software evolves during project execution. This allows us to track user experiences with the install CDs and also to collect comments and recommendations for improvements. Results of the survey have been reported to project management and packaging providers in order to support development and project planning.

The second category comprises the in-depth evaluation of XtreemOS components including node-level VO support, checkpointing and restart, DIXI message bus, XtreemOS API, Distributed Servers, Virtual Nodes, application execution management, data management, security services and the mobile device flavor. Experiments conducted in this category put emphasis on evaluating the performance, scalability, stability and correctness of the respective XtreemOS developments. Apart from WP4.2, each work package in SP2 and SP3 contributed to the planning, specification, execution and documentation of the experiments. For this purpose, each development work package introduced a dedicated task (cf. DoW) and devoted manpower to organize the performance evaluation. The respective contributions from the various work packages and partners are marked to clearly indicate the responsibilities for each test unit. Generally, WP4.2 focuses on the application-centric evaluation from the end-user's view whereas SP2 and SP3 put emphasis on lower-level performance benchmarking.

Another category deals with the comparison of XtreemOS with alternate Grid solutions. This comparison includes a theoretical comparison and experiments with the Globus toolkit as a well-established representative of Grid middleware tools. A further application-centric comparison of both Grid approaches is being executed at the time of writing this deliverable which gives an outline of these tests.

The final test category examines the usability of XtreemOS MD. The focus is on the XtreemOS-MD installer and the mobile application JobMA which provide many graphical user elements and are therefore considered as the best choice for the first sequence of XtreemOS usability evaluation. Usability evaluation starts with the analysis of the MD flavor as users expect intuitive usability for applications running on mobile devices providing only small screens and limited input capabilities. The experiments examine the ease of installation of XtreemOS-MD using the installer provided and the ease of use of the JobMA application. The methodology followed is based on a heuristic evaluation, which is a method to evaluate the possible usability problems of a user interface.

The remainder of this deliverable is structured as follows: Chapter 2 gives an overview and description of the new or changed applications contributing to WP4.2. The setup of the long term survey and the results of the evaluation of XtreemOS installation and configuration from install CD are provided in Chapter 3. Chapter 4 describes the test plans, specifications and results of the evaluation of the various XtreemOS components. The comparison with other Grid solutions is outlined in Chapter 5. The last evaluation category is addressed by Chapter 6 which presents the usability analysis of XtreemOS MD. Finally, the results of the evaluations are summarized in Chapter 7, and an outlook on future work concludes this deliverable.

Chapter 2

Application Descriptions

This chapter gives an updated overview of WP4.2 applications and further introduces the new applications added to the portfolio. These new applications will also be used for the evaluation of XtreemOS.

2.1 Overview

Table 2.1 provides an overview of the current set of applications in WP4.2.

Compared to the set of applications used for deliverable D4.2.5 [9], the following new applications have been added:

- Hmmpfam on COMP Superscalar (BSC), replacing GRID superscalar fastDNAml
- openTurns (EDF) replacing Simeon
- Trex, MaxDB and RBSM (SAP) replacing SAP Netweaver Application Server
- Cloud Computing (VUA/ZIB) as new application scenario

Another application, DBE from T6, was removed from the set of applications due to the withdrawal of the partner from the Consortium. During the execution of the experimental evaluation, the above-mentioned applications were identified as more suitable to exploit XtreemOS and to demonstrate the strengths of the system. In particular, the spectrum of applications has been widened and therefore provides an improved coverage of XtreemOS capabilities. Furthermore, the Cloud Computing scenario (currently being implemented) has been added to demonstrate the benefit of using XtreemOS in a cloud computing landscape.

The subsequent section provides descriptions of the new applications added to the WP4.2 portfolio.

Table 2.1: Applications in WP4.2.

Partner	**Application Name**	**Short Name**	**Application Area**
BSC	COMP Superscalar	COMPSS	Bio-informatics
BSC	SpecWeb	SPECWEB	Enterprise solutions
EADS	Elfipole	ELFIPOLE	Electromagnetics
EADS	jCAE	JCAE	Computer aided engineering
EDF	Moderato/Maestro	MODERATO	Particle physics
EDF	OpenTURNS	OPENTURNS	Sensitivity and reliability analysis
EDF	Zephyr	ZEPHYR	Fluid mechanics
EDF	Secured Remote Computing	SRC	Enterprise solutions
SAP	SAP NetWeaver Search and Classification	TREX	Enterprise solutions
SAP	SAP MaxDB replayer	MaxDB	Enterprise solutions
SAP	Rule-based System Management	RBSM	Enterprise solutions
TID	TID Instant Messaging Application	IMA	Instant messaging
TID	Job Management Application	JOBMA	XtreemOS job management
UDUS	Wissenheim	WISS	Virtual Presence
VUA/ZIB	Cloud Computing	CLOUD	Image Archive
XLAB	Galeb	GALEB	Economics, optimization

2.2 Hmmpfam on COMP Superscalar

BSC will contribute to XtreemOS with a COMP Superscalar enabled version of hmmpfam. hmmpfam is a bioinformatics application that compares sequences of amino acids against a database of Hidden Markov Models, which represent protein families, searching for significantly similar sequence matches with each model. The analysis performed by hmmpfam is computationally intensive and embarrassingly parallel, which makes it a good candidate to benefit from COMP Superscalar.

COMP Superscalar (COMPSs) is a framework that facilitates the development and execution of Java Grid-unaware applications. In the COMPSs programming model the user selects some methods of a sequential Java application, which should be run on the Grid. At execution time, COMPSs will be in charge of automatically replacing the invocations of these methods by the creation of remote tasks. Moreover, the COMPSs runtime will schedule and control the execution of these tasks on the available Grid resources, controlling the data dependencies between them.

COMPSs has to take into account the different file systems, resource management and scheduling, etc. The features offered by XtreemOS release this system from many of these duties, making it more efficient and portable. More precisely, COMPSs tests the following XtreemOS features: resource discovery and reserva-

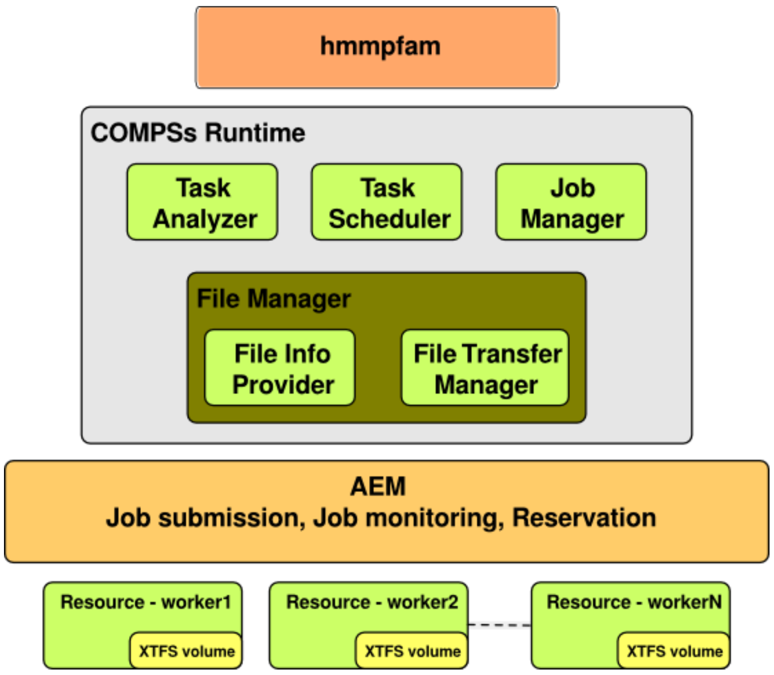


Figure 2.1: The hmmpfam application sits on top of COMPSs, which is in charge of parallelizing its execution. The COMPSs has been ported to XtreemOS (AEM API), which provides it with access to resource and job management capabilities, for the application to be run on the available resources.

tion, job management and scheduling, file system.

COMPSs-hmmpfam provides a good use case for XtreemOS since it demonstrates the parallelization of a scientific bioinformatics application using XtreemOS. In addition, we should note that the porting of the COMP Superscalar environment to XtreemOS allows not only hmmpfam, but also any other application intended to use the Grid transparently, to run on top of XtreemOS.

2.3 openTurns

Since the beginning of 2005, a partnership of three companies has been working together on building a tool designed to perform uncertainty treatment and reliability analysis in a structured industrial approach. This tool is "**OpenTURNS**" standing for an **Open** source initiative to **T**reat **U**ncertainties, **R**isks'**N** **S**tatistics.

Running an OpenTURNS study often consists of running thousands or millions of independent calculations from which a global, consolidated answer is gathered.

The typical use case consists of launching these calculations on the grid formed by all connected XtreemOS nodes belonging to the same dedicated virtual organisation (VO).

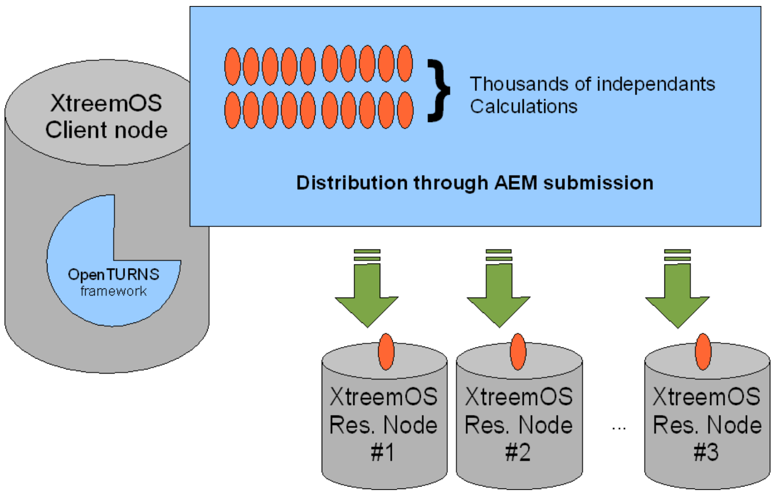


Figure 2.2: Through the XtreemOS scheduler, the OpenTURNS platform will use all the available resources to run a high number of independant calculations

In 2009, OpenTURNS will be able to sequentially use the resource of only a single node that it has been installed on. Then by June 2010, it will take advantage of the whole grid of computers sharing the same dedicated VO.

Through XtreemOS Job Manager, OpenTURNS will be able to submit jobs transparently and in parallel, reaching a sustained performance with minimum modification to the OpenTURNS framework.

2.4 SAP NetWeaver Search and Classification (TREX)

SAP BI runs a distributed middleware system over SAP TREX. Figure 2.3 shows the architecture of TREX. In the document mode, TREX indexes a large collection of documents. At the indexing stage, one TREX node reads the documents and indexes them, resulting in a collection of index files that are written onto the distributed file system imposing write load. At this stage, placing source files on the XtreemFS file system enables high read throughput of reading the source files, due to the striping capabilities of XtreemFS. At the write phase of the index creation stage, the index is written back to XtreemFS, exploiting high write throughput of XtreemFS, once again due to XtreemFS striping. To summarise, striping capabilities of XtreemFS open the opportunity of scaling out, as opposed to scaling up with most of the commercial filer solutions.

At the search stage, search queries are issued in parallel at all TREX nodes. To answer a search query, a TREX node reads relevant portions of the index, imposing read load on the distributed file system.

TREX uses the XtreemFS component of XtreemOS to distribute search indices among its landscape nodes. When a stream of search queries is applied to the TREX search engine, the striping mechanism of XtreemFS widens the overall throughput of answering queries due to parallel reading of striped index data. In the future, when replication is implemented within XtreemFS, it will make index data highly available and further widen throughput of answering search queries thanks to the mechanism of placing replica close to its reader. Overall, XtreemFS enables to reduce TCO, providing high QoS storage backend for TREX as an alternative to expensive filer technologies used currently.

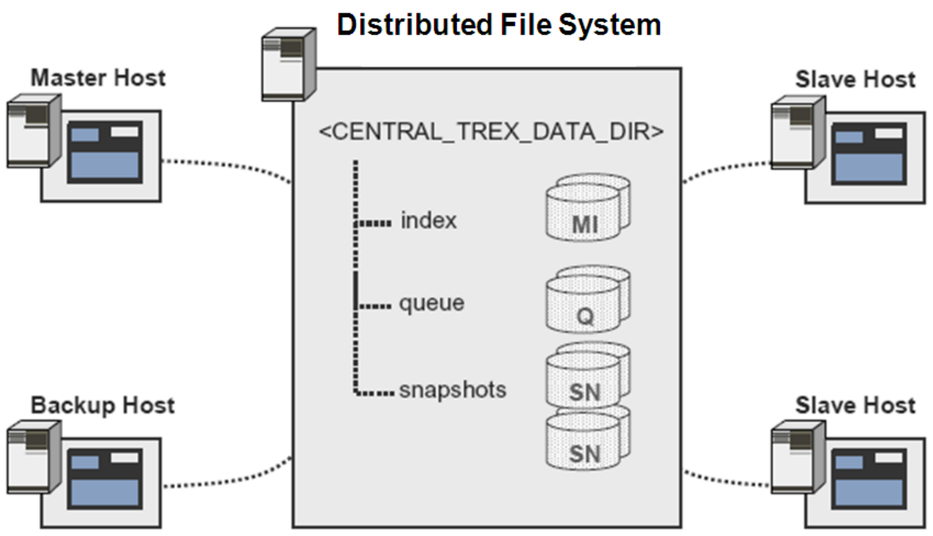


Figure 2.3: TREX - Enterprise Search Engine

2.5 SAP MaxDB Replayer

SAP Web Application Server runs over SAP MaxDB database. Thus the IO accesses generated by MaxDB represent transactional IO load that is applied to filesystem. SAP Web Application Server is a very complex system and it is hard to install, maintain and use it directly in performance testing. To avoid those difficulties, MaxDB symptoms are rather recorded in a trace file at the recording stage, using our Tracy tool, as depicted in Figure 2.4. The symptoms are the actual access sequences of MAXDB to the underlying file system. The recorder captures details of each access as well as the identity of the process and thread that issued it.

At the replaying stage the trace file is replayed over the distributed file system using our Tracy tool, thus porting the actual MaxDB load to the tested file system. To create parallel load on the file system in the distributed landscape of multi-node cluster, several trace files are replayed in parallel (one trace file on each node of the distributed system).

MaxDB uses the XtreemFS component of XtreemOS to achieve high overall throughput of handling DB accesses due to the striping mechanism of XtreemFS. Cross WAN data access of XtreemFS is used for disaster recovery purpose. XtreemFS is also used to provide low-latency access to data across LANs and WANs. Striping capabilities of XtreemFS enable us to achieve low response times in case of concurrent accesses to the file system. In the future, when replication of XtreemFS is implemented, it will be used for high availability of MaxDB data. Overall, XtreemFS enables to reduce TCO, providing high QoS storage back-end for MaxDB as an alternative to expensive filer technologies used currently.

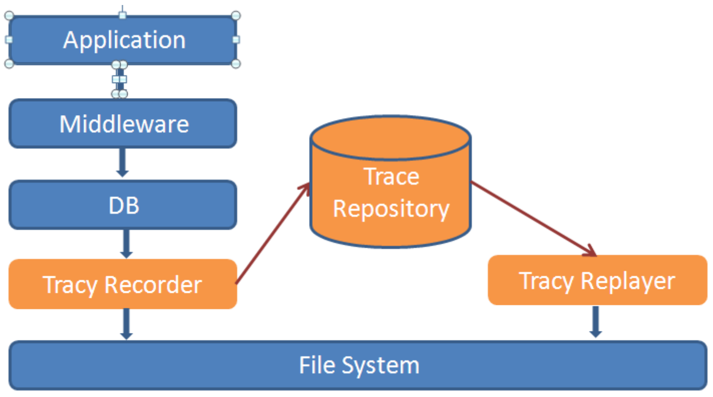


Figure 2.4: Enterprise Scenario - MaxDB Replay

2.6 Rule-based System Management

Rule-Based System Management (RBSM) is a graphical system management tool which aims to reduce the amount of administrative overhead required in managing large scale networks. This is achieved by providing a management framework and encoding frequently repeated tasks as a set of rules. Different decision models can be used to determine the actions chosen by the rules for a particular need; for example, efficiency, reliability, scalability or performance. RBSM has been designed as a framework, allowing administrators to tailor the system to their needs and set the level of automation that is required. This piecemeal automation allows for changing rule sets depending on the decision model, adding new rules and control scripts, and changing the data gathered by the landscape generator, as well as its frequency of updates and length of retention.

The main elements of the program (as shown in Fig. 2.5) are:

- **Landscape Generator** - Generates a model of the System Landscape under Control, consisting of a monitoring interface and a Landscape Manager.

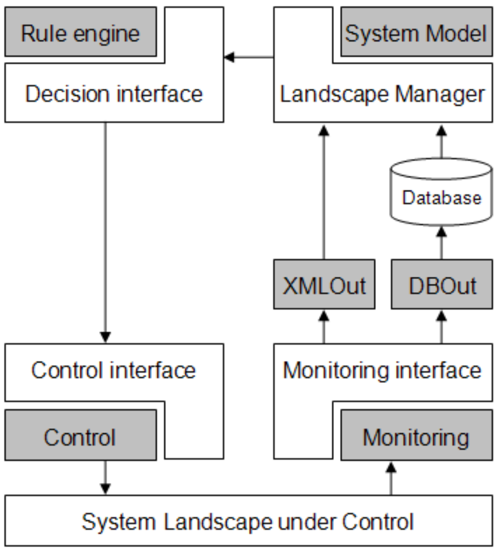


Figure 2.5: Simplified diagram showing main components of the RBSM system

- **Rule Engine** - Uses rule-based logic to automatically perform routine actions.

- **Control Manager** - Performs the actions on the system landscape as decided by the administrator or rules, implemented through a control interface.

Landscape Generator

The Landscape Generator is an information aggregation component used to populate the system landscape. The generator gathers data from each node in the landscape (e.g. CPU load, free memory, users and installed software) and returns it to the main RBSM component, where it is used to create a live model in Java. This Java model forms a hierarchy that can then be viewed by the administrator, or used by the rulebase, which analyses the landscape state to generate control actions, such as deploy software, or migrate virtual machines. This model remains constantly updated with the information changes from the landscape.

Rule Base

The Rule Base provides a set of rules that are used to decide what actions are required to be performed on the landscape depending on the decision model in use. The rules are used to automate decisions that are performed repeatedly or regularly to reduce the amount of effort required from the administrator. The different areas covered by rules include *installation*, *configuration*, *start up* and *shut down* of components, as well as *replication* and *migration* of software, services, data and virtual machines.

For example, an administrator may wish to install a piece of software to a single machine (physical or virtual) within a domain. RBSM can be used to decide which of these machines matches the criteria for installing the software based on a number of dynamic attributes, such as pre-installed dependencies, hardware requirements, machines with low CPU load, and so on.

Control Manager

The control engine component is responsible for performing actions on the system landscape as decided by the rules engine. It connects to a daemon program running on each machine in the landscape that uses a repository of scripts to manage the different elements of the landscape. Actions can include installing software, adding users, changing access rights, shutting down redundant machines and much more. This can be added to as more administrative patterns and repetitions are identified.

RBSM in XtreemOS

Benefits to XtreemOS

RBSM provides a management utility that enables large-scale distributed systems to be controlled more efficiently. Given the intention to scale to a large number of nodes, it is hoped that this will allow management of an XtreemOS deployment to be more streamlined, and allow new features to be tested and integrated as they are needed in the management system.

RBSM will also provide an evaluation of a number of features already provided by XtreemOS, such as Checkpointing, Migration and Single Sign On. The RBSM system will also provide a demonstration of these features and their value for Systems Management, Dependability and Cost Efficiency.

Benefits to RBSM

By developing RBSM for the XtreemOS platform it can use the functionality provided for Checkpointing, Migration and Single Sign On without requiring additional implementation. These features of the XtreemOS platform also means writing scripts for distributed deployment is simpler, and scalable storage is beneficial for storing the system landscape data for a large network.

Using the already existing functionality provided by XtreemOS, management within RBSM is made simpler, more secure, and more reliable, providing a robust and useful manageability interface.

Status

This application is currently under development, with a prototype administration interface already available and a subset of the control actions implemented. It is currently targeted to be shown in a demonstration for WP4.4.

2.7 Cloud Computing

XtreemOS will be utilized as the underlaying platform for Zmile. Zmile will be an on-line photo archive, where users will be able to upload and categorize there photos, as well as browsing and searching images which are available to the public.

Being a photo archive, Zmile has two major needs. On one hand it requires a huge amount of stable and scalable storage to store the images. And on the other hand it requires computational resources for tasks like the scaling or rotating of photos which a user uploads.

The outlined needs can be satisfied by services offered by XtreemOS. The first demand is addressed by the file-system of XtreemOS: XtreemFS. It can be accessed from the Zmile-server like a local file-system using the XOSAGA adapter. The gained advantage is that the server doesn't have to care where to put a file to reach the best fill rate and thanks to XtreemFS reliability it even doesn't have to care about backups. Thanks to the fact that the stored images are read-only (except for their deletion) Zmile can uses the full advantage of XtreemFS's *read-only replication*.

The second demand is satisfied by the job management system of XtreemOS: AEM. Again XOSAGA will be used on the server-side to send jobs like an ImageMagick command to shrink a photo to AEM. And again Zmile doesn't need to worry about the load-balancing.

An additional benefit is that Scalaris, which is also used in XtreemOS's SRDS, can be utilized to store image meta-data like EXIF information.

To this point, these are just the advantages from the usage of XtreemOS as a platform for a scalable memory and storage consuming Web-service. But this can even be driven further by lifting the whole application into a cloud. In this case, Amazon's EC2 [26] is the target cloud for the application. In the Amazon cloud it is possible to prepare images of XtreemOS with an installed an ready to use Zmile-server and store them into S3 storage [25].

Whenever the Zmile-server detects a shortcoming of one of its resources, may it be storage or compute power, it deploys a new application-instance from a stored image and thus compensates the bottleneck. As mentioned above, the Zmile-server doesn't need to take care about distributing its state or memory to the new instance. The scalability of XtreemOS, respect to its services, first and foremost XtreemFS, ensures that the new instance will be seamlessly integrated. This is shown in figure 2.6 where all the application traffic passes *Amazons Elastic Load Balancing*, which can be configured to detect overloaded nodes. If all nodes have too much load, a new node will be spawned, having access to the data of the other Zmile-Servers.

One can easily imagine that as long as the Zmile-Server can handle the load of incoming requests and only requires more storage, it will be sufficient spawning a plain XtreemOS image. Plain means here that there is no need for this images to have the Zmile-application installed.

This scenario is shown in figure 2.7 where the Zmile-server acts as a gateway to the cloud. When the user uploads a picture, it is dispatched through the XOSAGA

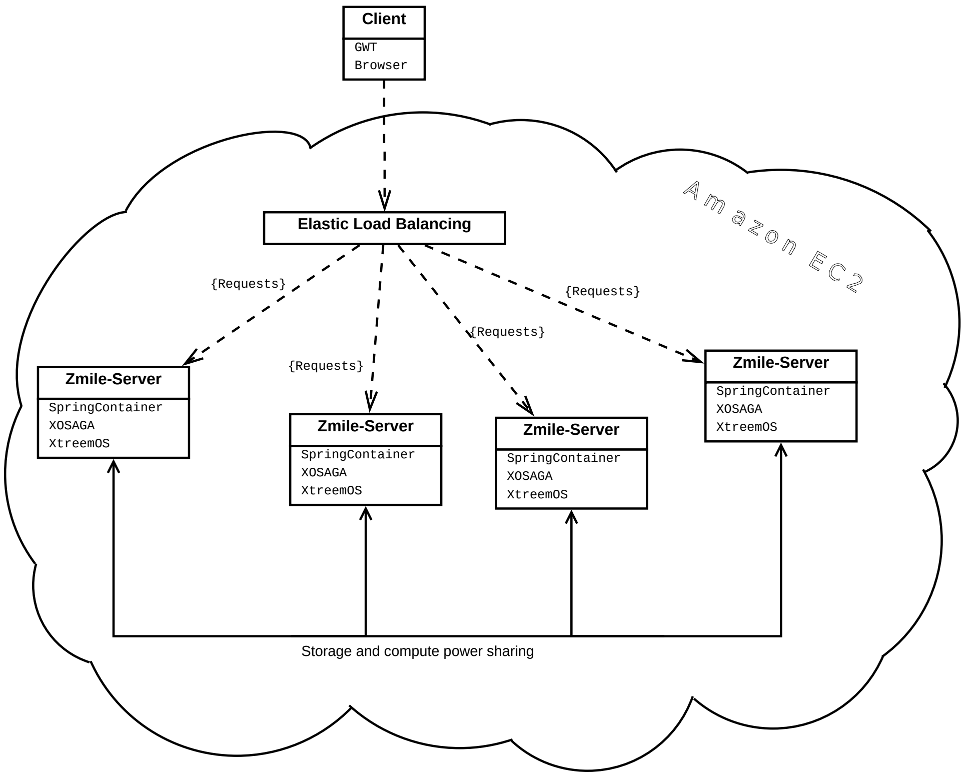


Figure 2.6: Zmile with replicated servers.

adapter an stored on one of the XtreemOS nodes marked as storage. If the picture should be preprocessed this is done by the XtreemOS-computing nodes. Of course a mixture of both scenarios is possible.

As closing note, it should be mentioned that for the development, only the Java programming language together with standard Java web-frameworks like Spring and GWT will be used. The whole integration with XtreemOS will be performed over the Java implementation of the XOSAGA adapter. So from the point of view of an application developer, there is hardly any overhead in utilizing XtreemOS but a big gain of scalability and stability. The developer doesn't need to worry about these aspects and thus can fully focus on implementing the application logic and GUI itself.

Given the early state of the Zmile development, no tests have been performed yet. It is planned to release the first public screen-shots in January 2010. The deployment of a public version is currently scheduled for late spring of 2010. This deployment will consist of only a single web-server. Until late summer of 2010 it planned to have the final version of the application running on Amazons EC2.

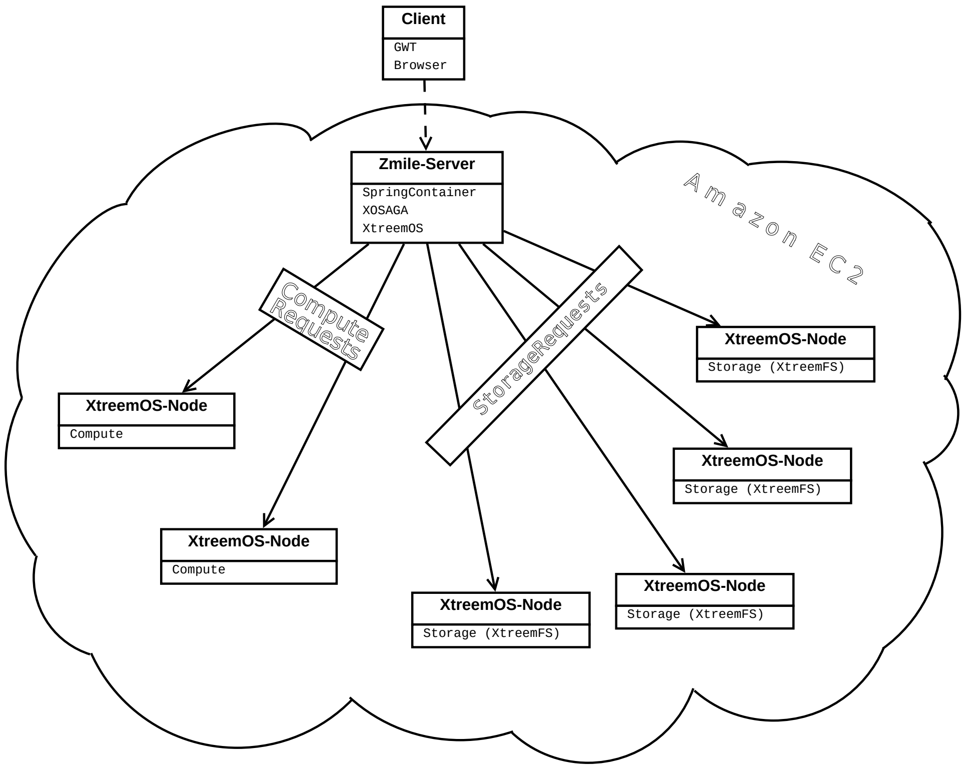


Figure 2.7: Zmile-server with some XtreemOS instances.

Chapter 3

Evaluation of Installation and Configuration

WP4.2 carried out a long-term survey to record the degree of end-user satisfaction with various XtreemOS releases and to monitor the change of results as the software evolves during project execution. Starting with the first public release until the latest release when writing this deliverable (public release XtreemOS 2.0), user experiences with the install CDs have been collected along with comments and recommendations for improvements. Results of the survey have been reported to project management in order to support development and project planning. Subsequently, the setup of the survey is explained followed by a presentation of the results and their analysis.

3.1 Survey Setup

The survey was conducted in the form of an online questionnaire collecting feedback from XtreemOS end users regarding their satisfaction and experience with the XtreemOS install CDs and the accompanying documentation. For each question, user satisfaction was rated on a scale from 1 (lowest satisfaction) to 6 (highest satisfaction). Experience could be expressed by additional fields where participants could enter plain text describing e.g. problems and further recommendations.

The questionnaire is sub-divided into four different categories:

Installation: This category comprises the activities related to the acquirement of the install media and the subsequent installation process. The question items are:

- Ease of getting the installation media (CDs/ISOs).
- Ease of installation.
- Ease of installing additional packages after install.
- Speed of installation.

Configuration: The activities for this category comprise the configuration and customization procedure of the installed XtreemOS. The question items correspond to the different types of XtreemOS nodes:

- Ease of configuring core nodes.
- Ease of configuring resource nodes.
- Ease of configuring client nodes.

Basic usage: After successful installation and configuration follow basic usage activities with the operating system. The respective question items include:

- Stability of the software.
- Ease of managing users.
- Ease of managing VOs.

Documentation: The install CDs are accompanied by user guides. The assessed properties include:

- Clarity of the documentation.
- Completeness of the documentation.
- Correctness of the documentation.

Test items are different versions of the install CD-ISOs available from the XtreemOS homepage:

- v1.0: The first public release of XtreemOS.
- v1.1 RC5: An internal release candidate with facilitated installation and configuration procedure.
- v2.0 beta 1: First internal beta release of XtreemOS 2.0
- v2.0 beta 2: Second internal beta release of XtreemOS 2.0
- v2.0 public: The second public release of XtreemOS.

3.2 Survey Results

A total of 17 end-users from the XtreemOS consortium participated in the survey. Participants were required to download the install media and to install and configure a small grid of 3 nodes (1 core node, 1 resource node and 1 client node) on inhouse testbeds. They were asked to follow the instructions given in the documentation and also to keep records of the progress and incidents discovered (feedback to developers and packaging providers was given via bug trackers).

3.2.1 Overview

Figure 3.1 shows the mean satisfaction with the different test items (XtreemOS releases) averaged across all survey categories. It can be observed that the results increase monotonously from a level of 2.78 for XtreemOS 1.0 to 3.92 for XtreemOS 2.0 public release. This increase can mainly be attributed to facilitated installation and configuration tools, automatized setup, enhanced user interfaces (more graphical user interfaces) and advancement in the documentation.

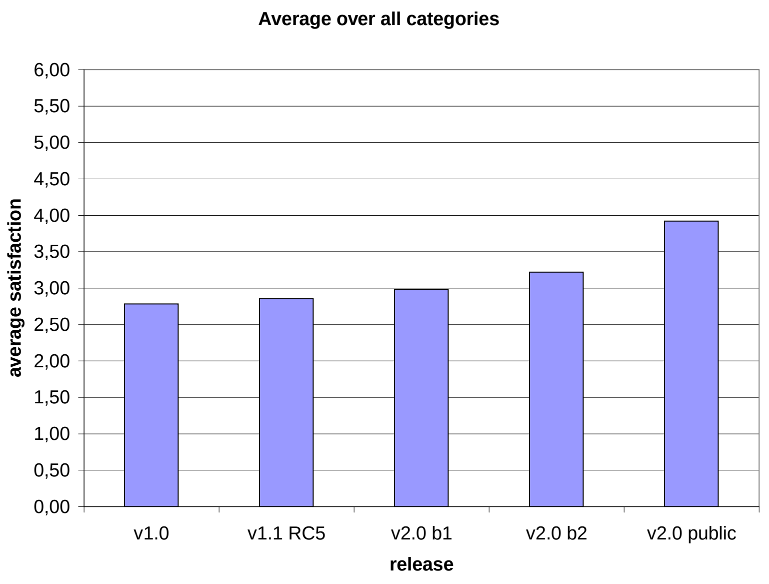


Figure 3.1: Overview of user satisfaction for all test items averaged across all categories.

Averaging the satisfaction across all test items, Figure 3.2 presents the results for the different survey categories (installation, configuration, basic usage and documentation). Installation receives the highest rating of 4.06 as people encountered least problems with the download and matured installation tools provided for the underlying Linux distribution. The subsequent configuration procedure, however, was rated with the worst average value of 2.63, which can be accounted to many bugs and manual work-arounds needed in particular for the early releases 1.x. Basic usage and documentation are rated on the second and third rank, respectively. Details and results are discussed and broken down later in sections 3.2.4 and 3.2.5.

Figure 3.3 gives an overview of the average satisfaction level for the different categories and test items. Starting from the first public release all categories show clearly increasing trends. Especially the latest version made a noticeable leap in quality in terms of stability, ease of usage and also supported by a revised documentation.

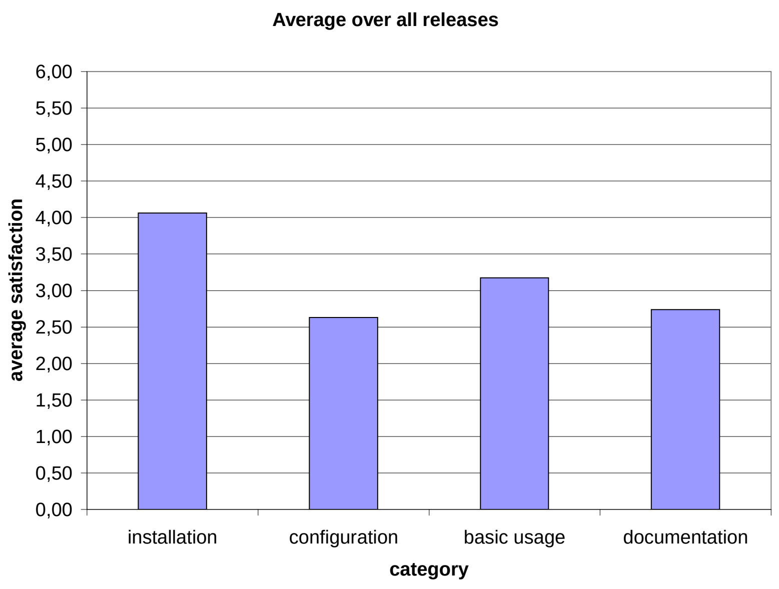


Figure 3.2: Overview of user satisfaction for all categories averaged across all test items.

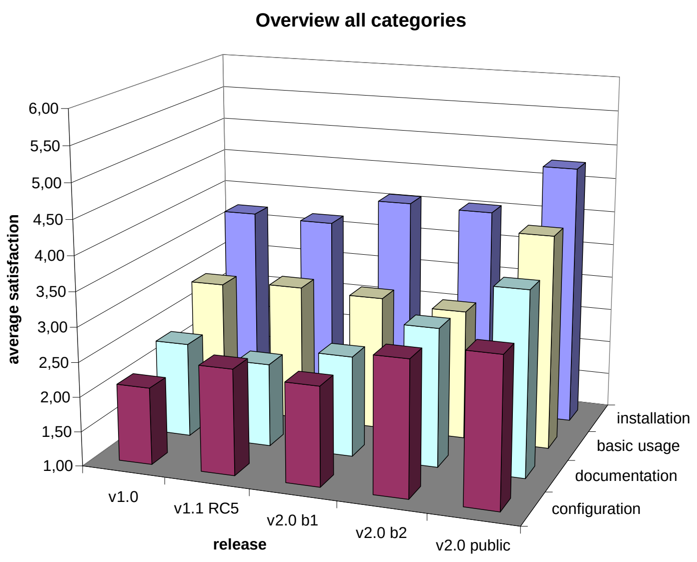


Figure 3.3: Overview of average user satisfaction for all categories and test items.

3.2.2 Installation

In Figure 3.4, the rating of the installation procedure is given per question. The ease of getting the install media, namely CD-ISO, has received the highest average results in a consistent manner. Nevertheless, some improvements have been recommended such as facilitating the access to the install media on the XtreemOS home page. The visibility of XtreemOS as a product has improved with the publication of XtreemOS 2.0, however, still too many mouse clicks are required to get to the download link. Occasionally, a slow ftp download connection has been reported but this may have been a temporal issue. Ease and speed of installation have also improved noticeably. In particular, starting with XtreemOS 1.1, preconfigured node types were introduced which became increasingly mature in the beta and public versions of XtreemOS 2.0. Meta-packages for different node type made the installation process much easier. Some inconsistencies of the installation steps with the instructions in the user guide(s) have been reported in the earlier versions. Though these problems have been tackled with the appearance of XtreemOS 2.0. Also post-installation of packages is easier, packages have been made available from the XtreemOS repositories. Some packages still fail to install, feedback has been reported.

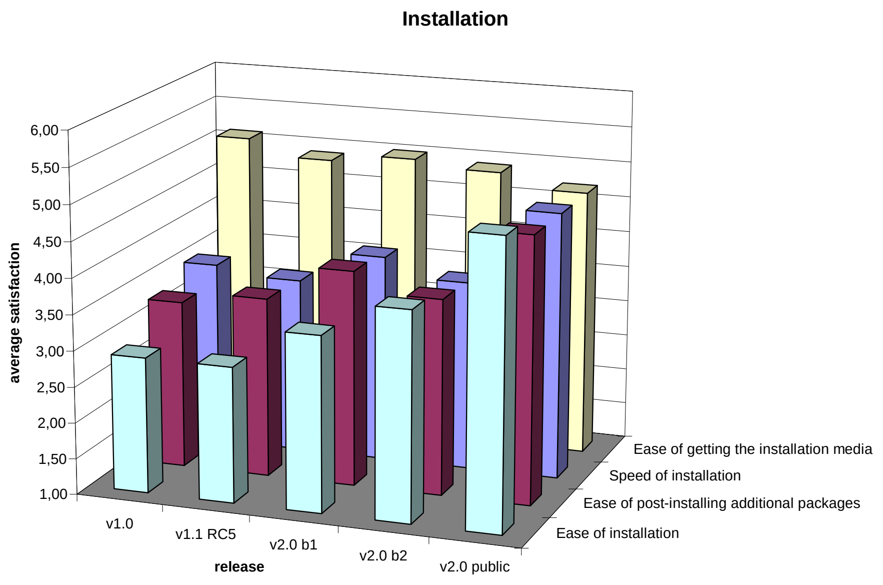


Figure 3.4: User satisfaction for category installation.

3.2.3 Configuration

Similar to the installation procedure, also the configuration has improved, especially for core and resource nodes (cf. Figure 3.5). The configuration of these nodes was a relatively tedious task in the earlier XtreemOS versions, whereas the configuration of a client is simpler and caused less problems reflected by the better rating. Coinciding with the introduction of automatized configuration procedures and pre-configured node types in the beta and public releases of XtreemOS 2.0 the execution of the configuration has been facilitated and is now at least as easy as the configuration of the client. Still the configuration of the latest XtreemOS release can be further improved by reducing the number of manual steps and workarounds, e.g. by introducing configuration scripts (either with default setup or with interactive parameter entry) and further self-explaining GUIs for the xosautoconfig tool. Also some lower level automation of certificate setup would be useful even if this process cannot be automated entirely.

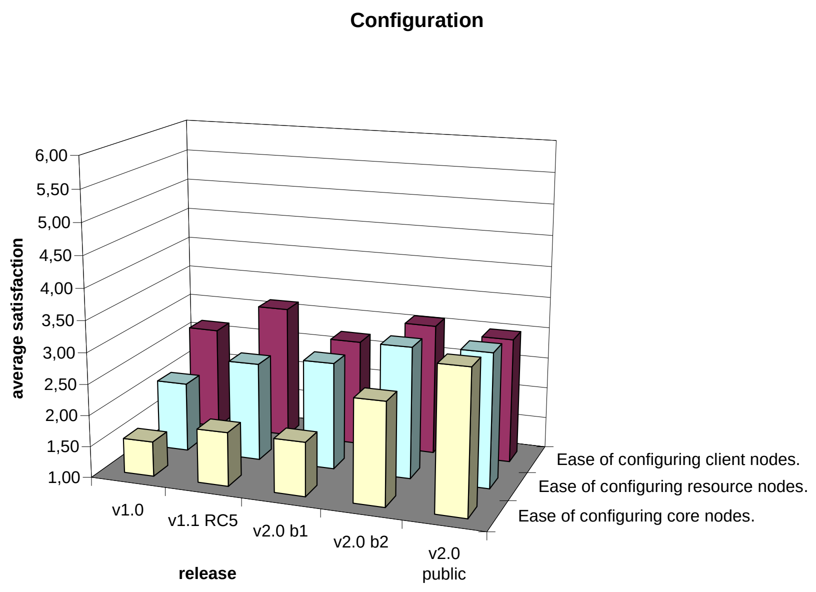


Figure 3.5: User satisfaction for category configuration.

3.2.4 Basic Usage

In Figure 3.6, a remarkable increase in the rating of basic usage actions can be observed. Most striking is the ascent regarding the stability of the software since the introduction of the public release of XtreemOS 2.0. This can be traced back to the enormous efforts for system integration, continued code stabilization, testing and debugging. Also the availability of VOLife may have contributed to a facilitated VO management. Whereas in earlier versions, management of users and VOs was reported to require way too much manual work, today recommendations restrict to minor bug fixes and minor usability enhancements. Basic usage may be further improved by, e.g., more detailed error outputs and by ensuring a proper startup of all services avoiding manual restart.

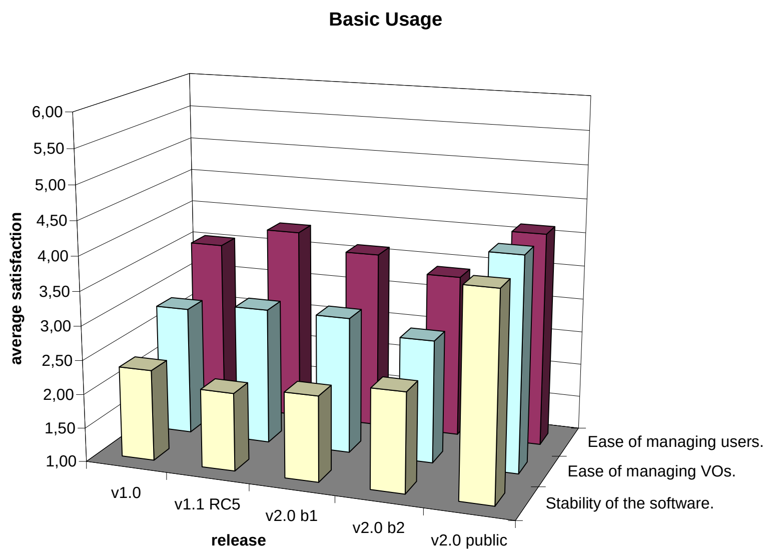


Figure 3.6: User satisfaction for category basic usage.

3.2.5 Documentation

Increased user satisfaction also applies to software documentation in all three properties, completeness, correctness and clarity (cf. Figure 3.7).

Probably, issues with the documentation of early XtreemOS versions gave rise for bad ratings in the other survey categories because the documentation did not help much to resolve problems. One major issue reported was the lacking synchronization between software development and documentation. This gap was most evident for the internal version 1.1 which is reflected by the lowest satisfaction values. The documentation was reported to miss required steps for OS setup, or, some instructions were incorrect or too technical for end-users. As of XtreemOS 2.0 beta 2 and the subsequent public release the situation has improved a lot. Primarily, the separation of the XtreemOS guide into an admin and a user guide has been highly appreciated. The synchronization between development and documentation writing has been streamlined and is reflected in way more consistent instructions. To

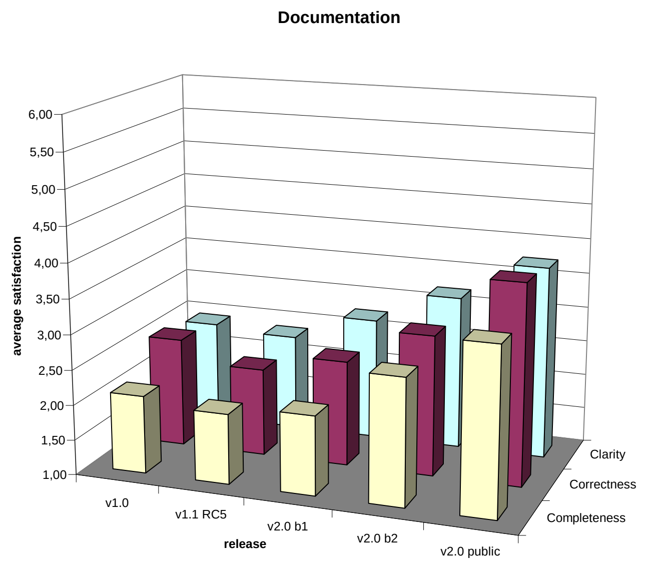


Figure 3.7: User satisfaction for category documentation.

further improve the quality of the documentation it is recommended to use more screenshots and to reduce the need to jump between sections.

3.3 Summary

The survey presented gave insights into the experiences with two public and three intermediate internal releases from the end-user perspective. The survey examined the satisfaction with the installation, configuration and basic usage of the install CDs provided as well as with the accompanying documentation. In all four categories, one could detect an remarkable upward trend with respect to end-user satisfaction. The ratings improved monotonously along all XtreemOS versions examined, most noticeable, however, is the leap made with the introduction of the public release of XtreemOS 2.0. Early major problems with lacking integration, instability, complicated manual setup, bugs and lacking synchronization between software development and documentation have been addressed to a far extend. Advancements with software integration have been reported and also the automatized installation and configurations tools have been added which render the adoption of the new OS much easier. One further major reason for improved satisfaction was the revised documentation which provides for more clarity, completeness and corrected many errors. Also the separation into a user and an admin guide is highly appreciated.

Further ways for improvements have been proposed. This includes debugging of various packages which still fail to install. Also the number of manual steps and work-arounds should be reduced, e.g. by introducing configuration scripts and further self-explaining GUIs, e.g., for the xosautoconfig tool. Furthermore, it would be useful to introduce lower level automation for certificate setup. Basic usage would benefit from more detailed error outputs and from ensuring a proper startup of all services which would avoid manual restart. And finally, it is suggested to further work on the quality of the documentation, e.g. by adding more screenshots and by reducing the need to jump between sections.

Summing up, end-users welcome the progress made regarding component integration, stability and basic usage and software documentation. The evolved maturity of the software product is reflected by thoroughly increasing satisfaction values and many positive comments for XtreemOS 2.0. Two of them shall be quoted here: "A big improvement over the previous release.", and "My main remark is that this version has improved a lot over previous ones. Keep the good momentum."

Chapter 4

Evaluation of XtreemOS Components

In this chapter, we present the evaluation of various XtreemOS components and features. Following an introduction to the overall test approach and test overview, this chapter presents the specifications and results for the experiments conducted.

4.1 Evaluation Overview

The evaluation of XtreemOS components is structured as follows:

- Node-level VO support (Section 4.2)
- Checkpointing and restart (Section 4.3)
- DIXI message bus (Section 4.4)
- XtreemOS API (Section 4.5)
- Distributed Servers (Section 4.6)
- Virtual Nodes (Section 4.7)
- Application execution management (Section 4.8)
- Data management (Section 4.9)
- Security services (Section 4.10)
- Mobile device flavor (Section 4.11)

The emphasis of the experiments conducted in this category is on evaluating the performance, scalability, stability and correctness of the respective XtreemOS developments. In this chapter, the experimentation results from the consortium shall be collected and presented in a common place. Apart from WP4.2, each work package in SP2 and SP3 contributed to the planning, specification, execution and documentation of the experiments. For this purpose, each development work package introduced a dedicated task (as defined in the DoW) and devoted manpower to organize the performance evaluation. In order to indicate the responsibilities for each test unit, the respective contributions from the various work packages and partners are clearly marked. Generally, WP4.2 focuses on the application-centric evaluation from the end-user's view whereas SP2 and SP3 put emphasis on lower-level performance benchmarking.

Table 4.1 gives an overview of the tested XtreemOS components along with the WP4.2-applications used for evaluation.

Table 4.1: Overview of XtreemOS components and their assigned WP4.2-applications used for testing them

Component Name	COMPSS	SPECWEB	JCAE	MODERATO	OPENTURNS	ZEPHYR	SRC	TREX	MaxDB	RBSM	IMA	JOBMA	WISS	GALEB	other
Node-level VO support														X	X
Checkpointing and restart		X	X			X				X				X	X
Mobile device flavor											X	X			X
XtreemOS API															X
Application execution management	X	X	X	X											X
Data management					X			X	X				X		X
Security services															X

Next to these application-centric tests, SP2 and SP3 contributed to the evaluation of components as listed in Table 4.2.

Table 4.2: Tested components with WPs and partners contributing to testing

Component Name	Contributing WP	Contributing Partner
Checkpointing and restart	WP3.3	UDUS
DIXI message bus	WP3.2	XLAB
XtreemOS API	WP3.1	VUA
Distributed Servers	WP3.2	VUA
Virtual Nodes	WP3.2	ULM
Application execution management	WP2.2, WP3.3	BSC, XLAB
Data management	WP2.2, WP3.4	CNR, XLAB
Security services	WP3.5	STFC
Mobile device flavor	WP2.3, WP3.6	TID

The following sections present the test documentation with a structuring derived from "IEEE Standard for Software Test Documentation", IEEE 829-1998 [20]. Accordingly, the documents in these sections cover the phases test planning, test specification, and test reporting.

Among others the *test plan* describes the scope, approach, resources, the items and the features to be tested as well as the test approach. Per XtreemOS component, one test plan is provided covering the test setup for all WP4.2 applications evaluating this feature.

The evaluation of an XtreemOS component is sub-divided into one or more *test units* where each test unit consists of exactly one *test specification* and one *test results* document. The test specification enumerates the test items, tested features and the test approach refinements for the given test unit. It is required that the specification follows a sound methodological approach which shall be applied to ensure accuracy, reproducibility and fairness of the tests. Test results have to be analyzed and presented in a comprehensible manner.

The *test summary report* summarizes the tests. For each XtreemOS component, we give a common conclusion for all applications testing it, summarize the results and outline future test activties.

4.2 Evaluation of Node-level VO Support

This section covers the software for node-level VO support and VOlife. The purpose, architecture and use cases are described in the XtreemOS deliverable D2.1.2 [8].

4.2.1 Test Plan

4.2.1.1 Responsibilities

These tests will be carried out by XLAB.

4.2.1.2 Test Items

The tests shall be done on Release 2.0 of XtreemOS updated with all the patches available at the time of testing.

4.2.1.3 Features to be Tested

This test plan includes testing the following features:

- the VO administration tools VOlife, including both the command-line interface as well as the web front-end (test units described in Sections 4.2.2 and 4.2.3),
- mapping global user identities to local user identities and group identities (test unit described in Section 4.2.4).

4.2.1.4 Features not to be Tested

Evaluation of the following features will be skipped in this iteration of the tests due to lack of time but is planned for the next iteration:

- translating VO-level access control policies into local OS-level access rights and capabilities,
- session management.

4.2.1.5 Overall Approach

The purpose of these tests is to evaluate the current version of software from the application and end-user perspective and to provide feedback to developers. We will thus focus on evaluating the higher-level design, features and usability of each module rather than covering a large part of possible inputs to each components. To ensure the feedback will be beneficial to the development, all tests will be done with the latest version of the software within the current major release version.

As not all of this software is intended to be used directly by the users or applications, some tests will be done through the application execution manager (AEM). For example, submitting a `whoami` job reveals what local account the global user identity is mapped to.

The tests will allow for assessing improvements with respect to the evaluation carried out in D4.2.5. Furthermore, the scope of the tests has been expanded towards more detailed tests also covering performance and scalability aspects.

4.2.2 Test Unit 01: Correctness of VOlife

4.2.2.1 Responsibilities

The tests in this test unit are to be carried out within WP4.2. The responsible partner is XLAB.

4.2.2.2 Test Specification

A testbed of two virtual machines was used with 748 MB RAM for the core node and 512 MB RAM for the resource node. They were hosted in VirtualBox 3.0.12 on a 2.40 GHz dual-core Intel-based computer with 2 GB RAM running Windows vista. Automatic update of both nodes was done right before the start of the test.

Test Items

This test unit tests the tools for configuring VOs and their membership.

Features to be Tested

This test unit tests the functional requirements of VO administration tools VOlife. Both the command-line and web interface are tested. A brief usability test of the web interface is also included.

This test unit tests requirements R21, R22, R24, R29; and, to a lesser extent, R25, R89, R91, R96.

Approach Refinements

Configuration

1. XtreemOS must be installed and configured on the two test nodes so that job submission is possible.
2. The original state of the XVOMS database must be dumped into a file with the command
   `mysqldump -u -p root xvoms -r xvoms.txt.`

Start The following steps are to be tested:

1. User1 creates an account in VOlife and tries to edit existing VOs. He must fail.
2. User1 creates a newVO.
3. User2 creates an account in VOlife and tries to edit existing VOs, including newVO.
4. User2 requests being added to newVO.
5. User1 approves the request and also adds User2 to a group and a role.
6. User1 also adds himself to a different group and role in newVO.
7. User2 checks that he is now a member of newVO.
8. User2 tries to modify group and role membership of newVO. He must fail.
9. User2 creates anotherVO.
10. User1 requests being added to anotherVO.
11. User2 approves the request and also adds User2 to a group and a role.
12. User1 checks that he is now a member of anotherVO.

Note that when a user tries to perform an illegal action, it is preferrable that:

- if using the web interface, the action is not available anywhere in the menu at all,
- if using the command-line tools, the action should return a helpful error message but must not in any case reveal too much information, which could hinder system security.

All these steps are to be repeated twice, once using the VOlifecycle web interface and once using the corresponding command-line VOlife utilities.

When using VOlifecycle:

1. Ensure that tomcat is running.
2. Open one web browsers on each of the two machines, and use them to execute the steps of the input case.
3. Check that the state of the VOs presented in the web interface is consistent with the expected output.
4. Check that each user can only view the properties, groups etc of his VOs.

5. Create multiple VOs, groups, roles, and users, and check whether this affects usability of the VOlifecycle web interface. For example, does the user have to scroll through long tables?

When using the command-line tools:

1. Use the command `volife_run.sh` to execute the steps of the input case. Before executing any VO administration steps ensure that the local account you are using has the XtreemOS user certificate of the VO administrator installed.

2. Check that the state of the VOs returned by the `volife_run.sh -list-all-vos` and `volife_run.sh -list-all-users` is consistent with the expected output.

3. Re-login as another local user who only has the XtreemOS user certificate of the VO user, and ensure that he cannot administrate the VO.

Wrap Up The state of the XVOMS database must be restored from dump with the commands

```
echo 'drop database xvoms;' | mysql -u root -p
echo 'create database xvoms; use xvoms; source xvoms.txt;' \
    | mysql -u root -p!
```

4.2.2.3 Test Results

The test was run on 2009-11-30 by Bojan Blažica, XLAB.

Procedure Results Functionality of the VOlife web interface worked as expected. All features that the current user is not authorized for were hidden in the graphical user interface.

On large grid deployment the administrators will have to scroll through long tables when managing users, groups, and roles because no search method is provided. Only the list of VOs has the option of filtering. It also takes a significant amount of time to perform certain actions. For example, listing all users takes approximately 5 seconds with 20 VOs, 1 group and role per VO and 20 users per VO.

The command-line interface is adequate to perform all tested operations and can be efficiently used in batch files. However, it does not use the XtreemOS user certificates nor has any provision to log in (see the next Paragraph, Anomalous Events).

Anomalous Events As mentioned above, the usability of VOLifecycle web interface could be improved. The command-line interface, on the other hand, does not use the XtreemOS user certificates nor has any provision to log in. This means that the local root on the core node can perform all operations on the database (which cannot be realistically prevented anyway), while no normal user is allowed to do anything.

4.2.3 Test Unit 02: Performance of VOlife

4.2.3.1 Responsibilities

The tests in this test unit are to be carried out within WP4.2. The responsible partner is XLAB.

4.2.3.2 Test Specification

The test was executed by on a testbed of two virtual machines. The core node had 748 MB RAM and the resource node had 512 MB RAM. They were hosted in VirtualBox 3.0.12 on a 2.40 GHz dual-core Intel-based computer with 2 GB RAM running Windows vista. Automatic update of both nodes was done right before the start of the test.

Test Items

This test case tests the VOlife tools for configuring VOs and their membership.

Features to be Tested

This test unit evaluates the scalability of VOlife by measuring the time required for certain operation in certain circumstances, as given below. The operations being timed are short, thus only the command-line interface can be tested because it can be automatically timed using the Linux `time` command.

This test unit tests requirements R21 and R30.

Approach Refinements

The scalability parameters that we test the performance against the size of the XVOMS database, i.e. the number of VOs, groups, roles, and users. The intent is to check whether a large number of these entities negatively impacts the usability or the performance of the VOlife tools.

The parameter being measured is the time needed to complete certain operations, such as creating an additional VO or checking group membership for a given user. The database sizes (number of entities – VOs, groups, roles, and users) that the tests are run on should be large enough to affect the performance, e.g. at least 1000 entities for the largest test. Each measurement should be repeated at least 50 times and the results averaged over the runs.

Configuration

1. XtreemOS must be installed and configured on the two test nodes so that job submission is possible.
2. The original state of the XVOMS database must be dumped into a file with the command
   `mysqldump -u root xvoms -r xvoms.txt.`

Set Up Using the command-line VOlife tools, fill the database with as many records as specified in the input.

Start Measure the time required for the following operations, using the command-line tools:

1. check VO and group membership of a given user; VO, group, and user given by names,
2. check VO and group membership of a given user; VO, group, and user given by IDs,
3. create new VO, group, and role,
4. create a new user,
5. add a user to a VO, group, and role.

Repeat the short operations as many times as necessary to get accurate measurements.

Wrap Up The state of the XVOMS database must be restored from dump with the commands:

```
echo 'drop database xvoms;' | mysql -u root -p
echo 'create database xvoms; use xvoms; source xvoms.txt;' \
    | mysql -u root -p
```

4.2.3.3 Test Results

The test was executed by Bojan Blažica, XLAB, on 2009-11-30.

Execution Description The procedure was run on 2009-11-30 following the above Test Unit desciption. The database sizes, i.e. the number of VOs, groups, roles, and users, were selected to be as large as practical, with the limiting factor being the fact that adding a VO, group, or user using the command-line interface takes more than one second.

Results The last columns give the time required for the five operations given in the Test Unit specification. Times in the table were averaged over 150 runs, except for the create user command, which was averaged over 50 runs.

VOs	groups and roles per VO	users per group	average execution time [s] for membership check by names	by IDs	create VO, group, and role	create user	add user to VO, group, and role
1	1	1	2.86	2.89	2.97	2.94	2.91
1	1	50	3.37	2.94	3.03	3	3.01
20	1	20	3.9	3.86	4.07	4.01	4.08

The measurements show a small but measurable effect of database size on the times required to execute these operations. In case of a very large database VO-life could fail to fulfill the times-related non-functional requirements. On the other hand these are all administrative operations that have no effect on overall performance of jobs, XtreemFS etc.

Anomalous Events As mentioned above, the times required to execute these operations in VOlife increase slightly but measurably with the database size.

4.2.4 Test Unit 03: Correctness of account mapping

4.2.4.1 Responsibilities

The tests in this test unit are to be carried out within WP4.2. The responsible partner is XLAB.

4.2.4.2 Test Specification

Test Items

This test unit tests the account mapping service.

Features to be Tested

The subject of this test unit is the correctness of the mapping from global VO user credentials to local groups and accounts on the VO-aware node.

Approach Refinements

This test design covers basic tests, which are best done by submitting jobs that run command-line utilities like `whoami` and `id`.

The input of this test case consists of the sequence in which the involved VO users submit their jobs and of the job length. The output consists of the username and group that they are mapped to.

The test will pass if the user is mapped according to her distinguished name and VO stored in the certificate. No password should be required, except for the passphrase of user's private key. The feature fails if:

- an unauthorized user manages to log into the node,
- an authorized user manages to log into the node, but is mapped incorrectly,
- two distinct users are mapped to the same UID,
- two users from a different VO are mapped to the same GID,
- an authorized user is denied access,
- grid users are not isolated properly once the account pool is used up.

Configuration

1. XtreemOS must be installed and configured on the two test nodes so that job submission is possible.
2. The original state of the XVOMS database must be dumped into a file with the command
   `mysqldump -u root xvoms -r xvoms.txt.`

Set Up Create a VO named VO1, containing Group1, Role1, and User1. Create a VO named VO2, containing Group2a, Role2a, and User1. Set up the local policies so that the default account and group mapping is used for the VO users:

```
xos-policy-admin-am -vo VO_ID --force
xos-policy-admin-gm -vo VO_ID --force
```

Add User2 to VO2 into Group2b, Role2b. Add User3 to VO1, Group1, Role1. Add a resource to both VO1 and VO2.

Start Execute the following steps, noting the account and group mapping in each case.

1. User1 submits 'id', acting as member of VO1.
2. User1 submits 'id', acting as member of VO2.
3. User2 submits 'id'.
4. User1 submits 'id' again, acting as member of VO1. Is he mapped to the same account as the first time?
5. Using the command `xos-policy-admin`, limit the account pool size to 2 accounts and 2 groups that should NOT be in the same range as the previously used account pool.

6. User1 submits ’id’.

7. User2 submits ’id’.

8. User3 submits ’id; sleep 60; id’. Is the first account from the pool reused?

9. User1 submits ’id; sleep 60; id’.

10. User2 submits ’id’. Is the submission rejected because of lack of free accounts?

11. Increase the account pool to a normal size.

12. Expire the certificate of one of the users by issuing a very short-lived certificate and waiting for it to expire.

13. User1 submits ’id’. Submission must fail.

Wrap Up The state of the XVOMS database must be restored from dump with the commands

```
echo 'drop database xvoms;' | mysql -u root -p
echo 'create database xvoms; use xvoms; source xvoms.txt;' | mysql -u root -p
```

4.2.4.3 Test Results

The test was executed by Bojan Blažica, XLAB, on 2009-12-22.

Execution Description The procedure was run on 2009-12-22 following the above Test Unit desciption.

Results In the first seven steps of the test procedure account mapping service behaved as expected. Users were mapped according to their distinguished names and VOs stored in their certificates and no passwords were required. After limiting the account pool size to 2 accounts and 2 groups, 60501 and 60502, two distinct users were mapped to the same UID. This happened during steps 8 - 10 in the test procedure. The following commands and the contents of the output files they produced show that User2 was mapped to the same UID (60501) as User3. Additionally, we can see the correctness of mapping different VOs to different groups - e.g., VO1 to group 60051 and VO2 to group 60238.

Commands:

```
[user3@core files]$ xsub -f id3sleep60.jsdl -c /home/user3/.xos/
    (cont.)truststore/certs/user3vo1.crt
[marjan@core files]$ xsub -f id1sleep60.jsdl -c /home/marjan/.xos/
    (cont.)user1vo1.crt
[matej@core files]$ xsub -f id.jsdl -c /home/matej/.xos/truststore
    (cont.)/certs/user2vo2.crt
```

Output files:

```
[root@core tmp]# cat id3.out
uid=60501(/CN=6b40da4d-8bf1-4d5f-ba64-f65058883921) gid=60051(
    (cont.)xosuser_g60051) groups=60051(xosuser_g60051)
uid=60501 gid=60051(xosuser_g60051) groups=60051(xosuser_g60051)
[root@core tmp]# cat id1.out
uid=60502(/CN=60a84dfc-a269-468f-8bad-a6ca2388fc72) gid=60051(
    (cont.)xosuser_g60051) groups=60051(xosuser_g60051)
uid=60502(/CN=60a84dfc-a269-468f-8bad-a6ca2388fc72) gid=60051(
    (cont.)xosuser_g60051) groups=60051(xosuser_g60051)
[root@core tmp]# cat id.out
uid=60501(/CN=d7eaf795-79c1-474a-9ab1-1829d44011bc) gid=60328(
    (cont.)xosuser_g60328) groups=60328(xosuser_g60328)
```

Subbmiting a job with an expired certificate failed as expected.

```
[marjan@core files]$ xsub -f idsleep60.jsdl -c /home/marjan/.xos/
    (cont.)truststore/user1vo1expired.crt
xsub: createJob: -1 (Generic exception)
certificate expired on 20091222111345GMT+00:00
```

Anomalous Events As mentioned above, submitting a job while all UIDs in the account pool are already mapped, causes two users to be mapped to the same UID. Such behaviour should be prevented at job submission.

4.2.5 Test Summary Report

4.2.5.1 Summary of Tests and Results

The functional requirements of VO and user administration tool were tested thoroughly. The web interface VOLifeCycle worked as expected and thus satisfied all the requirements. The command-line interface does not use the XtreemOS user certificates nor has any provision to log in, resulting in local root being able to do anything and no other user being able to use it at all. The former cannot be avoided, as local root can also edit the database manually etc, thus VOlife must run on a trusted node. The latter, however, limits the use of the command-line interface to test environments only. This shows that the CLI is provided just as a test tool. A VO administrator in a large grid environment thus has no practical means to automate repetitive administrative tasks. Furthermore, the administrator will have to scroll through long tables when managing users, groups, and roles because no search method is provided. Only the list of VOs has the option of filtering.

The performance test of VOlife shows that its scalability over large databases is good but not excellent. However, VOlife cannot really be a bottleneck of a grid environment because it does not affect performance of other XtreemOS components.

The account mapping performs as expected in normal usage, i.e. until the account pool is used up. Distinct grid users are mapped to distinct dynamicall created local accounts. The same user acting as a member of two distinct VOs is also mapped to distinct accounts to prevent VO interference. Members of the same VO are mapped to the same local group. The only detected problem was that once the account pool is used up two different users can be mapped to the same local account, thus they are not isolated - e.g., one of them can control jobs created by the other.

4.2.5.2 Conclusion and Directions for Future Work

We can conclude that the components for node-level VO support in XtreemOS Release 2 are adequate, with all major functional requirements met but performance and usability leaving some room for improvement.

Further testing is required for features skipped in this iteration, most of which relate to the account mapping service. All tests will also be repeated on later releases of XtreemOS to check for regressions.

4.3 Evaluation of Checkpointing and Restart

This section relates to the testing of the XtreemOS Checkpointing/Restart functionality developed as part of WP3.3.

4.3.1 Test Plan

4.3.1.1 Responsibilities

- WP3.3:
  - UDUS
- WP4.2:
  - SAP
  - BSC
  - EDF
  - XLAB

4.3.1.2 Test Items

Checkpointing and restarting have been implemented at multiple levels in the OS stack. There is a grid-level job checkpointer (JCP) that is able to control jobs in the grid through communication with a job-unit checkpointer (JUCP) on each node (XtreemOS grid checkpointer version 2.1). Using a common kernel checkpointer API, commands to the kernel-level checkpointer, which can change from node to node, are able to be translated into native commands in such a way that the kernel checkpointer implementation is obfuscated from the rest of the grid.

The kernel-level checkpointer used is a modified version of the Berkeley Library Checkpoint/Restart (BLCR) module (based on version 0.8.0). It is used as the basis for checkpointing in the regular flavour of XtreemOS. The checkpointer for XtreemOS adds functionality to BLCR so that it can save the executable and libraries as part of the checkpoint. The patches to apply these extensions can be obtained from svn:
svn+ssh://scm.gforge.inria.fr/svn/xtreemos/foundation/blcr/patchs

For LinuxSSI, the inbuilt LinuxSSI checkpointer is used as kernel-level checkpointer (version 2.0).

A user guide is available in section 7.1 of D3.3.6, and further information is provided on the wiki at `https://xtreemos.wiki.irisa.fr/tiki-index.php?page=Grid+Checkpointing+Service`.

4.3.1.3 Features to be Tested

The following features will be tested:

- Checkpointing of both simple and complex applications
- Restarting checkpoints from saved states
- Checkpointing functionality at various levels
  - Kernel-level
  - Container-level
  - Grid-level
  - Cluster-level (LinuxSSI)

4.3.1.4 Features not to be Tested

Currently, the following features will not be tested as they were not available at the time of testing:

- Checkpoint file management

4.3.1.5 Overall Approach

The purpose of these tests is to extensively evaluate the functionality, performance and scalability of the checkpointing/restart mechanisms in XtreemOS in a range of scenarios covering all implementations of this feature in various levels of the software stack. Due to the varying functional requirements of the WP4.2 evaluation applications being used in this set of tests, some applications have proved unsuitable for testing certain areas of the checkpoint/restart stack. Table 4.3 provides a breakdown of the requirements of each of the applications, according to functions that may make them suitable or unsuitable for certain tests.

Requirement	**RBSM**	**Zephyr**	**SPECWeb**	**Galeb**
External files	X	X	X	X
Sockets	X		X	
IPC			X	X
Multi-threaded	X		X	
Multi-process	X	X	X	X
GUI output	X			

Table 4.3: Functional requirements of WP4.2 applications involved in checkpoint/restart testing

From the requirements in table 4.3, the following structure has been decided for checkpoint/restart testing:

- Performance testing: WP3.3 CR Developers (UDUS)
  - Job checkpoint and restart (BLCR and LinuxSSI) - Test Unit 01

  - Incremental checkpointing (LinuxSSI) - Test Unit 02
  - Channel flushing (BLCR and LinuxSSI) - Test Unit 03

- Application-centric performance testing: WP3.3 CR Developers (UDUS)
  - Kernel-level checkpointing of Java applications with RBSM (SAP) - Test Unit 04
  - Container checkpointing benchmarks with SPECweb (BSC) - Test Unit 05
  - Grid-level checkpointing with Zephyr (EDF) - Test Unit 06

4.3.2 Test Unit 01: Job checkpoint and restart

4.3.2.1 Responsibilities

WP3.3, John Mehnert-Spahn (UDUS).

4.3.2.2 Test Specification

Test Items

A synthetic sequential application allocating 5 MB of memory and sequentially writing random values to it gets checkpointed and restarted using the grid checkpointer of XtreemOS (version 2.1 to be released 12|18|09), BLCR (version 0.8.0), LinuxSSI (version 2.0). The corresponding job consists of one single job-unit. The XtreemOS guide related to the second release covers the information of how to install and to use job checkpoint and restart. There is also a XtreemOS grid checkpointing related wiki page located under `https://xtreemos.wiki.irisa.fr///tiki-index.php?page=Grid+Checkpointing+Service`.

Features to be Tested

A job is checkpointed and restarted from the grid level by using the underlying checkpointers BLCR and LinuxSSI.

Approach Refinements

The testbed consists of nodes equipped with Intel Core 2 Duo E6850 CPUs (3 GHz) with 2GB DDR2-RAM. In a pre-boot execution environment LinuxSSI (with the associated LinuxSSI checkpointer installed) and XOS PC flavor (with BLCR 0.8.0 installed) nodes can be started. The Application Execution Management (AEM) service suite and the BLCR checkpointer are executed on one node, while a LinuxSSI cluster is constituted by a minimum of two machines, just one of it executing AEM.

4.3.2.3 Test Results

Table 4.4 indicates the times taken for the individual checkpoint and restart sequences and the total duration of coordinated checkpoint/restart of a job. The observed overhead is in the range of several seconds, however rather small compared to the times need to checkpoint/restart large grid applications. Future optimization approaches include an improved identification of contents that really need to be saved and those which can be skipped (e.g. binary files and unchanged memory contents). The latter can be achieved with incremental checkpointing.

Checkpoint					
	prepare	stop	checkpoint	resume	total
LinuxSSI (v0.9.3)	495,8	13,9	69,6	11,0	590,3
BLCR (v0.8.0)	381,2	40,1	250,5	5,3	677,1
Restart					
			rebuild	resume	total
LinuxSSI (v0.9.3)			2597,7	12,6	2610,3
BLCR (v0.8.0)			1659,3	5,7	1665,0

Table 4.4: Duration of grid checkpointing and restart in milliseconds

4.3.3 Test Unit 02: Incremental checkpointing

4.3.3.1 Responsibilities

WP3.3, John Mehnert-Spahn (UDUS).

4.3.3.2 Test Specification

Test Items

The test application is a synthetic sequential application which writes integer values to *random locations* at a 1 MB memory block in at *random times* using LinuxSSI (version 2.0).

Features to be Tested

In incremental checkpointing only those memory contents that have been changed since the previous checkpoint will be saved. The mentioned test application gets incrementally checkpointed under LinuxSSI.

Approach Refinements

Measurements have been performed on top of LinuxSSI whose checkpointer has been extended towards incremental checkpointing. The testbed equals the one described in section 4.3.2.2.

4.3.3.3 Test Results

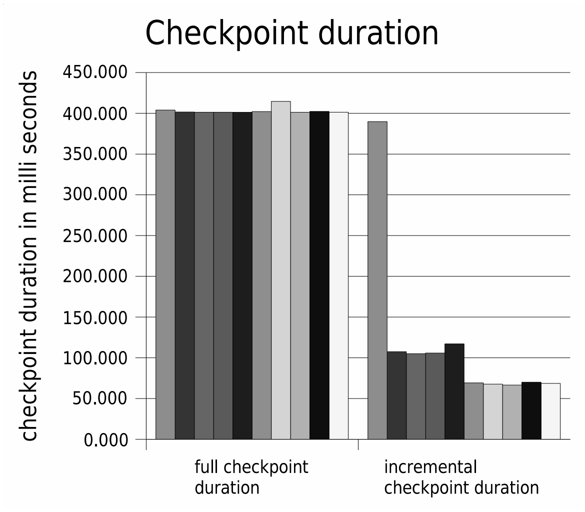


Figure 4.1: Duration of full and incremental checkpointing

Figure 4.1 shows the checkpoint duration of full checkpointing compared to incremental checkpointing when checkpoints are triggered in one second intervals. The diagram data indicates incremental checkpointing taking less time especially after the initial checkpoint. This is due to the need to save less, since pages get modified at random times. Figure 4.2 compares the image size of checkpoints taken under full and incremental checkpointing. In this scenario incremental checkpoint files are smaller than those of full checkpointing, especially after the initial checkpoint. Since the application modifies just a subset of pages per checkpoint interval, less memory pages need to be saved, thus less time is needed for the snapshot process. Incremental checkpointing does not always outperform full checkpointing. The more intense the write behaviour of an application is, the more meta data and modified pages occur. The later can lead to incremental checkpointing and restart having a bigger overhead than full checkpointing.

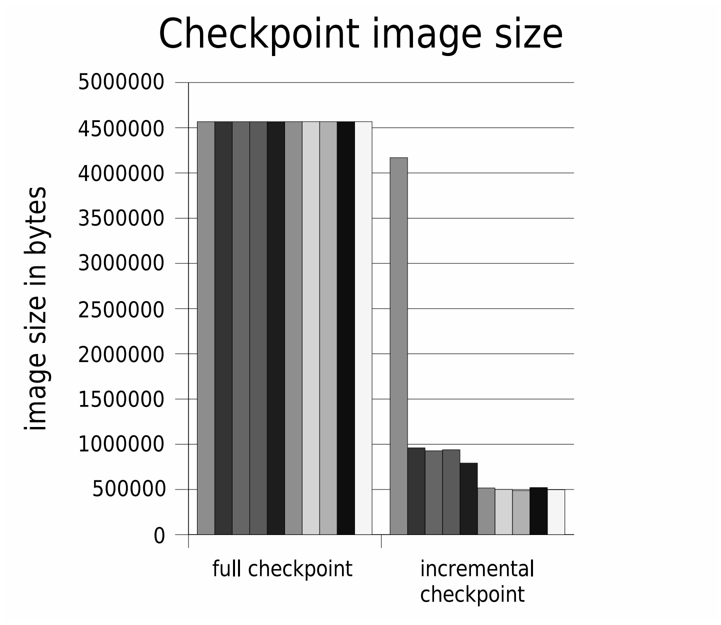


Figure 4.2: Checkpoint image size of full and incremental checkpointing

4.3.4 Test Unit 03: Channel flushing

4.3.4.1 Responsibilities

WP3.3, John Mehnert-Spahn (UDUS).

4.3.4.2 Test Specification

Test Items

A synthetic distributed client-server-application gets checkpointed and restarted. The so-called Channel Flushing Protocol (CFP) will not be part of XtreemOS release 2.x but 3.0.

Features to be Tested

Distributed applications whose parts communicate with each other require special attention in the context of checkpointing. For consistency reasons no orphan and lost messages may occur after a restart. Therefore communication channels must be flushed during coordinated checkpointing. The challenge comes with providing an application transparent channel flushing mechanism running on top of heterogeneous checkpointers that are unaware of each other. The CFP has been developed which allows to save and restore channels by having BLCR and LinuxSSI cooperating with each other without modifying them. However, this protocol is generic and allows other checkpointers to be involved as well.

Approach Refinements

The measurement of the so-called Channel Flushing Protocol (CFP), developed within XtreemOS, is based on a distributed test application, consisting of a client and server, communicating via a varying number of TCP channels. Per channel a different thread has been spawned at client and server side. Channel management is split into a pre- and post-checkpoint phase. The testbed equals the one described in Section 4.3.2.2.

4.3.4.3 Test Results

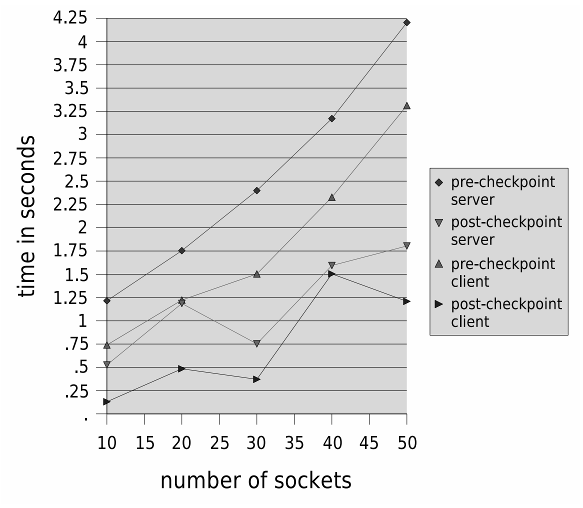


Figure 4.3: CFP behaviour on top of LinuxSSI and BLCR checkpointers with closed and reestablished channels

CFP can be used for two kinds of checkpointers. The first is able to save an application with open channels, since it can handle open socket descriptors during the snapshot procedure (LinuxSSI has recently been adapted towards this, other checkpointers still do not support this). These checkpointers are also able to restore sockets at restart time. The seconds type of checkpointer, which is supported by CFP, is incapable to save and restore sockets. Instead, CFP has to close and restore sockets in the pre-, post- and restart callbacks. Thus, a checkpointer does not see open socket descriptors at checkpoint and restart time.
CFP closes and reestablishes sockets: Figure 4.3 indicates the times taken for a client (checkpointed by LinuxSSI) and a server (checkpointed by BLCR) to flush, close and reestablish channels. The pre-checkpoint phase takes up to 4.25 seconds to handle 50 channels. The duration is mainly caused by the necessity for

the send and recv controller threads on both channel ends to synchronize with each other for serial channel flushing. Synchronization requires the involvement of the channel managers knowledge about existing channels and their states. Thus, control messages need to be exchanged between these three parties which adheres some overhead. A future solution will reduce *controller thread-channel manager-communication* and increase the channel managers ability to handle server and client requests in an improved parallel fashion.

CFP with open sockets: Figure 4.4 indicates the times taken for a client and

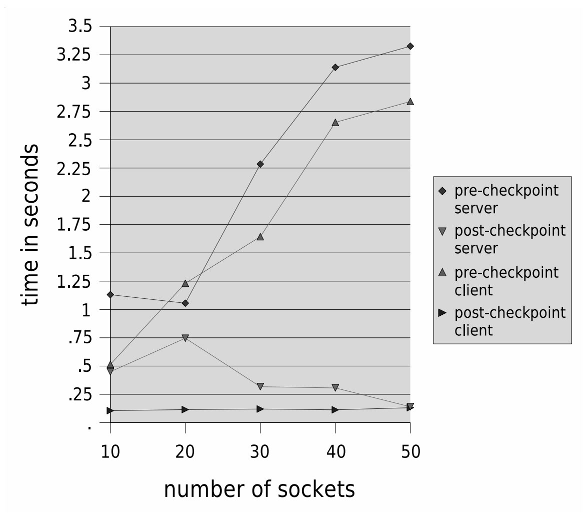


Figure 4.4: CFP behaviour on top of LinuxSSI and BLCR checkpointers with open channels

server to flush channels, that do not need to be closed and re-opened in the context of checkpointing. According to the pre-checkpoint phase from above, the pre-checkpoint phase takes less time (about 3.25 seconds to handle 50 channels) because no channels need to be closed. Furthermore, without rebuilding sockets and without socket reconnection the post-checkpoint phase is significantly shorter than the one from above. It takes just about 120 milliseconds for 50 channels. Obviously, checkpointers that can handle open sockets are preferred and the benefits will be even larger for wide area networks.

Another aspect is that CFP works on top of heterogeneous callback implementations without major performance drawbacks. While BLCR comes with its own callback implementation implicitly blocking applications threads, LinuxSSI does not. For the latter we have to use the generic callback implementation provided by XtreemGCP, [27].

4.3.5 Test Unit 04: Kernel-level checkpointing of Java applications with RBSM

4.3.5.1 Responsibilities

WP4.2: SAP

4.3.5.2 Test Specification

Test Items

We will test checkpointing with a Java application, using RBSM to automate the test environment and make decisions on which node to migrate the checkpoint to, and restart it.

Features to be Tested

Testing will involve checkpointing, manual migration and restarting of a simple Java application using the kernel-level checkpointing functionality of XtreemOS. This application will scale in memory size across the experiments, from approx. 8MB to approx. 1GB, and a timing will be taken for how long it takes to firstly create a checkpoint of the application and restart that checkpoint, then for how long it takes to migrate and restart a checkpoint, given variable network conditions. This will be timed and compared against a restart of the application to its previous state to determine the value of checkpointing for a particular application in a given scenario.

Approach Refinements

The purpose of this experiment is to show the performance of the checkpointing functionality within XtreemOS, and how it can be used to provide input for a rule-based system management tool in order to make decisions on the most efficient control method of a particular application.

To ensure that the overhead of processing the application as a job is removed, these tests will be performed using scripts to bypass the Grid-level checkpointer and access the kernel-level checkpointer directly. Also, due to implementation constraints in the current version, checkpoints had to be migrated manually.

These tests were performed on 2 identical nodes with the following specifications:

- HP Compaq dc7700 Convertible Minitower

- Intel Core 2 Duo 6600@2.4GHz with 4MB Level 2 cache
- 4GB DDR2 667MHz Dual Channel RAM
- 250GB Hard Drive

Configuration Have 2 computers ready to be installed as nodes (these can be virtual machines, but all must use the same architecture).

Set Up

1. Set up nodes with latest XtreemOS version, with patched kernel to enable checkpoint/restart
2. Deploy RBSM monitoring and control package to each system and run

Start The following steps can be performed inside the RBSM program:

1. Deploy Java application on 1st node
2. Configure memory size of Java application
3. Run script to start-up Java application
4. Retrieve timing for start-up to initial state
5. Start timing and begin to checkpoint Java application
6. Stop timing when checkpoint has been created
7. Start timing and begin to restart checkpointed Java application
8. Stop timing when checkpoint has been restarted
9. Run script to time migration and restart of checkpoint onto another node
10. Repeat steps 2 - 9, doubling the memory size used each time until application reaches approx. 1GB

Wrap Up Perform tests with varying network Quality of Service (QoS), i.e. simulated packet loss as a percentage of packets that are transferred successfully. The network will be tested with a QoS of 99%, 95% and 75%. If this is not possible, factor packet loss at varying QoS levels into calculations.

Contingencies If RBSM is unable to be run on XtreemOS version 2.0 for any reason, these steps should be completed manually to ensure a successful result. The use of RBSM is to ensure repeatable tests, and the results will feed back into the rule definition set.

4.3.5.3 Test Results

The first set of experiments were to time how long it takes to start up the Java application to its initial state, and then checkpoint and restart it as its memory size increases. These results are shown in table 4.5.

Memory size (MB)	Initial state (s)	Checkpoint (s)	Restart (s)
4.94	0.38	0.14	0.13
8.94	0.63	0.20	0.18
12.94	1.04	0.27	0.24
20.94	1.99	0.41	0.37
36.94	3.79	0.74	0.66
68.94	7.43	1.40	1.21
132.94	14.63	2.67	2.34
262.94	29.10	5.11	4.57
510.94	58.02	10.08	9.09
1028.94	115.87	20.80	18.88

Table 4.5: Timings for application reaching initial state compared to checkpoint and restart for increasing memory size

These results were plotted in the graph shown in Fig. 4.5. The graph shows a linear trend for both checkpointing and restarting up to a size of 1GB. As it was not possible to create a Java application larger than 1GB, the trend can not be shown to be continued up to a point. However, most applications that can be checkpointed within the limitations of the mechanism will be less than 1GB in memory size. For these applications, a checkpoint can be made quickly, with no impact on its operation.

Restarting a checkpoint is also faster, and in comparing it to the time taken to reach the initial state of the application, it can be seen to be significantly faster, although time taken for initial state will change, depending on the application. For an application with constantly changing state, there will also be the significant benefit that the state will be captured in the checkpoint, and that the state is recoverable very rapidly.

The second set of experiments were to migrate and restart the application on the 2nd node, and to compare this to a fresh run of the application to determine the case for migrating given simulated variable network conditions. The results are shown in table 4.6.

The results from the table have been summarized in Figs. 4.6 and 4.7. Fig. 4.6 displays the trend for applications up to 64MB in size. For the first 10MB, the curve shown is strange, but this could be due to operating system processing overhead in interpreting the command and putting it into action. Initially, a cold start of the application is faster than migrating and restarting a checkpoint up to approximately 30MB. From this point on, with QoS greater than around 85%, it is faster to migrate and restart.

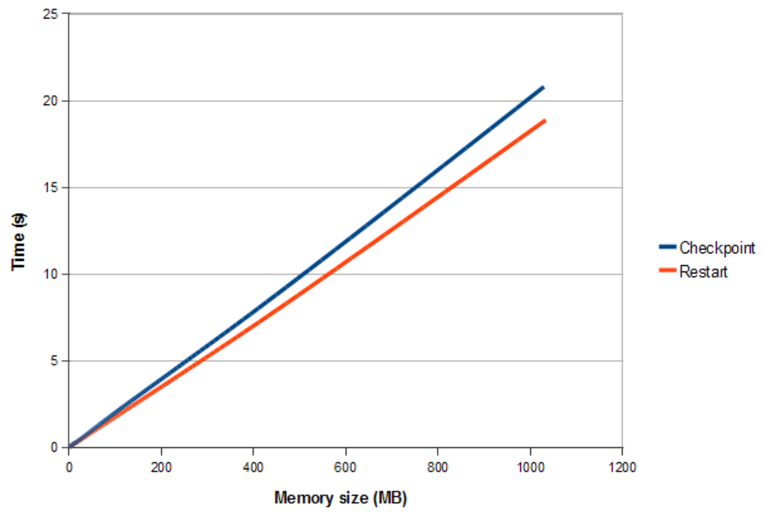


Figure 4.5: Comparison of time taken for checkpoint and restart over increasing memory size of application

Size (MB)	**Initial state (s)**	**Migrate/Restart - 99% QoS (s)**	**Migrate/Restart - 95% QoS (s)**	**Migrate/Restart - 75% QoS (s)**
4.94	0.38	0.67	0.69	0.84
8.94	0.63	0.89	0.92	1.12
12.94	1.04	1.28	1.32	1.61
20.94	1.99	2.05	2.13	2.59
36.94	3.79	3.64	3.77	4.60
68.94	7.43	6.79	7.02	8.57
132.94	14.63	13.11	13.56	16.55
262.94	29.10	25.72	26.61	32.49
510.94	58.02	51.01	52.78	64.42
1028.94	115.87	102.33	105.85	129.04

Table 4.6: Timings for application reaching initial state compared to migration and restart for increasing memory size over differing network conditions

This trend can be shown to continue in Fig. 4.7. With a QoS of greater than 85%, the time taken for migration and restart of an application up to 1GB in size is faster than the time taken to reach the initial state of the program.

The data received from these metrics shows that the case for checkpointing and migration, in most cases, it is faster than simply restarting the application. As the application runs for longer and the state changes significantly, the benefits of checkpointing and restart increase, as the state is kept in the checkpoint. However, even from an initial state, it can be shown that checkpointing is still a better choice in most cases.

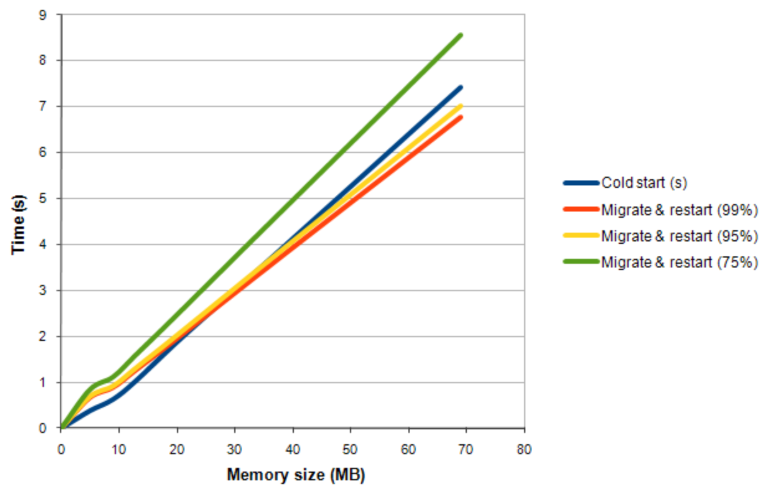


Figure 4.6: Comparison of time taken for migration and restart for varying network Quality of Service (QoS) up to 64MB

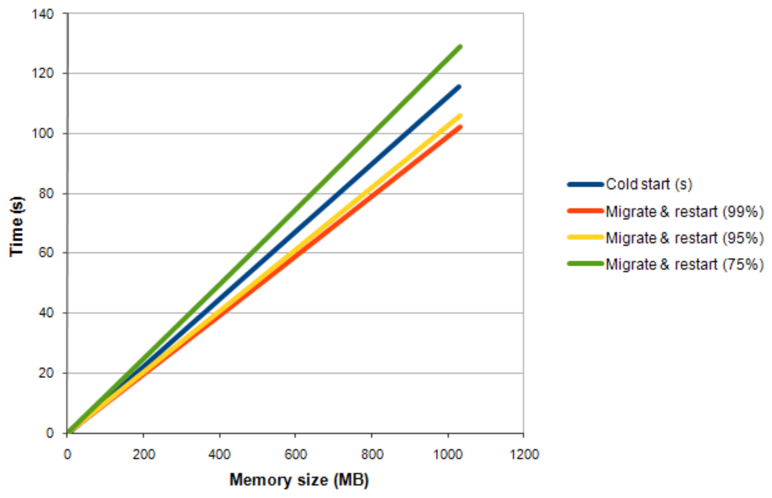


Figure 4.7: Comparison of time taken for migration and restart for varying network Quality of Service (QoS) up to 1GB

This leads to 2 useful conclusions:

- The kernel-level checkpoint/restart functionality built into XtreemOS provides a useful and quick way of managing application state and for recovering from error. However, there are limitations as to the kind of applications that can be checkpointed, so this must be factored into the decision to use this functionality.
- Depending on the network QoS and the current application state, there are decisions to be made as to the most appropriate and efficient course of action

to take when checkpointing or restarting an application. This provides useful feedback for the development of the Rule-Based System Management tool being developed as part of WP4.2, and will be implemented as a set of rules for application checkpointing/migration.

4.3.6 Test Unit 05: Container checkpointing benchmarks with SPECweb

4.3.6.1 Responsibilities

WP4.2, J. Oriol Fitó from Barcelona Supercomputing Center (BSC).

4.3.6.2 Test Specification

Test Items

We will test OpenVZ [33] [29] container-based checkpointing/restore mechanism as stand-alone feature.

Features to be Tested

The main goal is the testing of the XtreemOS feature "checkpointing and restart". Specifically, we want to test the OpenVZ container-based checkpointing and restore feature that will be integrated into XtreemOS.

OpenVZ is a container-based virtualization for Linux. It is free open source software, available under GNU GPL. In particular, OpenVZ creates multiple secure and isolated containers, i.e. Virtual Environments (VE), under a single kernel instance. Thus, a container can be rebooted independently and have root access, users, IP addresses, memory, processes and applications, among others.

OpenVZ checkpointing allows the "live" migration of a VE to the same or another physical server. The VE is "frozen" and its complete state is saved into a disk file. Afterwards, this file can be used to "unfreeze" (restore) the previously checkpointed VE. The whole process takes a few seconds and from the client's point of view it looks like a delay in processing, since the established network connections are also checkpointed/restored. Going more in detail, the OpenVZ checkpointing procedure consists of the following three stages:

1. *Freeze processes*, which moves processes to previously known state and disable network.

2. *Dump the container*, which collects and saves the complete state of all the container's processes and the container itself to a dump file.

3. *Stop the container*, which kills all the processes and unmount container's file system.

The restore procedure performs the same stages in an inverse mode.

Note that OpenVZ comes with a high-level command-line interface, i.e. *vzctl*, which is used to manage the Virtual Environments.

Approach Refinements

Due to constraints in the current integration of OpenVZ into XtreemOS 2.0 release, we are only able to evaluate it as stand-alone feature. We have decided to use the Apache Tomcat (v5.5) [40] as the web server to be checkpointed and restored. Actually, we have to checkpoint the Java Virtual Machine (JVM) in which the aforesaid web server is encapsulated.

We will consider the success of the feature if, at least, we are able to checkpoint and restart on the same node a container in which Tomcat is deployed and running. In addition, we will use SPECweb2005 benchmark [4] to validate that the checkpointing/restore operation is successfully performed, i.e. the state of the web server after a restore operation is the same than before the checkpointing action. This is easy to verify through SPECweb2005, because we will be able to check the benchmark results (i.e. web server performance) when performing a checkpointing/restore operation during the benchmark execution. In this way, we will be able to both validate that the web server is well checkpointed/restored and the loss of web server's performance due to this operation. Note that we don't test the migration of the container to another physical machine due to a constraint with the SPECweb2005 benchmark, which specifies that a web server under test must be into the same physical node during all the execution of a given test. Nevertheless, the migration operation is based on the checkpointing and restore procedures tested here.

Furthermore, and with the aim of testing the scalability of the OpenVZ mechanism, we will perform checkpointing/restore operations for different dimensions of Tomcat and JVM, e.g. different number of web server's threads, memory and input load (commonly expressed as the number of simultaneous user sessions within the web server). In this sense, we will present the time taken to checkpoint/restore these distinct containers.

Configuration On one hand, the configuration needed for running the Tomcat web server is based on a Debian Linux Operating System distribution with the 2.6.26 kernel version of OpenVZ for 64-bit architecture. We choose this system due to it is one of the recommended distributions by OpenVZ community. On the other hand, we have configured the needed environment for running the SPECweb2005 benchmark. It is composed by a Back-end database SIMulator (BeSIM) and the clients required to perform the input load to the web server under test.

Set Up The set up must follows the following steps:

- Create a new OpenVZ container
- Set the appropriate user beancounters (UBC), CPU, memory and disk parameters for the container
- Deploy Tomcat within this new container
- Check that the web server is available through the container IP, not the system IP
- Checkpoint and restore the container in the same node
- Show that the web server is still accessible at the same container IP and that its state is the same

Start The actions necessary to begin the test are:

- Check that OpenVZ kernel and application-level utilities (vzctl, vzprocps and vzquota) have been successfully installed
- Set the proper user beancounters (UBC), CPU, memory and disk parameters for each container created
- Configure the SPECweb2005 benchmark environment

Wrap Up There are no special actions to restore the environment.

Contingencies We have no estimation of any anomalous situation.

4.3.6.3 Test Results

We want to analyze and evaluate the OpenVZ capability of checkpointing and restore a container in which a Tomcat (v5.5) web server has been deployed. In this sense, we present results regarding both the OpenVZ checkpoint/restore scalability and the overhead introduced by this mechanisms.

OpenVZ scalability With the aim of analyzing the scalability of openvz checkpointing/restore mechanism, we present the results obtained from checkpointing and restore different dimensions of web servers, i.e. with different number of threads, memory allocated to the Java Virtual Machine (i.e. to the web server) and incoming load of the web server (expressed as simultaneous user sessions within the server).

First of all, in Figure 4.8 you can see the time OpenVZ takes to checkpoint and restore a Java Virtual Machine (with the Tomcat web server) with different amount of memory allocated to it.

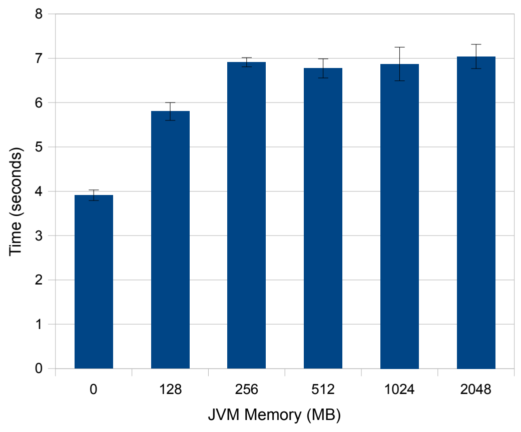


Figure 4.8: Time taken to checkpoint/restore a JVM with different amount of memory allocated to it.

If there is no JVM running (JVM heap memory usage 0 MB), the total time is around 4 seconds, while if there is a JVM running the total times remain almost constant around 6 and 7 seconds for a memory usage between 256 MB and 2048 MB. Consequently, OpenVZ shows a scalable behavior with respect to an increasing amount of memory to be checkpointed.

Secondly, in Figure 4.9 we illustrate the time differences in checkpointing/restore a Tomcat web server with different number of threads. Notice that all these tests were performed with a JVM with one gigabyte of memory allocated.

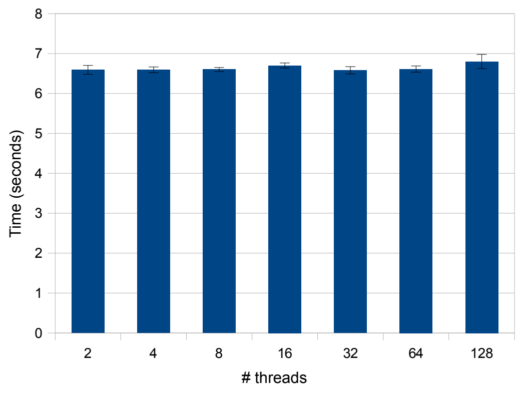


Figure 4.9: Time taken to checkpoint/restore a JVM with different number of threads, from 2 to 128.

The results suggest that the number of threads of the web server does not that affects the time needed to perform the container-based checkpointing and restore actions.

Finally, in Figure 4.10 we show the time needed for OpenVZ to checkpoint and restore a web server with different incoming loads (i.e. simultaneous user sessions). Notice that all these tests were performed using SPECweb2005 benchmark with the Support workload. In this way, we were able to produce several tests with different input load to the server under test. The JVM has been configured with 512MB of memory allocated and 32 threads.

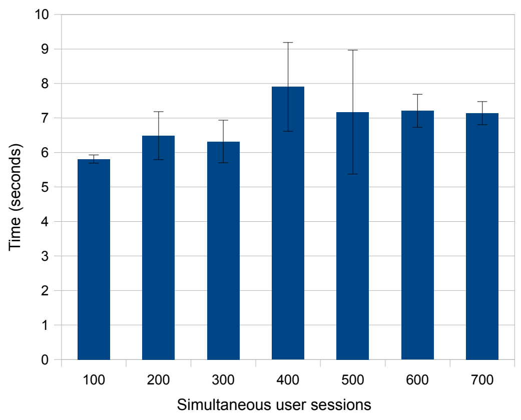


Figure 4.10: Time taken to checkpoint/restore a JVM with different input loads (expressed as the number of simultaneous user sessions within the web server.

We can state that the operation time increases more or less proportionally according to the incoming loads. In fact, the checkpointing and restore procedures treat all the network connections and, thus, the variability observed in Figure 4.10 can be a consequence of different connection states of the simultaneous user sessions simulated by the benchmark.

OpenVZ overhead In this second part we want to examine the loss of web server's performance introduced due to the operation of checkpointing/restart. For that purpose we use the SPECweb2005 benchmark in order to evaluate and obtain web server's performance metrics, such as throughput (expressed in requests per second), response time (in seconds) and Quality of Service (expressed as a percentage of the requests which response time is within a given ranges). In fact, through the executions of this benchmark we can check whether the state of the web server after a restore operation is the same or not than before the checkpointing. Notice that all these tests were performed using a JVM with 512MB of memory allocated and 32 threads.

Figures 4.11, 4.12 and 4.13 show the performance loss expressed in throughput, response time and Quality of Service web server's performance metrics, respectively. For the three graphics about QoS metric, we want to remark that a 'good QoS' is achieved if the response time is lower than a given threshold A, 'tolerable QoS' if it is between thresholds A and B and 'fail QoS' if the response time is greater than threshold B (where A < B).

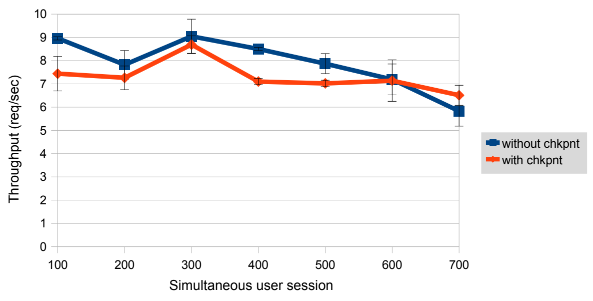


Figure 4.11: Web server's throughput for different input loads (simultaneous user sessions). Comparison between performing or not a checkpointing/restore operation.

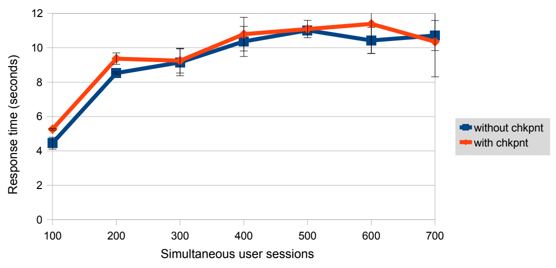


Figure 4.12: Web server's response time for different input loads. Comparison between performing or not a checkpointing/restore operation..

The results indicate that the loss of performance due to the checkpointing/restore mechanism of OpenVZ is quite low. In fact, Walters et al. [45] presents a performance comparison of different virtualization technologies. From the results presented in this paper we can see that OpenVZ has better performance than other virtualization alternatives like VMWare [44] and Xen [46]. Nevertheless, we have experienced a great variability in the web server's performance. For this reason, we present the 95% confidence interval.

Finally, we want to remark that after a checkpointing/restore operation the SPECweb2005 benchmark clients informs us that there are some reset connections (through the Exception message *[ERROR] Write to socket failed! IOException was: java.net.SocketException: Connection reset*).

OpenVZ bug During the setup of the environment to perform these tests we have found a problem regarding with checkpointing a container which has a NFS volume mounted. In fact, we have found that it is a known bug and this type of filesystem is unsupported by OpenVZ. Due to this restriction, we have had to deploy the web server into the local disk of the container.

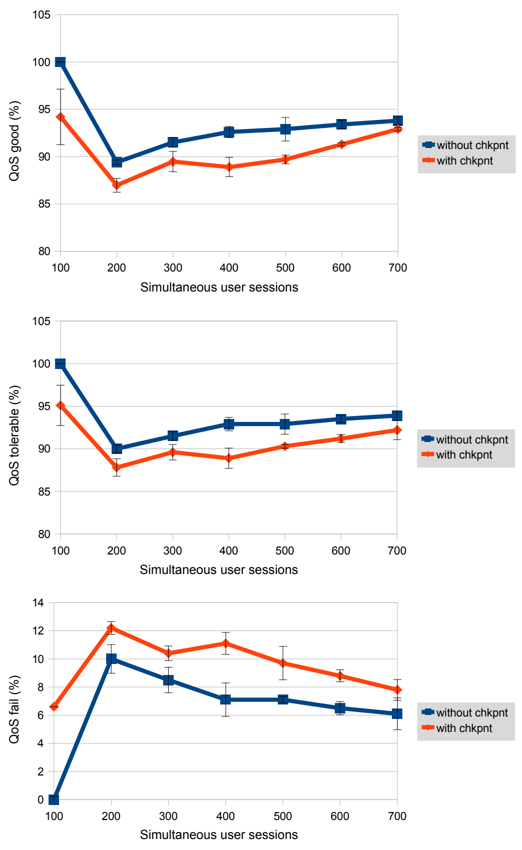


Figure 4.13: Web server's QoS for different input loads. Comparison between performing or not a checkpointing/restore operation.

Conclusions This experimentation conducted shows us that OpenVZ container-based checkpointing mechanism is a promising technique to be finally integrated into XtreemOS. As you can see in the results presented, we were able to checkpoint and restart a container with a Tomcat web server encapsulated. Through executions of SPECweb2005 benchmark we have checked the success of the checkpointing and restore operation. In addition, we were able to test its scalability and the loss of performance introduced.

Nevertheless, we have experienced that web server's performance on top of the OpenVZ virtualization layer is quite varying. However, the migration operation (i.e. checkpointing and restore) is quite efficient.

4.3.7 Test Unit 06: Checkpointing with Zephyr

4.3.7.1 Responsibilities

WP4.2: EDF

4.3.7.2 Test Specification

Test Items

Using the latest version of the BLCR checkpointer module available for XtreemOS: 0.8.0, we will test checkpointing features of the systems on a sequential EDF application, Zephyr.

Zephyr is a multidomain multigrid Preconditioned Conjuguate Gradient solver applied on academic 2D Navier Stokes driven cavity problem or 2D Bürgers viscous transport equation. Used essentially to bench HPC hardware if exists either in a standalone or distributed MPI version. In this test, we use the standalone one-node version of Zephyr.

More information on the calculation schemes implemented in Zephyr can be found in [24], and on their parallel efficiency in [23].

Features to be Tested

The tests for XtreemOS checkpointing and restarting mechanisms are focusing on the following features:

1. checkpointing of a sequential running application with or without results saved in a file.

2. restarting the application from the previously stored checkpoint.

Approach Refinements

The tests are focused on the EDF simulation application Zephyr run sequentially on one XtreemOS node. We will address the interaction of Checkpoint/Restart Mechanism with the filesystem.

The test case will perform the checkpointing and restarting tests using the EDF Zephyr application used in different modes (i.e: with or without saving results on files periodically).

The two individual features, the checkpointing and the restarting of an application are tested here as well as the overhead on the required memory of the checkpointed process and the size of the checkpointed archive file.

The tests are successful if the applications can be checkpointed and restarted with no noticeable differences. Anything else will be considered as a failure and reported back to the developers.

In detail, the test passes when Zephyr ends nominally its calculation with the results correctly saved in a file if required and with the state of the graphical interface correctly set if it was required.

The purpose of these tests is to see how the checkpointing and restart module will handle a Zephyr application running in different modes :

- Sequential execution on one node, with no results saved in a file,
- Sequential execution on one node, with results regularly dumped in a file,

Test were run on a Virtual Box Virtual Machine, running on a Dell Precision Laptop using Intel Dual Core T2300 1.66 GHz processor running Windows XP SP2. The memory dedicated to the virtual machine as been set so that no swapping either on Linux or the hosting system is observed while the tests are made : memory usage and activity have been constantly monitored on both systems.

Installation The test requires to install and configure properly working XtreemOS nodes from the XtreemOS 2.0 distribution.

Log Zephyr summary results are printed on standard output device and can be easily redirected to a file. These results are checked to determine if the checkpoint/restart process has succeeded.

Set Up The setup is as follows.

Start Zephyr is launched directly from command line and read the input values through standard input. These values determine the problem Zephyr solves, and the frequency of results to be dumped in a result file. Classically we will redirect an input file to stdin to feed the application.

Proceed The multiprocess is launched through a shell or submitted via AEM. In another shell the checkpoint/restart is performed as follows:

```
(In Shell A)
# ./zephyr.exe < input.dat > output.dat
#tail -f ouput.dat

(In shell B)
# cr_checkpoint --kill --save-private --save-shared <PID>
# cr_restart context.<PID>
```

Measure The present test do not require special measurement setups.

Stop Zephyr has to reach the end of its calculation.

Wrap Up This test does not require an explicit wrap up. However, it is meaningful to remove the stored checkpoints after the test in order to keep the hard disc clean.

Contingencies In case of anomalies, the applications should simply be restarted.

4.3.7.3 Test Results

In order to estimate the overhead of Checkpoint/restart in elapsed time and memory, we run Zephyr on problems of different sizes. Zephyr solves a partial differential equation on a $[0,1] \times [0,1]$ square discretized in 32^2, 64^2, 128^2, 256^2, 512^2, or 1024^2 degrees of freedom.

Problem Size	**time per iteration (s)**		
(degrees of freedom)	w/o C/R	with C/R	**ratio**
32^2 = 1024	3,55E-003	3,55E-003	1
64^2 = 4096	1,42E-002	1,37E-002	1,04
128^2= 16384	8,39E-002	7,91E-002	1,06
256^2= 66536	3,83E-001	3,84E-001	1
512^2= 262144	1,39E+000	1,39E+000	1
1024^2= 1048576	5,62E+000	5,62E+000	1

Table 4.7: time per iteration with and without Checkpoint/Restart feature activated

Problem Size	**size of process (MB)**			**size of**
(degrees of freedom)	w/o C/R	with C/R	**ratio**	**C/R file (MB)**
32^2 = 1024	3,8	3,83	1,01	0,59
64^2 = 4096	5,3	5,39	1,02	1,81
128^2= 16384	11,54	11,31	0,98	5,7
256^2= 66049	35,75	36,29	1,01	21,75
512^2= 262144	133,97	134	1	85,03
1024^2= 1048576	460,3	478,52	1,04	205,54

Table 4.8: Memory usage and checkpointed file size with and without Checkpoint/Restart feature activated

The results from the table have been summarized in Figs. 4.14 and 4.15. They demonstrate the following points about checkpoint/restart feature:

- it works as expected: final results from Zephyr stopped and restarted via BLCR are exactly matching including the produced output file.
- on Fig. 4.14, its memory overhead is shown to be negligible : memory size of the process with or without checkpointing activated is roughly constant.
- As well, the same Figure shows that the overhead observed on the calculation time needed per iteration is less than 5%.

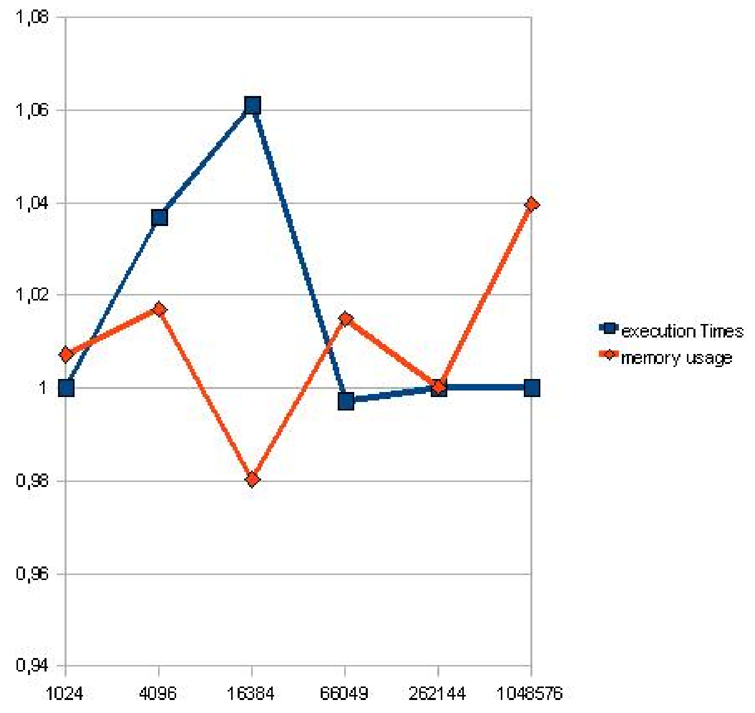


Figure 4.14: Memory use and size of the checkpointed archive file with respect to the problem size (logarithmic scale)

- on Fig. 4.15, the size of the checkpointed archived file is shown to be less than 5% larger than the memory used by the process.

4.3.8 Test Summary Report

4.3.8.1 Summary of Tests and Results

The experimentation completed for checkpoint/restart functionality provided good coverage of the features provided by XtreemOS. All main functions relating to checkpointing worked adequately, despite a number of limitations:

- kernel-level checkpointing is unable to checkpoint complex, multi-process applications
- container functionality was not available in the 2.0 release
- graphical applications are not checkpointable in any form

Despite these limitations, checkpointing has been shown to have good performance and scalability, as shown in the kernel-level testing, and the functionality

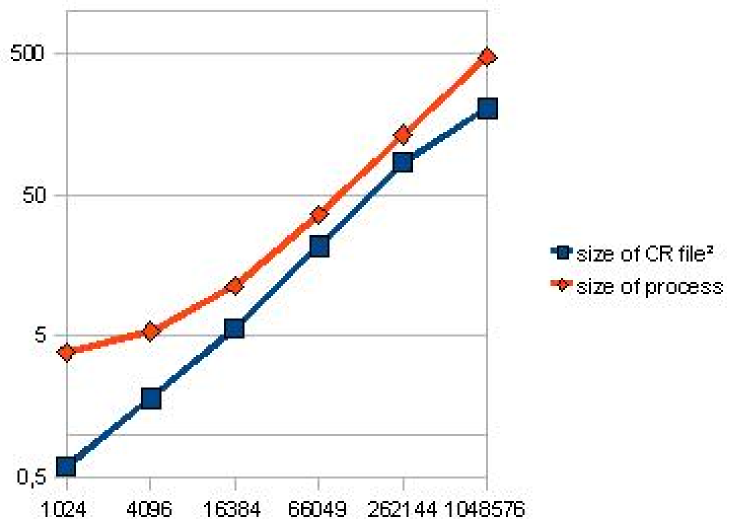


Figure 4.15: Time per iteration and memory footprint of the Zephyr process with and without Checkpointing feature activated

provided by the checkpointing and migration of containers provides a useful alternative to larger virtualisation formats.

This functionality holds significant benefit for a number of WP4.2 applications, particularly Rule-based System Management (RBSM). As the RBSM application is able to make decisions based on the state of the systems under its control, checkpointing and migrating processes and containers between nodes allows for a powerful additional layer of control for the application.

The data received from these experiments will be fed back into the development of the application in order to make the decision and control process more robust, intelligent and efficient, thereby strengthening the demo being prepared for WP4.4.

4.3.8.2 Conclusion and Directions for Future Work

Given the strong base provided by the XtreemOS 2.0 release, the next step would be to expand the usefulness of this functionality by removing a number of the aforementioned limitations. This should be implemented without causing any regression in terms of the functionality and performance already displayed in these experiments.

Later releases should use a similar approach to testing for comparisons, and also include a wider range of experimentation, including other test applications not used in these experiments.

4.4 Evaluation of the DIXI Message Bus

DIXI is a communication bus, a middleware for staging services developed for XtreemOS. It features the ability to stage services developed in Java, distribution of the services throughout the grid, the facility to publish the services' access points and service call invocation that is abstracted from the means of the service message exchange.

4.4.1 Test Plan

4.4.1.1 Responsibilities

The DIXI framework is the irresponsibility of WP3.2 and has been developed by XLAB. The tests will also be carried out by XLAB.

4.4.1.2 Test Items

The focus of the tests is the main DIXI library, packaged in the `dixi-main` package. The client-side component in XATI, also included in the test, is packaged in the `dixi-xati` package. At the time of the test, the components were the development versions of the release 3, commonly known under version 3.0.2.

4.4.1.3 Features to be Tested

We will test the following features of DIXI:

- Staging the services and exposing their interfaces to the message bus.
- The invocation of the service calls defined by the service interface in an asynchronous manner from another service.
- The invocation of the service calls defined by the service interface using client library (XATI), which occurs in a synchronous manner.
- The DIXI's ability to redirect the service requests to an access point capable of handling the request.

We will test the features in the following combinations:

- Two services, one of which will act as a server (the invoked service), another service, which will act as a client (the invokee service) and both will be staged on the same node, sharing the in-memory message bus (Test 1).
- A staged service invoked by a XATI client from the same node, using the network message bus (Test 1).

- Two services, one of which will act as a server (the invoked service), another service, which will act as a client (the invokee service), staged on two separate nodes. The message bus in this case will need to use the network interface and use the built-in redirection facility (Test 2).

- A service staged on another node than the XATI client, requiring the DIXI to use the network message bus as well as the redirection facility (Test 2).

4.4.1.4 Features not to be Tested

The DIXI framework, in part, contains tools that help develop a DIXI service. We will not test and compare them, because they need to only be used at the design time, and their outcome is implicitly tested within the features used during the runtime.

4.4.1.5 Overall Approach

To find a solution to compare DIXI to, we selected CORBA. This is a well-known, mature standard for providing distribution of functionality, meaning that it also provides the ability of communication between components running on different nodes. The functionality is provided in servers, which publish their presence in CORBA's naming service. The developer expresses the capabilities of the service using an IDL-formatted interface. The CORBA's tools can use the interface to generate the stubs, imported by the client. The client then uses the naming service's facility to locate the registered servers, or consume a server descriptor string, to resolve the target server's details such as the host and port. A reference bound in runtime can then be used like an ordinary class, with the underlying CORBA's layers marshaling the calls to the network and towards the server.

In this respect, the DIXI's capabilities resemble those of CORBA's, making it a good candidate for testing the performance against DIXI's. The community and the market offers several implementations of the CORBA ORB. However, for best comparison, we require that the implementation used for the test supports Java for both server and client. This is due to the fact that DIXI is Java-based. As a result, we chose the standard CORBA that ships along the Java virtual runtime and developing environment.

The focus of the benchmarking and comparison is the speed of the communication between entities, measuring the latencies that are to be expected when using either platforms. The basic assumption we have before benchmarking is, that the system is properly configured and initialized, and that invoking a service call (in case of DIXI) or the server call (for CORBA) will succeed. Both the DIXI service and the CORBA server use the same class implementation, which exposes a method, performing a simple multiplication. This means that the differences in the performance will be due to the framework used, not the actual service's implementation.

4.4.2 Test Unit 01: co-located service staging

4.4.2.1 Responsibilities

Both the developed code and the performance tests are the responsibility of XLAB within the WP3.2.

4.4.2.2 Test Specification

Test Items

In this test unit we test the DIXI framework, installed in `dixi-main` package. The client-side component in XATI, also included in the test, is packaged in the `dixi-xati` package. The packages are installable following the standard package installation procedure. The installation and usage guides are a part of the standard XtreemOS Administration Guide.

Features to be Tested

We will test the following features of DIXI:

- Staging the services and exposing their interfaces to the message bus.
- The invocation of the service calls defined by the service interface in an asynchronous manner from another service.
- The invocation of the service calls defined by the service interface using client library (XATI), which occurs in a synchronous manner.
- The DIXI's ability to redirect the service requests to an access point capable of handling the request.

We will test the features in the following combinations:

- Two services, one of which will act as a server (the invoked service), another service, which will act as a client (the invokee service) and both will be staged on the same node, sharing the in-memory message bus.
- A staged service invoked by a XATI client from the same node, using the network message bus.

Approach Refinements

In the test we would like to measure the latencies that we can expect when issuing a call to a service sharing a host or a DIXI daemon. This is not a very realistic test, but it establishes a baseline performance of the framework. The performance of the XATI client invocation is also a good reference for comparison with CORBA's performance.

Considering that basic CORBA provides a point-to-point invocation between the CORBA client code and the CORBA server code, the CORBA's benchmark consists of a single scenario. In this case, the implementation of making a CORBA call appears like calling the method of the class that implements the logic. We use the synchronous calls, meaning that the service call's stub blocks the execution of the thread until it receives the return value, whereupon the calling thread resumes.

In the test, we are measuring the time it takes between the call to the stub and the collection of the return value, assuming different scenarios and set-ups. To obtain a more reliable benchmark, we invoke the call several times in the loop, measuring the time of the whole loop. The result is therefore the whole time elapsed for the series of invocations and, indirectly, the average call time of a single invocation.

Each call takes three parameters: multiplication factor a, multiplication factor b and a certificate identifying the caller. The choice of adding the certificate as a parameter stems from the fact that many API calls in XtreemOS require the inclusion of the user's certificate as a credential for performing access control, and for extracting crucial attributes that pertain to the user. The choice of the framework also defines the format in which the certificate can be exchanged. Namely, in the Java code, a specific class (`X509Certificate`) can hold the certificate details and is used for exchanging the credentials. In CORBA, however, one is bound to a limited set of IDL types, and therefore the only supported way to exchange such variables is by serializing them into a string or array. This also means that the developers using the framework need to invest additional effort to convert the custom classes to and from the CORBA supported classes.

The result of tests, performed on a standard hardware (64-bit Intel CPU with 1.6GHz and a gigabit network interface) are measured in ms of the time elapsed for the invocation.

4.4.2.3 Test Results

As explained in the previous section, the tests we carried out were done in a series of batch method invocations. Each batch was composed of a certain number of subsequent method invocations, and we measured the time it takes for each batch to complete. The Figure 4.16 shows the result of the tests performed with all the entities (services, servers, clients) running on the same node. The side-by-side comparison shows the timings of the following test items, performed independently:

- **CORBA** is a reference measure, involving a CORBA client calling the CORBA server.
- **DIXI client** uses a client (XATI) that invokes a DIXI call directly.
- **DIXI service seq.** represents the timings of one service, acting as a client, that invokes another service's call similar to a synchronous manner, i.e., the next asynchronous call occurs only in the call-back of the previous call's result.

- **DIXI service** also involves a client service calling a server service, but the invocations occur in an asynchronous manner, i.e., the client does not wait for the response of the previous service call before invoking the next one. The overall time still signifies the time elapsed between the first invocation and the last result's collection.

The chart on Figure 4.16 shows the time elapsed for each batch depending on the number of iterations within each batch.

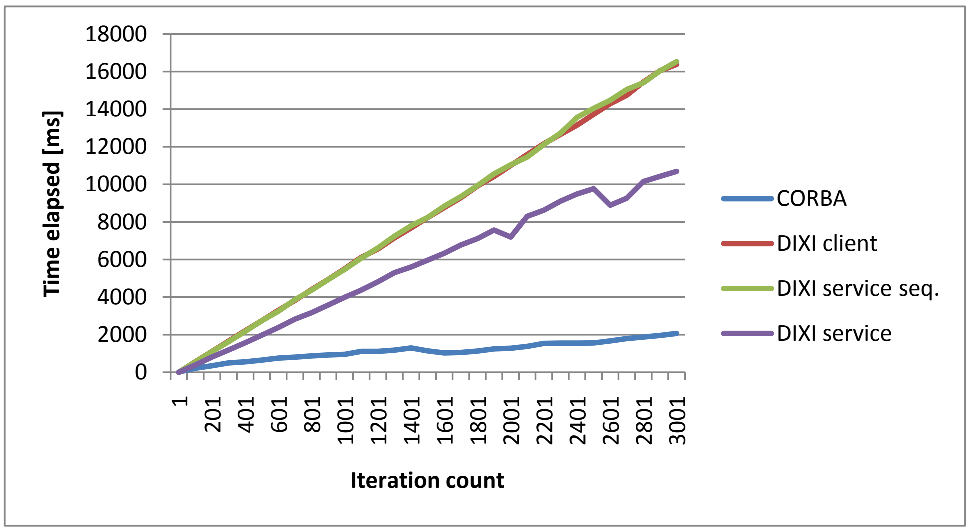


Figure 4.16: The overall duration of the service calls with an increasing number of subsequent calls. All client and server entities were on the same node.

The result shows a linear trend, meaning that the requests get served in due time. This is also demonstrated in Figure 4.17, which shows the average time of each invocation, i.e., the time elapsed for the batch divided by the number of invocations within the batch.

The results show that the DIXI invocations take longer time than the CORBA's. We can explain that by additional work that the framework needs to take for marshalling the service calls and demarshalling the service messages passed in the exchange. The **DIXI client** and the **DIXI service seq.** represent the time it takes for the invocation to yield a result. Both timings are comparable, meaning that a client service is in a similar position as a XATI client. The interpretation of **DIXI service seq.** is less straightforward because it does not show the time needed for a single method invocation. On the average, however, it shows the throughput of the system, which queues the requests and serves them in the order they arrive.

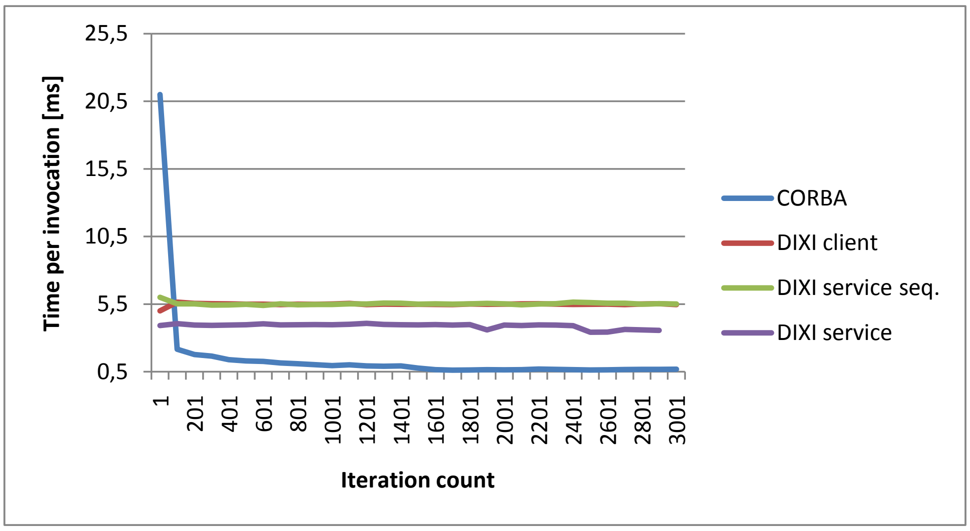


Figure 4.17: The average duration of one service calls within an increasing number of subsequent calls. All client and server entities were on the same node.

4.4.3 Test Unit 02: distributed service staging

4.4.3.1 Responsibilities

Both the developed code and the performance tests are the responsibility of XLAB within the WP3.2.

4.4.3.2 Test Specification

Test Items

The test items in this test unit are the same as the ones in Test Unit 01.

Features to be Tested

We will test the following features of DIXI:

- Staging the services and exposing their interfaces to the message bus.
- The invocation of the service calls defined by the service interface in an asynchronous manner from another service.
- The invocation of the service calls defined by the service interface using client library (XATI), which occurs in a synchronous manner.
- The DIXI's ability to redirect the service requests to an access point capable of handling the request.

We will test the features in the following combinations:

- Two services, one of which will act as a server (the invoked service), another service, which will act as a client (the invokee service), staged on two separate nodes. The message bus in this case will need to use the network interface and use the built-in redirection facility.
- A service staged on another node than the XATI client, requiring the DIXI to use the network message bus as well as the redirection facility.

Approach Refinements

This test unit is identical to the Test Unit 01 in all except the distribution of the entities taking part in the test. Here we perform a much more realistic test where the client of the invocation runs on a different host than the server offering the functionality. The tests therefore take into account the network transport. Further, for DIXI, the tests also include the ability to redirect the service messages to the access point of the host that can handle the service call.

4.4.3.3 Test Results

The basic premise of the tests is similar to the one the previous test unit:

- **CORBA** is a reference measure, involving a CORBA client calling the CORBA server.
- **DIXI client** uses a client (XATI) that invokes a DIXI call. The client sends service message to the DIXI daemon on the same host, and the daemon has to redirect the service message via the network to the host running the target service.
- **DIXI service seq.** represents the timings of one service, acting as a client, that invokes another service's call similar to a synchronous manner, i.e., the next asynchronous call occurs only in the call-back of the previous call's result.
- **DIXI service** also involves a client service calling a server service, but the invocations occur in an asynchronous manner, i.e., the client does not wait for the response of the previous service call before invoking the next one. The timing shows the time from the first invocation until the call-back of the result that arrives last.
- **DIXI Remote Client** uses a client (XATI) that invokes a DIXI call by sending the service message to the DIXI daemon hosting the target service, thus involving no redirections.

The chart on Figure 4.18 shows the time elapsed for each batch depending on the number of iterations within each batch. In this case we also see a linear trend, also shown with flat averages on Figure 4.19.

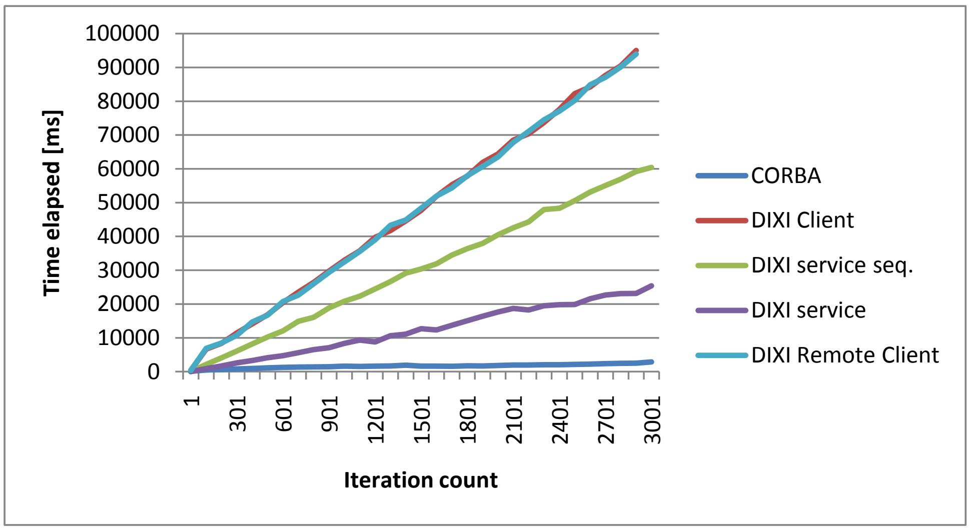


Figure 4.18: The overall duration of the service calls with an increasing number of subsequent calls. Server entities were on a different node than the host of the client entities.

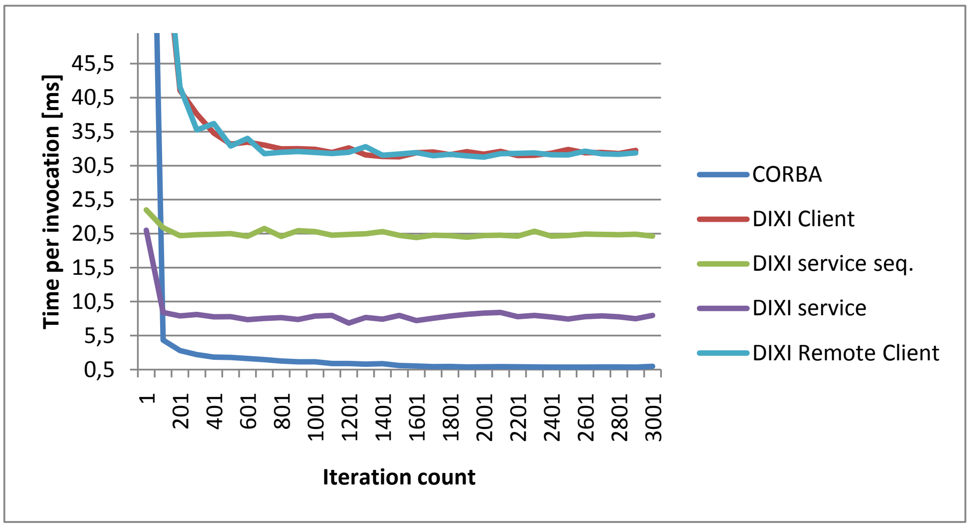


Figure 4.19: The average duration of one service calls within an increasing number of subsequent calls. Server entities were on a different node than the host of the client entities.

The comparison between the timings show that the network communication introduces additional delays in the service invocations. Again, the **DIXI service** results show us the average speed of the server serving each request. The time it takes for the result to requester are shown in the **DIXI service seq.** and it includes the network round-trip delay. The **DIXI client** and the **DIXI Remote Client** show comparable results with the effect of having to pass the service message back to the client.

We can explain the high delays introduced by the network by an overhead due to the extensive amount of information stored in each service message. A closer examination of result logs suggests that the latencies may be introduced by the Apache Mina networking library, and the fact that we have not yet optimised the message transport process by reducing the size or the service message's skeleton or compressing the payload.

4.4.4 Test Summary Report

4.4.4.1 Summary of Tests and Results

In the tests we measured the performance of the DIXI message bus and staging environment in terms of the time it takes to serve a request. The tests involved various scenarios that we based on the features provided in the DIXI runtime. The basic feature represents the ability to stage a service and thus provide the functionality of the service's interface. Through DIXI, a client needing the functionality can use a service call that is not different from making an ordinary Java call. The client of a DIXI service may be a third-party program that uses the XATI library, making synchronous calls to the service. In XtreemOS, the services will invoke other services' calls. This means that a service can appear in a role of a server, providing the functionality, or a client, invoking other services' functionality. Further, we can stage multiple services in one node, or, more commonly, distribute them throughout the grid. We performed benchmarking in set-ups representing each of these scenarios. As a reference middleware and messaging bus we used CORBA.

4.4.4.2 Conclusion and Directions for Future Work

The results show that in the current version of DIXI there is a penalty for the middleware's added functionality that manifests as increased time needed for each invocation to finish with a result. This is due to the fact that, during the development of DIXI, we have put a lot of emphasis in enriching the framework with features, while the optimising of important aspects of the solution and speeding up the message exchange has been planned for later stages of the development. The speed-ups can be achieved, for instance, by replacing the generalised service message marshalling with a content-aware one. Many of the elements of the service message are occurring often, and the overall structure of the message is defined in advance. Further, we need to examine the possibilities to speed up the Apache Mina library used for the networking.

4.5 Evaluation of XtreemOS API

The XtreemOS API has been designed to grant access to XtreemOS's features and services. In order to enable as many applications as possible, the API strives for alignment with both the Linux and the Grid world. This has been achieved by using the OGF-standardized *Simple API for Grid Applications* (SAGA) [18] as the basis for the XtreemOS API. For XtreemOS-specific functionality, SAGA has been extended. SAGA together with these extensions is called XOSAGA, forming the XtreemOS API. The design of XOSAGA has been an iterative process, gradually extending to the full scope of XtreemOS functionality and improving over early API design weaknesses. This XOSAGA design process has been documented by the XtreemOS deliverables D3.1.1 [50], D3.1.2 [51], and D3.1.5 [52].

The XOSAGA API has been and is being implemented in three programming languages: Java, C++, and Python. The state of the XOSAGA implementations at the time of writing this deliverable is contained in the deliverables D3.1.8 [53] and D3.1.9 [54]. Some parts of XOSAGA are scheduled for the final implementation (March 2010) only.

4.5.1 Test Plan

The performed tests are divided into two categories. The first category contains micro benchmarks for estimating the performance of the XOSAGA implementations, comparing them to the performance of using the underlying service interfaces directly. The second category contains application-driven evaluations of the XOSAGA API.

4.5.1.1 Responsibilities

The micro benchmarks of the XOSAGA implementations are being performed by WP 3.1 (partner VUA). The application-driven evaluations are performed by WP4.2 (partner BSC).. . .

4.5.1.2 Test Items

For the performance-oriented micro benchmarks, we have created a Subversion tag for each of the tested XOSAGA implementations. Each tag marks the exact version of the implementation that was used in the tests. They will be referred to with the respective test units.

4.5.1.3 Features to be Tested

The performance micro benchmarks have the sole purpose of quantifying the runtime overhead caused by using the integrated XOSAGA API, compared to using the plain, underlying service interfaces. Due to the size of the XOSAGA interface, we have focused on assessing the core functionality needed by most applications:

file I/O using XtreemFS and job submission using AEM. These tests have been performed for all three test units (programming languages) described below.

4.5.1.4 Features not to be Tested

The performance micro benchmarks do not address any other XOSAGA feature not mentioned above. In particular, the SAGA *Streams*, *Monitoring*, and *RPC* packages have been left out. The XOSAGA packages *Sharing* and *DS* have not yet been available for these tests.

4.5.1.5 Overall Approach

4.5.2 Test Unit 01: Java XOSAGA – Performance

4.5.2.1 Responsibilities

WP3.1, Thilo Kielmann, Mathijs den Burger, and Emilian Miron from VUA.

4.5.2.2 Test Specification

Test Items

We have created a Subversion tag for each of the tested XOSAGA implementations, marking the exact version of the implementations that were used in the tests. All XOSAGA implementations are located in subdirectories of the Subversion URL `svn://scm.gforge.inria.fr/svn/xtreemos/grid/xosaga`. The Java XOSAGA implementation is located in SVN, relative to the above URL:

Implementation	Tag relative to **`svn/xtreemos/grid/xosaga/`**
Java	`java/tags/benchmarks-D4.2.6`

Features to be Tested

We test the following features of the Java SAGA API:

- Job submission.

  Experiment: execute a simple job of known runtime ('/bin/sleep 60') on an XtreemOS node and wait for its completion using:

  a) a program on top of AEM's XATI API

  b) a program on top of XOSAGA

  Both a) and b) are performed 10 times and the average values are computed. The difference between a) and b) shows the overhead of job submission via the XOSAGA API.

- Namespaces.

  Experiment: execute a set of namespace operations on an XtreemFS volume using:

  a) a program that explicitly mounts an XtreemFS volume and operates on the locally mounted directory using Java's native local file API via the XtreemFS client
  b) a program on top of XOSAGA

  Both a) and b) perform the following set of operations on an (empty) XtreemFS volume:

  1. create 10 directories ('/dir000' to '/dir009') with 10 subdirectories each ('/dir000/subdir000' to '/dir009/subdir009')
  2. in each of the 100 subdirectories, create 10 text files ('file000' to 'file009'). The contents of each file is its own full path
  3. print the type (file or directory) and size (in bytes) of all entries in the volume
  4. find all files and directories in the volume with '01' in their name
  5. delete all files and directories

  Repeat these operations 10 times and measure the total execution time. The difference between the time for a) and b) shows the performance overhead for namespace operations of the XOSAGA API.

- File I/O.

  Experiment: execute a set of file I/O operations on an XtreemFS volume using:

  a) a program that explicitly mounts an XtreemFS volume and operates on the locally mounted directory using the native local file API via the XtreemFS client
  b) a program on top of XOSAGA

  Both a) and b) perform the following set of operations on an (empty) XtreemFS volume:

  1. create a binary file '/foo' of 100 MB size.
  2. copy '/foo' to '/bar'
  3. read '/bar'
  4. delete all files

  Repeat these operations 10 times and measure the total execution time. The difference between the time for a) and b) shows the performance overhead of the XOSAGA API.

Approach Refinements

The job submission tests have been performed on two nodes of the XtreemOS testbed at VUA. These (rather old) nodes have the following characteristics:

- 1GHz Pentium-III
- 1GB RAM
- 20GB harddisk
- 100Mbit/s Ethernet

These machines were chosen to evaluate an XtreemOS installation, performing job submission across two separate nodes. The exact performance of the machines is not relevant as we are only interested in the added overhead of XOSAGA, compared to the underlying software, here AEM's XATI.

The namespace and file I/O tests have been performed on a standalone workstation with the following characteristics:

- AMD Phenom X4 9600B
- 6GB RAM
- 500 GB harddisk, 7200RPM
- 1Gbit Ethernet
- Ubuntu Linux 9.10 x86_64

This machine was chosen as it was capable of running a local XtreemFS installation. As with the job submission tests, the actual performance numbers of the underlying machine are not relevant as we are only interested in the overhead added by the XOSAGA layer, here on top of the XtreemFS client. By using a machine with a local XtreemFS installation, we exclude all network influence on the results.

4.5.2.3 Test Results

Figure 4.20 compares job submission for Java XOSAGA and AEM's XATI. As the completion time of the 'application' was exactly 60 seconds, it can be seen that AEM adds almost 10 seconds overhead on top. The overhead caused by XOSAGA on top of AEM (XATI) is negligible.

Figure 4.21 compares namespace operations on a locally mounted XtreemFS volume for Java XOSAGA and native Java using the XtreemFS client. In average, XOSAGA adds about 5 seconds on top of native Java, accumulating the overheads of the hundreds of operations invoked by this test.

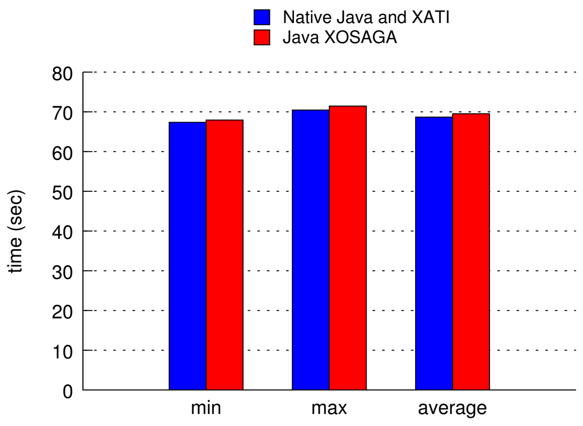


Figure 4.20: Comparing the job submission completion times for Java XOSAGA and native Java/XATI.

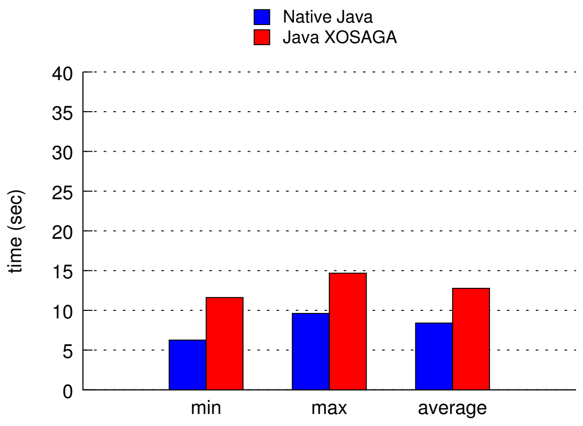


Figure 4.21: Comparing the completion time of namespace operations for Java XOSAGA and native Java via the XtreemFS client.

Figure 4.22 compares file I/O operations on a locally mounted XtreemFS volume for Java XOSAGA and native Java using the XtreemFS client. In average, XOSAGA adds about 3 seconds on top of native Java, indicating some moderate overhead for passing the file data through the XOSAGA engine.

4.5.3 Test Unit 02: C++ XOSAGA – Performance

4.5.3.1 Responsibilities

WP3.1, Thilo Kielmann, Mathijs den Burger, and Emilian Miron from VUA.

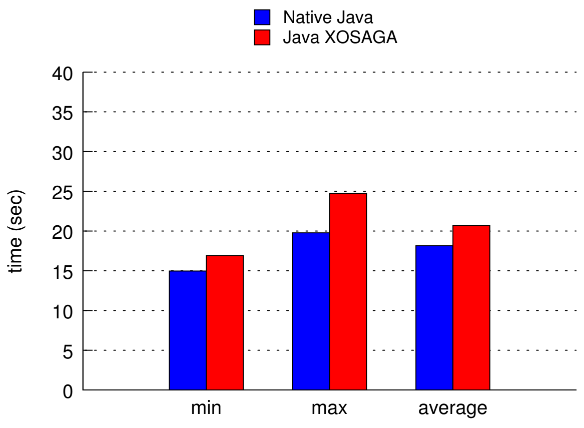


Figure 4.22: Comparing the file I/O operations for Java XOSAGA and native Java via the XtreemFS client.

4.5.3.2 Test Specification

Test Items

We have created a Subversion tag for each of the tested XOSAGA implementations, marking the exact version of the implementations that were used in the tests. All XOSAGA implementations are located in subdirectories of the Subversion URL `svn://scm.gforge.inria.fr/svn/xtreemos/grid/xosaga`. The C++ XOSAGA implementation is located in SVN, relative to the above URL:

Implementation	Tag relative to `svn/xtreemos/grid/xosaga/`
C++	`cpp/tags/benchmarks-D4.2.6`

Features to be Tested

We test the following features of the C++ SAGA API. These tests are equivalent to the Java-based tests. The detailed descriptions of the Java-based tests also apply here. Here, we merely summarize the software that has been compared to.

- Job submission.

  Experiment: execute a simple job of known runtime ('/bin/sleep 60') on an XtreemOS node and wait for its completion using:

  a) a program on top of AEM's XATICA API

  b) a program on top of XOSAGA

- Namespaces.

Experiment: execute a set of namespace operations on an XtreemFS volume using:

a) a program that explicitly mounts an XtreemFS volume and operates on the locally mounted directory using the Boost C++ filesystem library via the XtreemFS client

b) a program on top of XOSAGA

- File I/O.

  Experiment: execute a set of file I/O operations on an XtreemFS volume using:

  a) a program that explicitly mounts an XtreemFS volume and operates on the locally mounted directory using the standard C++ I/O classes via the XtreemFS client

  b) a program on top of XOSAGA

Approach Refinements

The C++ tests have been performed on exactly the same machines as the Java-based tests.

4.5.3.3 Test Results

Figure 4.23 compares job submission for C++ XOSAGA and AEM's XATICA. As the completion time of the 'application' was exactly 60 seconds, it can be seen that AEM adds almost 10 seconds overhead on top. The overhead caused by XOSAGA on top of AEM (XATICA) is negligible.

Figure 4.24 compares namespace operations on a locally mounted XtreemFS volume for C++ XOSAGA and native C++ using the Boost filesystem library and the XtreemFS client. In average, XOSAGA adds about 8 seconds on top of native C++, accumulating the overheads of the hundreds of operations invoked by this test. Thisindicates a higher runtime overhead of the C++ XOSAGA engine, compared to its Java counterpart.

Figure 4.25 compares file I/O operations on a locally mounted XtreemFS volume for C++ XOSAGA and native C++ using the Boost filesystem library via the XtreemFS client. In average, XOSAGA adds about 3 seconds on top of native C++, indicating some moderate overhead for passing the file data through the XOSAGA engine.

4.5.4 Test Unit 03: Python XOSAGA – Performance

4.5.4.1 Responsibilities

WP3.1, Thilo Kielmann, Mathijs den Burger, and Emilian Miron from VUA.

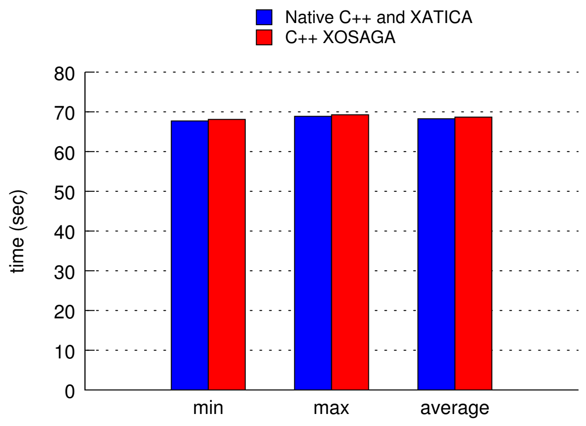


Figure 4.23: Comparing the job submission completion times for C++ XOSAGA and native C++/XATICA.

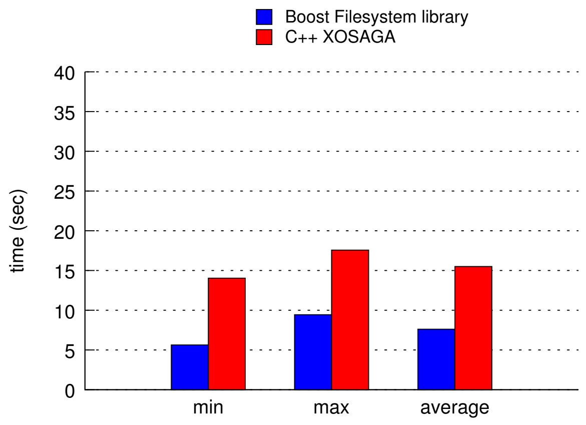


Figure 4.24: Comparing the completion time of namespace operations for C++ XOSAGA and native C++ using the Boost filesystem library via the XtreemFS client.

4.5.4.2 Test Specification

Test Items

We have created a Subversion tag for each of the tested XOSAGA implementations, marking the exact version of the implementations that were used in the tests. All XOSAGA implementations are located in subdirectories of the Subversion URL `svn://scm.gforge.inria.fr/svn/xtreemos/grid/xosaga`. The Python XOSAGA implementation is located in SVN, relative to the above URL:

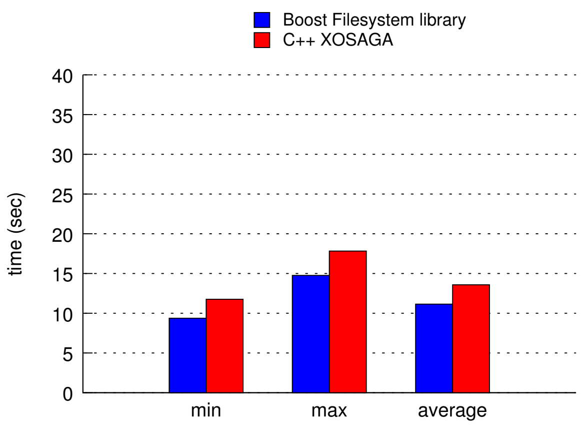


Figure 4.25: Comparing the file I/O operations for C++ XOSAGA and native C++ using the Boost filesystem library via the XtreemFS client.

Implementation	**Tag relative to `svn/xtreemos/grid/xosaga/`**
Python	`python/tags/benchmarks-D4.2.6`

Features to be Tested

We test the following features of the Python SAGA API. These tests are equivalent to the Java-based and C++-based tests. The detailed descriptions of the Java-based tests also apply here. Here, we merely summarize the software that has been compared to.

The Python XOSAGA implementation is a layer on top of the Java implementation, based on the Jython interpreter of the Python language. All tests in this unit have been performed using the Jython interpreter.

- Job submission.

  Experiment: execute a simple job of known runtime ('/bin/sleep 60') on an XtreemOS node and wait for its completion using:

  a) a Python program accessing AEM's XATI API
  b) a program on top of XOSAGA

- Namespaces.

  Experiment: execute a set of namespace operations on an XtreemFS volume using:

  a) a program that explicitly mounts an XtreemFS volume and operates on the locally mounted directory using the Python modules `os`, `os.path`, `shutil`, and `fnmatch` via the XtreemFS client

   b) a program on top of XOSAGA

- File I/O.

  Experiment: execute a set of file I/O operations on an XtreemFS volume using:

   a) a program that explicitly mounts an XtreemFS volume and operates on the locally mounted directory using the Python modules `os`, `os.path`, and `shutil` via the XtreemFS client

   b) a program on top of XOSAGA

Approach Refinements

The Python tests have been performed on exactly the same machines as the Java-based and C++-based tests.

4.5.4.3 Test Results

Figure 4.26 compares job submission for Python XOSAGA and Jython with AEM's XATI. As the completion time of the 'application' was exactly 60 seconds, it can be seen that AEM adds almost 10 seconds overhead on top. The overhead caused by XOSAGA on top of AEM (XATI) is negligible.

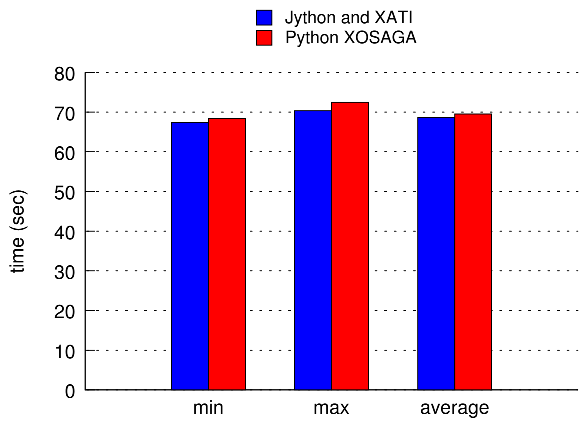


Figure 4.26: Comparing the job submission completion times for Python XOSAGA and native Jython/XATI.

Figure 4.27 compares namespace operations on a locally mounted XtreemFS volume for Python XOSAGA and native Jython modules via the XtreemFS client. In average, XOSAGA adds about 7 seconds on top of native Jython, accumulating the overheads of the hundreds of operations invoked by this test. This indicates a modest runtime overhead of the Python XOSAGA layer on top of the Java XOSAGA engine.

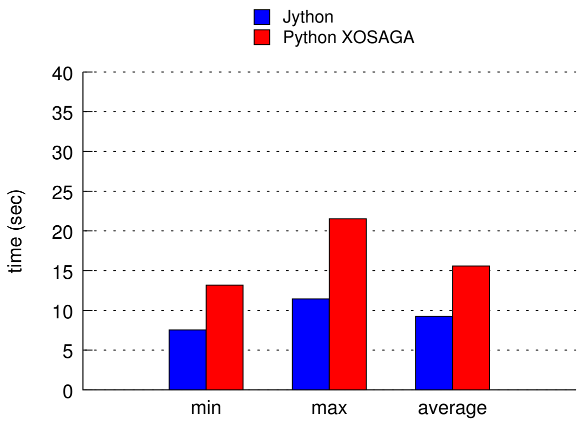


Figure 4.27: Comparing the completion time of namespace operations for Python XOSAGA and native Jython via the XtreemFS client.

Figure 4.28 compares file I/O operations on a locally mounted XtreemFS volume for Python XOSAGA and native Jython via the XtreemFS client. In average, XOSAGA is about 7 seconds faster than native Jython, indicating some inefficient file I/O in Jython. After further investigation, it turns out that Jython's implementation of the method `shutil.copy(from, to)` for copying a file is about twice as slow as the XOSAGA equivalent `Directory.copy(from, to)`. Jython performs marginally faster in reading and writing files, but its overall performance suffers from the slow file copy implementation.

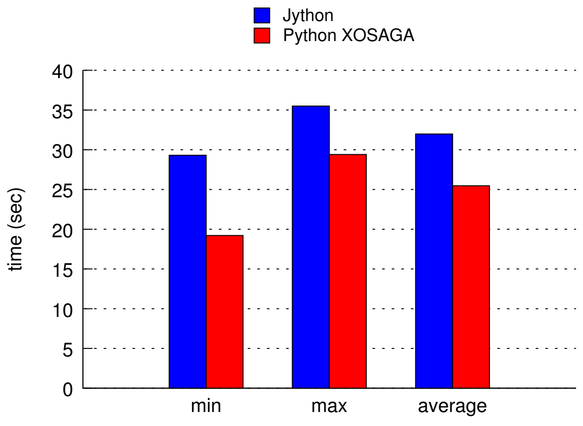


Figure 4.28: Comparing the file I/O operations for Python XOSAGA and native Jython via the XtreemFS client.

4.5.5 Test Unit 04: Java XOSAGA – Applications

4.5.5.1 Responsibilities

WP4.2, Enric Tejedor from BSC.

4.5.5.2 Test Specification

Test Items

The test items include:

- SAGA for Java
- test applications downloadable from: `svn://scm.gforge.inria.fr/svn/xtreemos/grid/xosaga/java/trunk/test/src/test/xosaga/job/JobAdaptorTest.java`

Features to be Tested

We test the following features of the Java SAGA API:

- Job submission
- Job monitoring (callbacks)

Approach Refinements

The objective of this test is to check the correct execution of a simple job using the Java SAGA API. The test is performed with the help of a simple application that can be found in the Java SAGA distribution.

The application that has the following parameters:

- Input: the resource where to submit the job.
- Output: state transitions of the job (console prints), output file with the name of the resource that runs the job.

The application performs the following steps:

- Creation of a job: the parameters are the /bin/hostname executable, number of processes = 1, and the output and input files.
- Registration for the callbacks at each job state transition.
- Submission of the job to the user-specified resource.
- Wait for the completion of the job.

4.5.5.3 Test Results

Results for each step of the test application:

- Job creation, callback registration and job submission without error.
- Callback reception correct, finishing in a DONE state.
- Job completion without error, with the output file filled with the name of the host where the job has run.

4.5.6 Test Summary Report

4.5.6.1 Summary of Tests and Results

The XtreemOS API (XOSAGA) has been implemented in three programming languages: C++, Java, and Python. For each of these three implementations, their performance overhead (compared to directly using the underlying service interfaces) has been investigated. With the micro benchmarks for each of the implementations, we have shown that the performance overhead caused by using XOSAGA is negligible for job submission and modest for namespace operations and file I/O. In the case of Python, file I/O was even faster than the native counterpart.

4.5.6.2 Directions for Future Work

Further evaluations with XOSAGA will be done with the ported versions of the WP4.2 applications COMPSs (from BSC) and openTurns (from EDF). At the time of writing this deliverable the application-centric experiments performed by WP4.2 failed because of bugs and missing documentation of XOSAGA. With the ported applications it shall be possible to evaluate the usage of the interfaces and the scalability and performance overhead from the application perspective. The prospected tests should also include large-scale performance tests on Grid5000 and interoperability tests with other Grid solutions.

4.6 Evaluation of Distributed Servers

Distributed Servers provide location transparent networked services [39]. Clients connect to a single *distributed server address* for a service and may be moved transparently among multiple locations. Mobile IPv6 (MIPv6) route optimization [22] does the heavy lifting: all IPv6 connections from a client are atomically changed directly to each location, avoiding triangular routing. The distributed server address is simply an IPv6 [14] address. In the terminology of Distributed servers, a client first connects to a *contact node*. A client may then be transparently *handed off*—the server endpoint of all of the client's connections are transferred—to different servers for load-balancing or for client-specific processing. Distributed servers are described in Deliverables D3.2.2, D3.2.6 and D3.2.11 [47, 48, 49].

Please note that all evaluation being reported here are duplicates of information present in Deliverable D3.2.11 [49]. They are provided here for completeness.

4.6.1 Test Plan

4.6.1.1 Responsibilities

This performance evaluation is under the responsibility of VUA within WP3.2 (task T3.2.1).

4.6.1.2 Test Items

These tests evaluate the Distributed Servers package as of their version from December 1st 2009 including comparisons to earlier versions.

4.6.1.3 Features to be Tested

The Distributed Servers provides the *handoff* of a client as a basic action. Characteristics of this handoff are tested in the following sections. We first evaluate handoff latency as seen by the client then we estimate the throughput at the server for a handoff by measuring the CPU usage for a given handoff.

4.6.1.4 Features not to be Tested

Other aspects of the Distributed Servers package, such as initialization and group management of server nodes are not evaluated. We also consider evaluating policies for performing handoffs, such as for load balancing, beyond the scope of this document.

4.6.1.5 Overall Approach

Although the Distributed Servers package is not included in XtreemOS version 2.0, it is planned to be included in later releases. The functionality provided by

Distributed Servers is independent of other XtreemOS packages. We have thus tested Distributed Servers in isolation.

We evaluated DS on a testbed of four machines connected by a 100 Mb/s ethernet switch. The machines were identical with a single 1.5 GHz AMD Athlon CPU and 500 MB of memory. One machine was configured as the client. Two were configured as distributed servers (one contact node and one server node). The remaining one acted as the network's home agent. Note that there is only one home agent for both servers, but the servers are configured to be in a different (foreign, in MIPv6) network from the home agent. We evaluated the new implementation of DS for latency and throughput and have included measurements from the previous implementation for comparison. Distributed servers are a unique feature of XtreemOS so there is no other comparable system against which we can evaluate distributed servers. Although the testbed used for these measurements is not identical to that used for the evaluation of the previous version, the machines and network are of equivalent capability. For a complete evaluation of the previous version, please see [39]. For a discussion of the motivation for and development of the new version, please see [49].

4.6.2 Test Unit 01: Handoff Latency

4.6.2.1 Responsibilities

This performance evaluation is under the responsibility of VUA within WP3.2 (task T3.2.1).

4.6.2.2 Test Specification

We first evaluate the handoff latency seen by the client to determine the minimum observed disruption to service. By providing a lower bound, application developers can determine whether such disruptions are manageable for their particular cases. The times for the old and new systems are provided in Figure 4.29. For the new design, we use tcpdump to determine the time of the last and first packets arriving from the donor and receiver of the handoff, respectively. For comparison with the old system, we used the minimum times of the old system given when the socket is first drained of data (that is, all data buffered in the kernel is sent to the client) before performing the handoff. Our additional method *tcpcp_flush* provides this functionality in the new implementation instead of the fixed wait time used in the previous implementation.

Local handoff times are presented in Figure 4.29 when a host hands-off the connection to itself, which does not require route optimization with the client or communication with another server node. These numbers thus provide the network-independent overhead and are quite similar to the previous design of DS.

	LAN handoff	Host-local handoff
Old	18.4 ms	9.4 ms
New	620 ms	13.4 ms

Figure 4.29: Handoff latency of connection as seen by the client. Time given is between arrival of last packet from the server donating the handoff and the first packet from the server receiving. The Old values are from [39] (ignoring RTT for Route Optimization), while New values were obtained using the design described in this document.

4.6.2.3 Test Results

The results of our handoff latency tests are presented in Figure 4.29 above. The results show quite similar local handoff times, showing that our TCPCP port is competitive with earlier versions. However, the LAN handoff times of our new version is an order of magnitude slower. This discrepancy is largely due to the use of the Gecko library at the server end of the handoff. The Gecko library implementation initiates a new connection between the donor and accepter of a handoff before sending the handoff data. The accepter also sends an acknowledgment before continuing to send data to the client. These high-level coordinating events need further optimizations to be competitive with the low-level measurements done in [39].

4.6.3 Test Unit 02: Handoff Throughput

4.6.3.1 Responsibilities

This performance evaluation is under the responsibility of VUA within WP3.2 (task T3.2.1).

4.6.3.2 Test Specification

In addition to the client perceived handoff latency, we measured the cost of DS at the server side of the connection. To answer how many connections could the server handoff per second, we measured the CPU time spent at both user and kernel level on either receiving or donating a handoff over a LAN. The results given in Figure 4.30 show that our current design is fairly expensive. Note that the latency for donating or receiving is not given and may be longer depending on network latencies. Fig. 4.30 provides the times spent on our rather dated 1.5 GHz AMD Athlon CPU and can be expected to be lower on modern processors. Handoff throughput is estimated between 3-4 clients per second, and we believe this will also improve as we optimize the new implementation.

4.6.3.3 Test Results

The results of our throughput estimation tests are presented in Figure 4.30. Although these tests show few connections per second, we note these measurements

	Total Length	Avg. Throughput
Donate	264 ms	3.8 conn. / sec
Receive	384 ms	2.6 conn. / sec

Figure 4.30: Handoff (over LAN) CPU time and estimated throughput as seen by the server in an application. The time given is the CPU time spent in both the system and user levels, not real-time. Donate and receive times are time spent by Accept and Handoff functions, respectively, in the Gecko library.

use the Gecko library, as in the previous latency evaluation. In this library, handoffs require an extra connection setup and teardown. The additional work reduces throughput significantly from that of a local handoff. Again, we believe this will improve as we adjust and optimize the Gecko library.

4.6.4 Test Summary Report

4.6.4.1 Summary of Tests and Results

Distributed servers are a unique feature of XtreemOS so there is no comparable other system against which we can evaluate them. We compared two implementations of distributed servers: an old, non-portable kernel-based version, and the new portable user-level version. The user-level implementation is demonstrated to work well, although it is in certain cases significantly slower than the old version. However, no particular effort has been spent on optimizing it yet, so important performance gains can be expected in future versions.

4.6.4.2 Conclusion and Directions for Future Work

In the remaining months of the projects our efforts will be directed to three main topics:

- Performance improvements of the user-level implementation of distributed servers
- Packaging and full integration within XtreemOS
- Building one or more demo applications

4.7 Evaluation of Virtual Nodes

Virtual Nodes is a replication framework for Java services that has been developed within WP3.2 by Ulm University (ULM). All evaluations performed are also contained in Deliverable D3.2.14 [12], so that we will only present a summary in this document. The evaluations have been performed as part of a master thesis that also presents a theoretical model of Virtual Nodes based on queuing theory and a comparison of the practical evaluation with respect to this model [35].

In the following we present the results of the three classes of evaluations. We first show the overhead of a single method invocation using Virtual Nodes in comparison to Java RMI and prove that our Java-RMI-compatible middleware layer works. Second, we compare the performance of different replication protocols provided by Virtual Nodes varying different system parameters such as state size and workload. This test also proves that those protocols are working. Finally, we measure the effect of node failures on service availability showing that our system can handle failing and joining nodes.

4.7.1 Test Plan

4.7.1.1 Responsibilities

WP3.2, ULM

4.7.1.2 Test Items

Virtual Nodes 0.2.2
Virtual Nodes 0.3.0

4.7.1.3 Features to be Tested

- Java RMI-compatible middleware layer
- Implementation for passive replication
- Implementation for active replication
- Shutdown and restart of replicas

4.7.1.4 Features not to be Tested

- Dixi integration
- Protocol switch
- Parallel Joins
- Network partitions

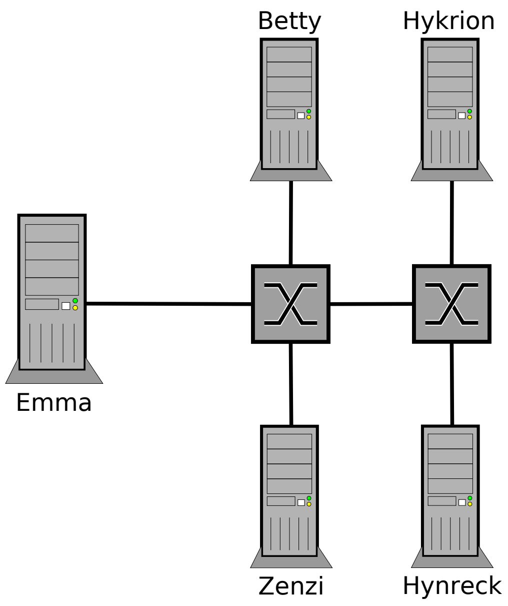


Figure 4.31: Evaluation setup

4.7.1.5 Overall Approach

For our test scenarios we used a distributed set-up with up to five nodes shown in Figure 4.31. We configured Virtual Nodes to use Jgroups 2.6.8[1] as a group communication system. Using Jgroups's SEQUENCER layer provides the required total order property for all messages sent to the replica group. In order to obtain comparable evaluation results we kept an identical startup order for all tests, so that the sequence of sequencers used by Jgroups would always be identical. Unless mentioned, all machines ran Sun's Java RTE 1.6.16. Please refer to [12, 35] for more details about the hardware configuration of the individual machines and the network connecting them. For all tests, the client was configured to wait for replies of all correctly working replicas. For both active and passive replication the client request is sent to one so-called *contact replica* and processed there according to the replication protocol. For active replication the contact replica is chosen in a random manner whereas in passive replication the contact replica is always the leader replica. In order to allow a meaningful comparison between active and passive replication, we decided to use sequential scheduling for active replication. Sequential execution also is the only possible scheduling option for passive replication. Unless mentioned, client requests were issued concurrently.

[1] www.jgroups.org

RMI	$2,941$ ms	0.5882 ms per request
Virtual Nodes	$21,096$ ms	4.2192 ms per request

Table 4.9: Java RMI vs Virtual Nodes Invocation Time per Method Invocation

When discussing the evaluation we will use hykrion as a single replica, hykrion and zenzi as the group of two replicas, and hykrion, zenzi and hynreck as three replicas.

4.7.2 Test Unit 01: Comparison to Java RMI

4.7.2.1 Responsibilities

WP3.2, ULM

4.7.2.2 Test Specification

Test Items
Virtual Nodes 0.2.2 shipped with XtreemOS2.0. User documentation is available in the XtreemOS user guide. Additional information on how to use Virtual Nodes can be obtained from Deliverable [11].

Features to be Tested

- Java RMI-compatible middleware layer

Approach Refinements
For the comparions of a single method invocation of Virtual Nodes and Java RMI we had a client application and a server application both running on different hosts. For Java RMI, the server application hosted a single RMI object that was exported via a Java RMI registry running in the same process. For Virtual Nodes the server application hosted a single replica, that was also exported via a Java RMI registry running in the same process. Both server applications provided the same interface to the client so that the client application did not have to be changed for accessing the servers. In both machines we used Sun's SDK 1.6.12 to perform the tests.

For the comparison of Java RMI and Virtual Nodes we measured the time it takes to invoke the remote service 5000 times, not counting another 5000 warm up invocations we did in order to rule out effects of just-in-time compilation.

4.7.2.3 Test Results

As shown in Table 4.9 the overhead of using Virtual Nodes is almost one order of magnitude. This is due to multiple reasons. First of all, the RMI protocol is highly optimised. A single TCP connection is kept open all the time and requests are multiplexed over this connection. Virtual Nodes, in contrast have no optimisation integrated so far. That is, for each request a new TCP connection is opened and closed again. Second, Virtual Nodes code contains logging operations which, even though not printed, require a significant amount on operations on `String` objects which are known to be highly inefficient. Finally, the protocol stack used for RMI is smaller than the one for Virtual Nodes. This is due to the fact that requests in RMI can simply be forwarded to the service implementation whereas in Virtual Nodes the request has to pass through an additional layer containing replication logic.

Even though one order of magnitude of overhead seems to be a high price to pay for using Virtual Nodes we want to stress that this factor of 10 may be reduced to probably $2-3$ and furthermore, the overhead introduced by group communication in multi-replica set-ups is much higher than for client-replica communication.

4.7.3 Test Unit 02: Comparison of Replication Protocols

4.7.3.1 Responsibilities

WP3.2, ULM

4.7.3.2 Test Specification

Test Items
Virtual Nodes 0.3.0. The version can be retrieved from the current development trunk in the XtreemOS repository. Due to its development status, no user or installation guides are available.

Features to be Tested

- Implementation of protocol for passive replication
- Implementation of protocol for active replication

Approach Refinements
In active replication requests are executed by all replicas. In passive replication only a single (primary) replica executes a request and informs all other (follower) replicas about the state changes in application state caused by the request. Due to their different approaches to fault-tolerance both strategies provide different properties. For instance active replication has smaller delays after replica crashes whereas passive replication is less demanding with regards to determinism.

For the sake of brevity, we only show results for a utilisation of 10%, 50%, and 90% and replica groups of one, two, or three replicas. Furthermore, we restrict our results to the round trip delay time experienced by the client and do not present

Utilisation	**1 Replica**	**2 Replicas**	**3 Replicas**
0.1	14.50±11.54	29.69±23.85	31.04±22.62
0.5	104.46±101.55	114.09±102.06	114.16±101.74
0.9	1091.40±1092.98	1106.54±1093.97	1107.63±1085.37

Table 4.10: Evaluation Results for Active Replication

Utilisation	**1 Replica**	**2 Replicas**	**3 Replicas**
0.1	18.84±15.37	26.28±22.64	38.55±25.87
0.5	112.78±106.50	125.18±110.79	133.04±112.99
0.9	1302.43±1138.11	1338.35±1151.07	1392.85±1156.78

Table 4.11: Evaluation Results for Passive Replication and Application State 0MB

results for individual phases of request processing. Refer to [35] for such detailed information.

For active replication we performed an evaluation over several group sizes and utilisation factors. As the performance of active replication does not depend on the state size of the application, our application used for evaluation had a size of 0 MB. For passive replication we run the same tests with identical input parameters. However, we did one run with an application state size of 0MB and another with .25 MB.

4.7.3.3 Test Results

The results for active replication are presented in Table 4.10. They show that the response time heavily depends on the utilisation factor whereas the number of replicas seems to play a less important role; the higher utilisation, the longer the execution takes. For a fixed utilisation going from one replica to two replicas adds another overhead of $10 - 15$ms whereas adding a third replica barely affects the does total time. This effect is caused by the fact that in comparison to hynreck which is the third replica added zenzi is so slow that hynreck's reply is almost always faster than zenzi's. The additional costs caused by reaching total order for three instead of two group members is compensated by the fact that now zenzi is only contact node for $\frac{1}{3}$ of all requests, wheras before, it had to process half of them.

The results for passive replication with an application size of 0MB are summarised in Table 4.11. Just as for active replication, it shows that execution takes longer the higher the utilisation is. Furthermore, adding new replicas increases the time it takes to process a request. In comparison to active replication, passive replication is clearly slower. It takes longer than active replication as the level of concurrency allowed is smaller for passive replication. While active replication

Utilisation	1 Replica	2 Replicas	3 Replicas
0.1	23.98±24.54	33.21±25.00	54.73±30.03
0.5	122.27±112.55	134.45±114.96	171.83±121.02
0.9	1651.32±1334.88	1775.22±1370.86	1912.17±1394.03

Table 4.12: Evaluation Results for Passive Replication and Application State .25MB

allows making the broadcast and collecting replies for different clients in parallel, passive replication requires that execution for successive requests be blocked until the state has been serialised. The evaluation for passive replication with a state size of .25MB shown in Table 4.12 underlines the previous results.

It turned out that evaluations beyond a state size of .25 MB was impossible for larger utilisations. This is due to the fact that for passive replication a full state snapshot is serialised and sent to the backup replicas. This delays the execution of subsequent requests so that more and more connections from the client are opened leading to a `TooManyOpenFilesError`.

4.7.4 Test Unit 03: Effects of Node Failures

4.7.4.1 Responsibilities

WP3.2, ULM

4.7.4.2 Test Specification

Test Items
Virtual Nodes 0.3.0. The version can be retrieved from the current development trunk in the XtreemOS repository. Due to its development status, no user or installation guides are available.

Features to be Tested

- Shutdown and restart of replicas

Approach Refinements
For scenarions that include node failures we evaluated utilisation ratios of .25 and .75 for application state sizes of 0MB and .25MB. We used an average failure of 1 : 500 per replica. That means, on average after 500 requests each replica has failed once; or after all 5000 requests each replica has failed about 10 times. The interval between failures and recovery is set to 6 seconds which is the default time JGroups requires to recognise a node failure. Instead of execution time, we measured service availability, that is the ratio of successfully processed request to

Utilisation/State Size	**1 Replica**	**2 Replicas**	**3 Replicas**
0.25/0MB	.845	.977	.990
0.75/0MB	.835	.972	.950
0.25/.25MB	.845	.962	—
0.75/.25MB	.824	.877	—

Table 4.13: Availability for Active Replication with Node Failures

Utilisation/State Size	**1 Replica**	**2 Replicas**	**3 Replicas**
0.25/0MB	.847	.952	.978
0.75/0MB	.846	.900	.962
0.25/.25MB	.846	.913	.946
0.75/.25MB	.850	.900	.972

Table 4.14: Availability for Passive Replication with Node Failures

the total number of requests sent to the service. For shutting down a replica we used the `shutdownReplica` method that comes with each Virtual Node.

We assumed a stable and fault-tolerant registry, where replicas insert the most up-to-date version of the contact addresses of the entire replica group. The client, before executing a request, fetches the latest information from this repository and uses this contact information for invoking the service. Even though this execution model requires a third entity that provides additional fault-tolerance, we think it is valid to use it. This is mainly because of the fact that the currently ongoing integration of Virtual Nodes in the Application Execution Manager uses the JobDirectory service to provide a last ressort for contact addresses of replicas. In consequence, a Virtual Node is only unavailable if all of its replicas have failed.

4.7.4.3 Test Results

The results for availability are shown in Tables Tables 4.13 and 4.14. Neither for active nor passive replication it is possible to make a statement on whether or not state size has an impact on availability. In contrast, however, the number of replicas clearly has an impact on service availability. The more replicas we used the more available the service was. The only exception to that rule is in active replication where two replicas provide better availability that three replicas when the state size is 0MB. So far, we have not been able to track down this anomaly, so that further evaluation is required.

4.7.5 Test Summary Report

In this section we have provided an evaluation of the Virtual Nodes replication framework regarding performance and availability.

4.7.5.1 Summary of Tests and Results

We have shown that our system is about one order of magnitude slower than Java RMI. Furthermore, we have presented that the runtime overhead of passive replication as we implemented it, is slightly higher than for active replication. This gap might turn out to be more serious in case deterministic schedulers are used for active replication. Finally, our evaluation did show that an application's state size barely has an impact on service availability. Yet, for more replicas the availability increases drastically.

4.7.5.2 Conclusion and Directions for Future Work

Evaluation has given us some hints on what to improve on our system. First of all multiplexing of connections or at least re-using of open sockets would be beneficial to the overall system performance.

Second, overload renders the system unusable or may even lead to the failure of individual replicas or the entire Virtual Node. This makes the system vulnerable for denial-of-service attacks. Thus, adding mechanisms that are able to handle overload and support graceful degradation in such situations are desirable extension.

Third, it is not clear which set-up is the best alternative in case of node failure. Here, further investigation and evaluation is required.

4.8 Evaluation of Application Execution Management

4.8.1 Test Plan

4.8.1.1 Responsibilities

SAP is responsible for the test plan evaluations performed by WP4.2.

Ramon Nou (BSC) is responsible for the evaluations performed by WP3.3.

4.8.1.2 Test Items

The test items related with this evaluation are the following:

- the release of AEM component that is shipped with the XtreemOS release 2.0.
- The hmmpfam application using COMPSs ported to the AEM XATI API.
- The documentation and distribution of SPECweb2005 application are available at http://www.spec.org/web2005.
- The Moderato application.
- The documentation and source of Globus Toolkit 4 may be found at http://www.globus.org/toolkit.

4.8.1.3 Features to be Tested

We tested the following features of the AEM XATI API:

- Job management: submission, state checking.
- Resource management: discovery, reservation.

Furthermore, we present a comparison of the aforementioned features of AEM with the Globus middleware, as well as measuring the scalability of AEM.

4.8.1.4 Features not to be Tested

We test all the features offered by AEM components.

4.8.1.5 Overall Approach

To evaluate the AEM, we used the AEM *job submission*, *job monitoring* and *job reservation* features. The execution of the tested applications generally begins with the initial discovery and reservation of the needed resources to execute the applications themselves. Once the reservation is done, the applications submit jobs to the previously reserved resources and check their state for completion. Finally, when all the jobs are finished, the applications release the resources.

We also obtained basic results of AEM features to compare with other offerings, i.e. Globus. In addition, we have performed scalability tests with the hardware available to us. Globus is a standard Grid middleware implementation and makes for a worthwhile comparison. We have compared the submission, execution and job status query on one node, as this is allows us to perform our evaluations without other components.

On the mobile device side, a test is included that compares the computation power of the Grid based on the XtreemOS version that is invoked through AEM API: XtreemOS PC flavour or XtreemOS MD flavour. The goal of this test is to observe the performance of the AEM API as it is invoked from a XtreemOS-PC client or from a XtreemOS-MD client.

4.8.2 Test Unit 01: COMPSs

4.8.2.1 Responsibilities

WP4.2, Enric Tejedor from BSC. The tests included here are partially submitted in [31].

4.8.2.2 Test Specification

Test Items

We tested the following features of the Application Execution Management (AEM) component:

- Job submission and monitoring
- AEM resource reservations

Features to be Tested

We tested the following features of the AEM XATI API:

- Job management: submission, state checking.
- Resource management: discovery, reservation.

Approach Refinements

The evaluation of the features enumerated above will be performed during the execution of the hmmpfam application, with COMPSs on top of XtreemOS.

When COMPSs parallelizes hmmpfam, it creates a task dependency graph based on the workflow of the application. The tasks will be submitted as jobs to the XtreemOS grid and periodically checked for completion on nodes running the application. In addition, COMPSs will perform an initial discovery of the available

Grid resources by means of the AEM XATI API and then it reserves a set of these resources nodes to act as workers running the jobs in the application workflow. At the end of the application, the COMPSs runtime will release the reservations.

The purpose of the tests is to verify the proper operation of the main AEM functionalities and analyze its performance.

4.8.2.3 Test Results

In order to evaluate COMPSs-hmmpfam on top of XtreemOS-AEM, we conducted tests to measure the execution time of the application. The testbed that we used is composed of twelve single core VMs running XOS Core and XOS resource nodes. This testbed is represented in Figure 4.32. For job execution, we used one VM as the master node (the one that hosted the runtime) and a variable number of worker VM nodes (from two to ten).

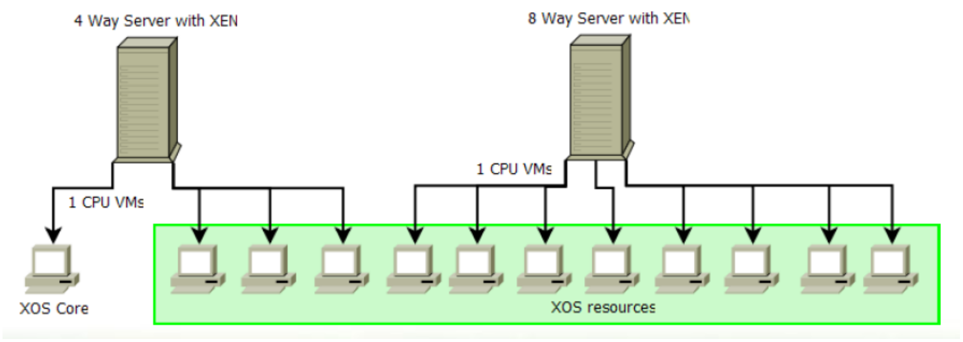


Figure 4.32: Testbed used in COMPSs-hmmpfam experiments.

In order to compare the XtreemOS performance with another Grid middleware implementation, we ran the same series of tests using two different configurations of the COMPSs runtime: first, the COMPSs runtime ported to XtreemOS, making use of the AEM API; second, the same runtime using the JavaGAT [1] interface and its SSH adaptor to submit jobs and transfer files. No distributed file system was used in either case. Rather we copied the application files from node to node when necessary, i.e. every time that a task needed a given input file, that file was SSH-copied to the destination node before the task execution. This way, we achieved a more direct comparison between AEM and SSH.

Figure 4.33 shows the performance and scalability results of running COMPSs-hmmpfam, both using XtreemOS and SSH. For each number of workers and configuration, we ran two tests and the average is plotted. There was no significant difference between executions of the same number of workers and configuration. We see how the results are quite similar for both configurations, which achieve good scalability results.

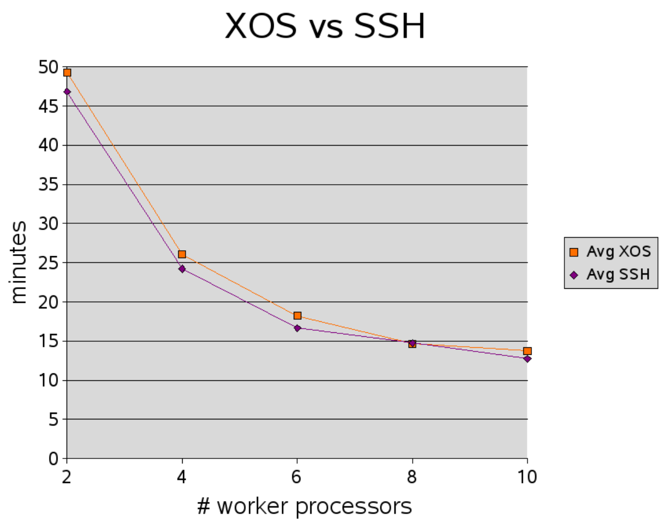


Figure 4.33: Execution times of the hmmpfam application on top of COMPSs, both with the XtreemOS and the SSH flavours.

4.8.3 Test Unit 02: SPECweb2005

4.8.3.1 Responsibilities

WP4.2, J. Oriol Fitó from Barcelona Supercomputing Center (BSC). The tests included here are partially submitted in [31].

4.8.3.2 Test Specification

Test Items

We tested the following features of the Application Execution Management (AEM) component:

- Job submission
- Job monitoring
- AEM resources reservations

Features to be Tested
The principal goal is the testing and evaluation of the AEM component of XtreemOS. Our focus during the evaluation is on the following features: job submission, monitoring and reservations.

Approach Refinements

In order to evaluate the AEM, we have decided to use its *job submission*, *job monitoring* and *job reservation* features. We deplyed the SPECweb2005 application to validate these features. Notice that we have decided not to change to the newest version, that is SPECweb2009 [5], because it's equal to the previous version except the inclusion of the power workload. The rest of the benchmarks in this sotware is exactly the same in both older and newer versions. In this case, the power workloads are not truly relevant for our purposes,so there are no advantages to using the newer version.

Our test scenario is comprised of a web server, a back-end database server simulator (i.e. BeSIM) and many client machines (see Figure 4.34).

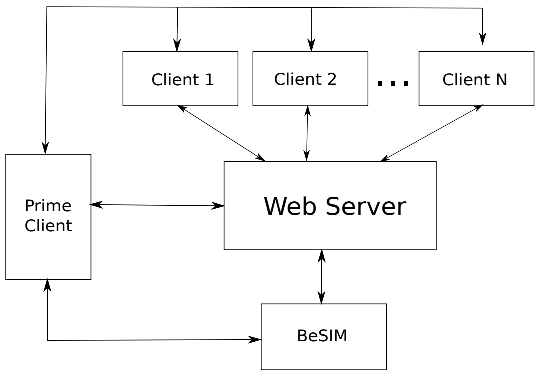


Figure 4.34: SPECweb2005 benchmark architecture.

In particular, we want to execute the benchmark by submitting the needed client jobs and processes by the application itself. This submission will be done by using a suitable reservation of an available resource node for the web server. In addition, all the needed clients will be encapsulated into a job, each one having its own processes inside. Then, when the application is launched, we will monitor it by using a simple monitoring web interface developed by BSC. Via the web interface, we are able to verify which nodes fulfill the requested resource requirements for example, the number of available CPUs or amount of memory or available disk space. The web interface also allows us to view prior job history, terminate running jobs, as well as perform other tasks.

The test procedures are the following:

1. Make a reservation for the web server and submit the job which will execute on it.

2. Make a reservation the BeSIM component of the benchmark and submit the job to be executed.

3. Make a reservation for a job which will encapsulate all the clients, each one as a process inside this job.

4. Make a reservation for the prime client, which is the component that initializes the environment, manages the benchmark execution, and collects the results.

Specifically, we repeatdly execute the benchmark in order to observe the server performance with varying input loads, expressed as simultaneous user sessions. These repeated executions will be performed by using one and two CPUs allocated to the resource reservation of the web server requested of the AEM. Through these experimentats we will be able to see how the performance of the web server scales up when we provide it with an increasing number of resources.

Finally, we check the output results of the application executions in order to validate the proper functioning (i.e. job submission, monitoring and reservations) of the XtreemOS component that is tested here, i.e. the AEM. In addition, we will show the web server's performance high-level metrics from the benchmark results: throughput (requests per second), response time (seconds) and Quality of Service.

4.8.3.3 Test Results

The testbed used for performing the experiments presented in this section is illustrated in Figure 4.32. In order to evaluate the AEM component of XtreemOS, we have repeated several executions of the SPECweb2005 benchmark [4] with different input loads on the web server being evaluated. Actually, for each input load (i.e. number of simultaneous user sessions), we have performed three execution repetitions with the aim of obtaining stable results regarding the performance of the web server. These tests were carried out using the appropriate number of client machines needed to emulate the desired input load, i.e. simultaneous user sessions. For this reason, we used horizontal scaling of the benchmark's client component, which involves testing this type of scalability around AEM reservations. To obtain these results, we used a range of one to nine client machines, acquired through the resource reservations as needed. The possibility to reserve efficiently as many resources as needed is a remarkable strength of XtreemOS. In addition, we present the vertical scalability results, in terms of the performance (i.e. throughput, response time and Quality of Service metrics) offered by the web server running on top of XtreemOS.

Web server's performance metrics In Figures 4.36, 4.37 and 4.38 you can see a graphical representation of the results regarding to the performance of the server. Note that all of these results are expressed depending on the number of simultaneous user sessions emulated by the clients in the benchmark. In addition, the benchmark used a time-based Quality of Service (QoS). This means that QoS is based on the amount of time that elapses between a web page request and the receipt of the complete web page, including any image files. In this sense, the output results of any benchmark execution can be assessed for "Aggregate QoS compliance". It consists of calculating the total amount of requests (expressed as a percentage of the requests) that are within each of three different ranges: *Good*, *Tolerable* and *Fail*. In particular, a configuration file of the benchmark establishes two response time thresholds, called *TIME_GOOD* and *TIME_TOLERABLE*, that are used to classify the amount of requests that are within each range: if the response time of a given request is less or equal than *TIME_GOOD*, then the request is classified within the Good range; if it is greater than *TIME_GOOD* and less or equal than *TIME_TOLERABLE*, the request is classified as Tolerable and, finally, if it is greater than *TIME_TOLERABLE* the request is classified as Fail (see Figure 4.35).

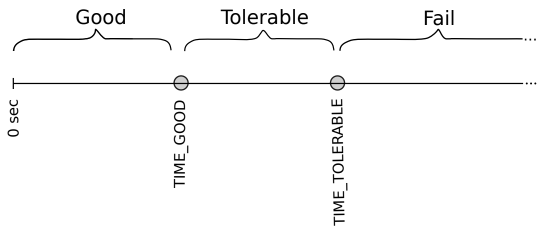


Figure 4.35: SPECweb2005 QoS criteria. TIME_GOOD and TIME_TOLERABLE which define the QoS ranges: Good, Tolerable and Fail.

These percentages give us a quantitative overview of what level of QoS has been reached during the execution of any test.

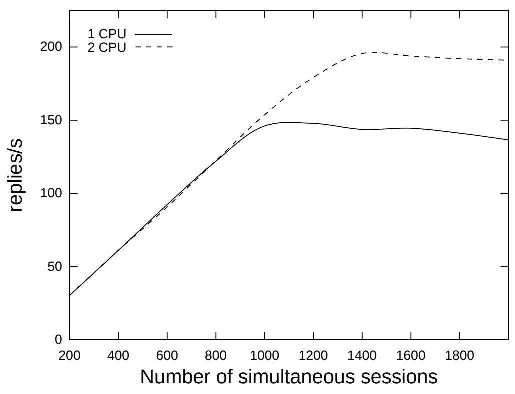


Figure 4.36: Banking throughput (requests per second) when running with one and two processors.

These tests show that XtreemOS is a suitable environment in two different ways, with regard to web servers:

- **Testing platform** This is a viewpoint around the setup process for the benchmark. Typically, a large number of clients are required to run an accurate test on a web server. Acquiring and setting up these disparate resources can

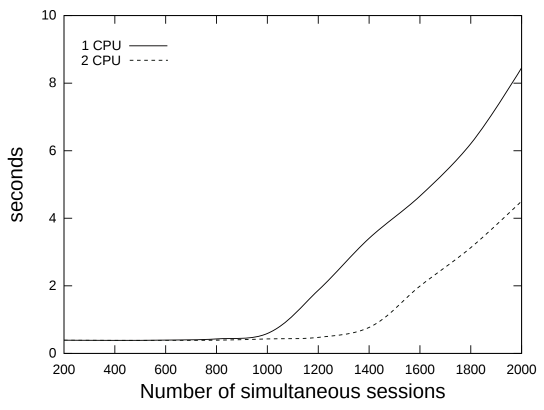


Figure 4.37: Banking response time (seconds) when running with one and two processors.

be very time consuming. Thus having easy access to remote resources, as in XtreemOS, will no doubt greatly help in this task. Actually, through the tests performed, we check that we are able to execute tests on he web server with a large variety of capabilities examined. Thus we are able to acquire the required resources (through XATI AEM) suficient client machines to emulate the input load, i.e. the number of simultaneous user sessions.

- **Evaluation platform** The distributed nature of XtreemOS expands the range of possibilities in terms of scalability and robustness for a service hosted on the said OS. Therefore. even when taking into account the vertical scalability results presented above, we verify that XtreemOS is a proper environment for hosting web servers with their required services and deployed web applications. In fact, we are also able to replicate the web server services among several XtreemOS resources, i.e. perform a horizontal scalability of the web server. In this scenario, the performance capabilities of the pool of web servers will be multiplied by the number of web servers minus the overhead introduced by a load balancing mechanism.

For all these reasons, we think that we have successfully proven that XtreemOS is a suitable environment for hosting web services and applications.

4.8.4 Test Unit 03: Customizable SSI Scheduler

4.8.4.1 Responsibilities

WP4.2, Guillaume Alleon of EADS.

4.8.4.2 Test Specification

Test Items

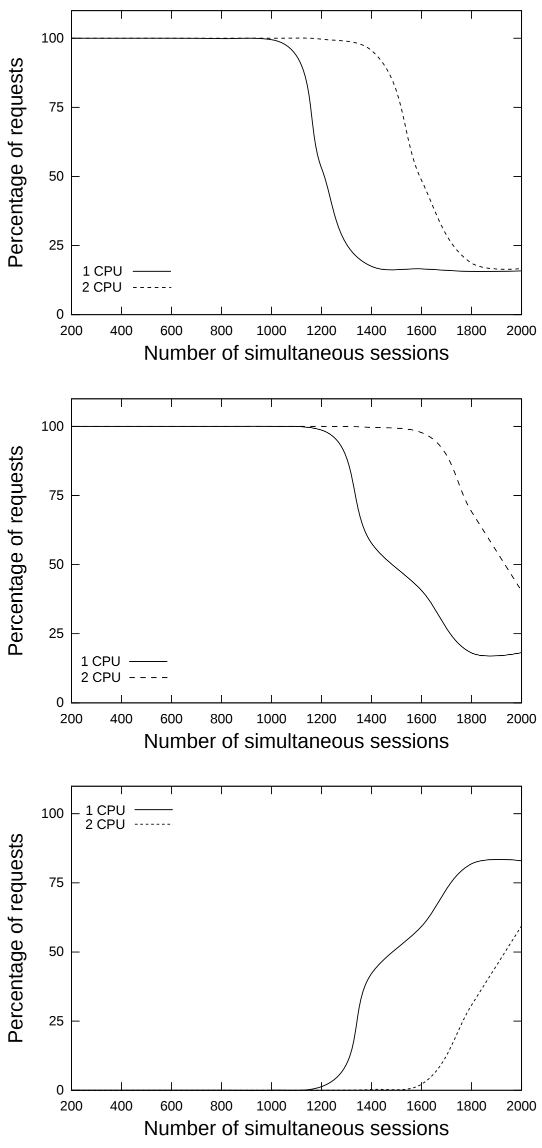


Figure 4.38: Banking “Aggregate Quality of Service (QoS) compliance” when running with one and two processors. Good, tolerable and fail ranges, from top to bootom.

For this test, EADS used the amibe software. Amibe is an open source software that can be retrieved from `http://jcae.sf.net`. Amibe is the pre-processor of the Elfipole Electromagnetics solver. It basically transforms a CAD (geometry) in a set of triangles (discretized geometry). Amibe itself is written in Java but does rely on a native library (OpenCascade). In order to provide a fair measure, we have used an ideal CAD that provide the same amount of work per processing unit. The CAD is made of a cube (box) made of 4056 patches which therefore allows to run up to 4056 processing units.

Features to be Tested

The main interest in this test is to evaluate the difference of performance between XtreemOS PC flavor and a grid running the Condor middleware. The feedback will be given for each of the module of the amibe software.

Approach Refinements

The meshing as handled by Amibe is a three steps process:

1. the meshing of the edges also known as 1d mesh [jcae1d],
2. the meshing of the patches also known as 2d meshing [jcae2d],
3. the assembly of the different patches in a global mesh also known as 3d meshing [jcae3d].

The first part is fast and sequential, it ends up in a splitting of each and every edge in segments of similar size. The results are stored in a single directory. The second step is parallel since the code iterates over the patches. Each patch can be processed independently and depends on the results of the first phase. Each result is stored in a separate directory. The last phase aggregates the results of the previous phase eliminating the redundant points (located on the edges). It is built upon all the results of the step 2 runs.

As result, the measure of those different phases are kept and compared for both XtreemOS and Condor. The runs are performed for both configurations on Grid5000 resources. XtreemOS version 2 is used in its version provided by INRIA.

4.8.4.3 Test Results

The first test shows the performance of XtreemOS and Condor on 40 processors for a mesh size of 1. Both configurations generate the same meshes: 23 MB for the 1d mesh, 40 files and 98 MB for the 2d meshes and a final 3d mesh of size 5.3 GB. For XtreemOS, the total computing time is about 33 minutes while for Condor it is 36 minutes. In both cases, the computing times are roughly the same i.e. 30 seconds for the 1d phase, about 11 minutes for the 2d phases and 20 minutes for the 3d phase. The main difference is therefore in the management time by each of the middleware. For XtreemOS, this time is about 1 minutes and 30 seconds while it is roughly 5 minutes for Condor.

For a mesh size of 0.5, the time for the 1d phase remains the same while, the 2d phase jumps to 43 minutes and the 3d phase goes up to 86 minutes. The total execution time is respectively 130 minutes and 133 minutes. Therefore the difference in management by the different middleware remains independant of the mesh size.

4.8.5 Test Unit 05: Moderato

4.8.5.1 Responsibilities

WP4.2, Samuel Kortas from EDF.

4.8.5.2 Test Specification

Features to be Tested

We test the following features of the AEM:

- Job management: submission, state checking, cancellation.
- Resource management: discovery, reservation.

Approach Refinements

In order to make the evaluation of the AEM, we use its job submission, monitoring and reservations features. The Moderato Application launches a large number (up to several hundreds of thousands) of independent runs describing extensively a given set of parameters. A typical case will use

- AEM reservation and discovery service to gather information on the resource available in XtreemOS that can be used to execute these runs.
- AEM submission, monitoring and cancellation service to manage the runs

Correctness of obtained results will be checked in the end. The results will be dumped either on shared XtreemFS, or on each local file system of participating nodes and transfered back at the end of the execution.

Using the mockup of Moderato, we test the ability of XtreemOS to support a high number of jobs and to load-balance them well on the available nodes. The test input parameters are the number of submitted jobs. Jobs are gathered in chunks, whose size is another varying input. Time of a given calculation is plotted with respect to the number of resource nodes used.

Test were run on dual four-core processor Hewlett-Packard Desktop box running natively XOS version official 2.0.

4.8.5.3 Test Results

In order to test the robustness of the AEM scheduler, we used MODERATO to submit hundreds of one-processor jobs to the XtreemOS grid at the rate of 30 jobs per minute.

As for now, December $20^t h$, we observe that only a part of the jobs submitted reaches the “Done” final Status, others stays either in a “Running” or “LocalSubmitted” Status.

So far, this problem has not been solved yet and is under investigation. We also noted that it doesn't occurs when we submit a lighter number of jobs (typically less than 15).

4.8.6 Test Unit 06: AEM vs. Globus Toolkit

4.8.6.1 Responsibilities

WP3.3, Ramon Nou from BSC. The tests included here are partially submitted in [31].

4.8.6.2 Test Specification

Test Items

For this test BSC used a set of bash scripts that execute a time measuring the time to do a xsub (normal xcommand included in XtreemOS Release). For the query, xps is called.

Features to be Tested
We test the following features of the AEM interface:

1. Single job submission.

2. Job status query.

In this test, we compare the results with the same test in Globus Toolkit 4.2.

Approach Refinements

The evaluation of the features enumerated will be performed using the same testbed for the two installations (Globus and AEM). The test will use synthetic benchmarks. For the first feature, we submit a large number of jobs (1000) that execute /bin/true. We wait until the job finishes and get the time that the job uses from submission to finalization. Between every submission we wait one second. There is no saturation of the middleware in this test.

The last feature uses a submission of a very large job (sleep) that will run during the whole test. Then a job status query (1000 queries) is issued via the command line interface. We record the time it takes to get an answer and wait 1 second between queries. We ensure that the job is running by always recording the query result.

We present all results with a c.i. of 95% of the 1000 repetitions showing stable measures.

4.8.6.3 Test Results

The testbed in Figure 4.39 that we used is composed of two 2-way VM running a XOS Core and a XOS Client. Another installation is setup on a 2-way VM with a Globus Toolkit (core) and a client. The hardware is using a Core 2 Quad Intel Processor (4 cores).

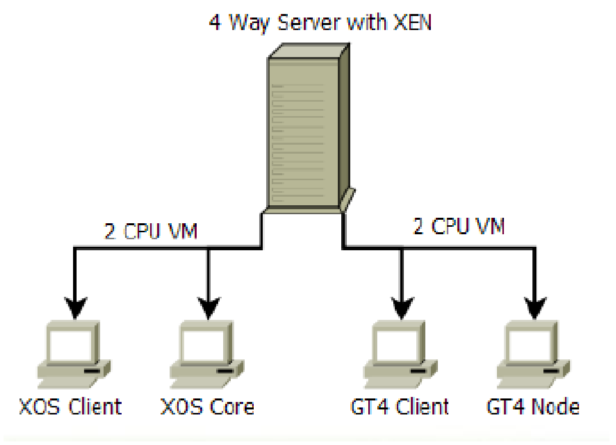


Figure 4.39: Testbed used in Globus vs AEM experiments.

First test, Figure 4.40, compares Globus and XtreemOS submission and execution. The test does not use any feature not found in Globus, so we are using basically NSS-PAM and AEM; XtreemOS is not using resource location services or XtreemFS. The test submits a job that executes `/bin/true` in one node. With this kind of job we are measuring the performance of the middleware code.

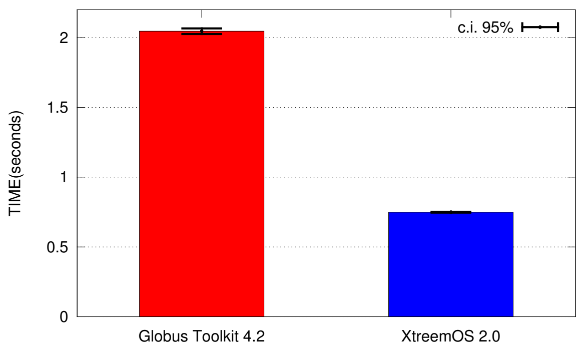


Figure 4.40: Job Submission and execution on environment 1. One core, 1000 repetitions, /bin/true job.

The second test compares Globus job status (obtained from GRAM4) and XtreemOS job status obtained from AEM. AEM job status includes the process status. The test is measured on testbed 1 and consists of a large job (sleep) while calling job status utilities. The results are shown in table 4.15.

The results obtained in the job submission and execution shows a great improvement over Globus in this basic unit.

Table 4.15: Performance of job Status in environment 1 with confidence interval of 95%.

	Globus Toolkit 4.2	XtreemOS
Mean (s)	0.2842714	0.04654
c.i. 95%	[0.2832860,0.2852567]	[0.04527204,0.04780796]

In conclusion, we can see the basic set of actions to submit and execute a job faster than Globus with the same hardware, However, as we increase the set of features in XtreemOS (reservations, different scheduling, enhanced security, XtreemFS mount) excecution time can increase. The results also show how we are apparently more stable in the results than Globus. Globus typically spends a lot of time in SOAP processing [32] and unlike AEM, the middleware is not integrated into the operating system. This makes processes like job status query more time consuming for Globus.

4.8.7 Test Unit 07: AEM scalability

4.8.7.1 Responsibilities

WP3.3, Ramon Nou from BSC. The tests included here are partially submitted in [31].

4.8.7.2 Test Specification

Test Items

For this test BSC used a set of bash scripts that execute a `time` command measuring the time to perform a xreservation (when we have manual reservations), xsub, and xps. All the xcommands are included on XtreemOS. The bash scripts loops until all spawned processes are created via xsub -j. For the job status query, a simpler script calling xps (several times to get noticable times by “time“) and measuring the time is used.

Features to be Tested

We test the following features of the AEM interface:

1. Single job submission with multiple processes spawned over different nodes.
2. Different reservation methods used (automatic or manual with different number of resources).
3. Job status query in multiple nodes submission.

Approach Refinements

The evaluation of the features enumerated will be performed using a 12 VM installation over two real nodes. The test will use synthetic benchmarks. For the first feature (1 and 2) we submit a large number of jobs (100 each figure point) that execute a sleeping process. A job is created, executed, and then a number of n sleeping processes are spawned sequentially. We wait until the last process is spawned. Between every job submission we wait one second deleting the reservation and cancelling the job.

The reservation can be automatic or manual, with a fixed number of resources or a dynamic value. We record the time used from the creation of the reservation (user call) to the submission of the last process.

The last feature (3) uses a submission of a similar set of jobs and processes (using n resources) and send, via the command line interface, a set of job status query (30 sets of 30 queries, to get measurable times as queries are small). We plot the time we need to get an answer.

We present the results using a c.i. of 95%. We do not compare with GT as Globus does not support these features.

4.8.7.3 Test Results

The testbed that we used is composed of twelve single core VMs running XOS Core and XOS resource nodes. This testbed is represented in Figure 4.41.

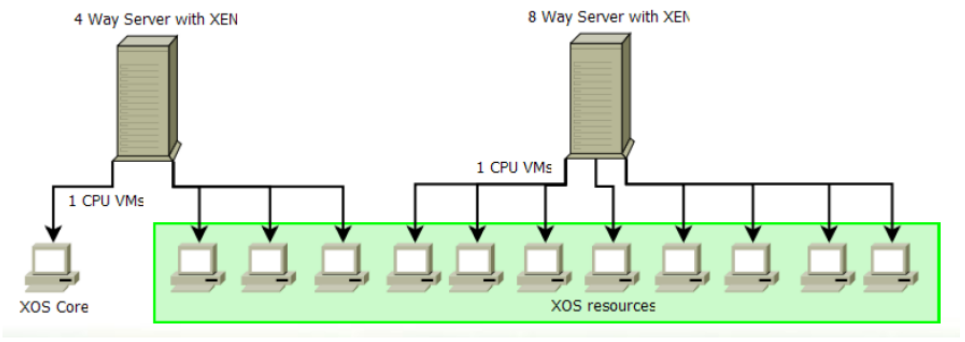


Figure 4.41: Testbed used in AEM scalability experiments.

For comparison, we present the results in Table 4.16 of a Globus job submission in this test environment.

First test (Testing features 1 and 2), shown in Figure 4.42, compares scalability related to job submission. In XtreemOS, we can use the advance reservation scheme (manual reservation) or an automatic reservation system based on the JSDL requirements. In this test, we have automatic reservations for a fixed number of resources, as 5 and 12 resources while submitting n processes to them. We also test the reservation of n resources submitting and executing n processes inside them.

Table 4.16: Summary of Job Submission (no execution) in GT4.2 in one node (Test Feature 1 and 2).

	Globus Toolkit 4.2
Mean (s)	2.596
c.i. 95%	[2.488,2.704]

Automatic reservations are faster as the reservation is managed inside the XOSD. Other test lines are manual reservations where the user interacts with the system to reserve in advance n resources and send n processes to the system. The time for this test includes the whole stack: reservation and job submission. In table 4.17 we summarize the different formulae that follow the lines.

Table 4.17: Environment 2. Submission behaviour. (Test Feature 1 and 2)

Submission Scenario	Formulae
Automatic (12 Nodes)	$0.7067n + 1.1728$
Automatic (5 Nodes)	$0.3560n + 0.9140$
Automatic (n Nodes)	$0.623886 * n * log(n) + 0.411018$
Manual Reservation (n Nodes)	$0.7168n + 5.5239$

In the Figure 4.42, we have the resource finding and reservation overhead, but we remain below Globus values in the same environment (Table 4.16), but with higher capabilities (dynamic resource discovery in XtreemOS). To reproduce the same scenario in Globus we will need a Job queue system like Condor or to send multijobs JSDL (this will introduce external players, out of the scope of the test). Execution time with fixed reservations are following a linear regression and measurements are stable. In the case of the Automatic reservations asking for n resources where n are the number of job and processes going to submit the result is $O(n * log(n))$ for the number of resources. We can reduce this time doing the submission in parallel, also changing the scheduler to another one (we are using a RR scheduler that makes a sort) can reduce (or increase if we are doing more tasks to select a suitable node) this value.

These may change if many users are sending jobs through the same JobMng or XOSD, but requests will be redirected to other JobMng to maintain the a responsive service level. The same test is repeated, but using manual resource reservations (measured inside the submission time, including removal of the reservation after the submission), the result is similar in scalability terms. Performance is lower as we are performing each step using separate processes which implies connecting to the XOSD. We can see how the performance is proportional to the number of resources requested (for that specific job) as the DHT needs to look for the resources, match the JSDL and finally JobMng needs to guarantee a reservation for a specified period. Future work on the different DHT overlays (concretely RSS) will reduce this time via reduced network transactions. As a summary, scalability has a

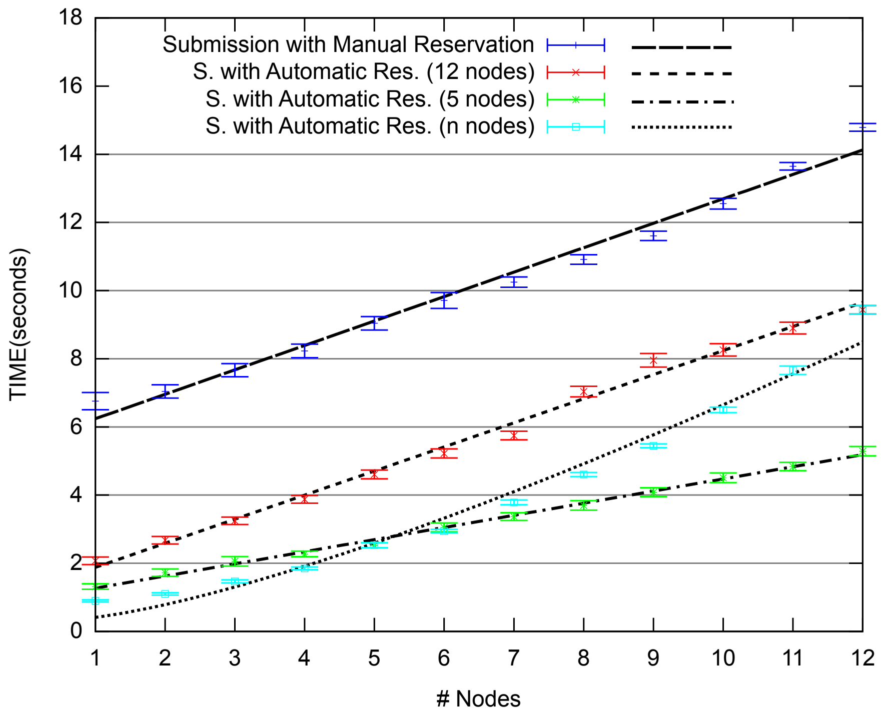


Figure 4.42: Job Submission scalability on environment 2. 12 Nodes, 100 repetitions for test. Without automatic reservations (requesting 5 nodes and 12 nodes, and n nodes) and with manual reservations for n nodes. (Test Feature 1 and 2)

slope of 0.7067 when requesting 12 resources (although only 1 is used) and 0.3560 when we request 5 resources (in Table 4.17 there is a summary of the formulae that follow the submission behaviour). Those times are based on local network times and are bound to network latencies. Also $O(n * log(n))$ is not exactly perfect as long as Java sort is switching between insertion sort and merge sort when we are in a low number of nodes.

The second test of scalability explores the scalability of job status using the second testbed (Test feature 3). We create n jobs with 12 processes distributed through the 12 nodes or n jobs with 12 processes distributed among 5 nodes. Then we check the job and process status of a job. The user request should go to the JobMng managing all job status and through all the nodes in the system (ExecMng) to get process status. The results are the mean of 30 job status queries repeated 30 times.

In Figure 4.43 we have a limitation on the number of jobs that can be run on each XOSD due to memory restrictions. We can see how the result does not depend on the number of jobs running in the system as the time is almost constant, and only depends on the number of nodes used for each job. We should remember that we can ask only job status (excluding process status) resulting in a constant time independent of the number of nodes running the job, as the JobMng does not need to contact the ExecMng. Additionally as ExecMng requests are done in parallel by the JobMng time asking the process status to all ExecMng is reduced. Checking job status in a saturated system (with a large number of users, processes or jobs) does not imply overhead and as far as we (or the system) can distribute the load between different Job Managers we remain with low load levels.

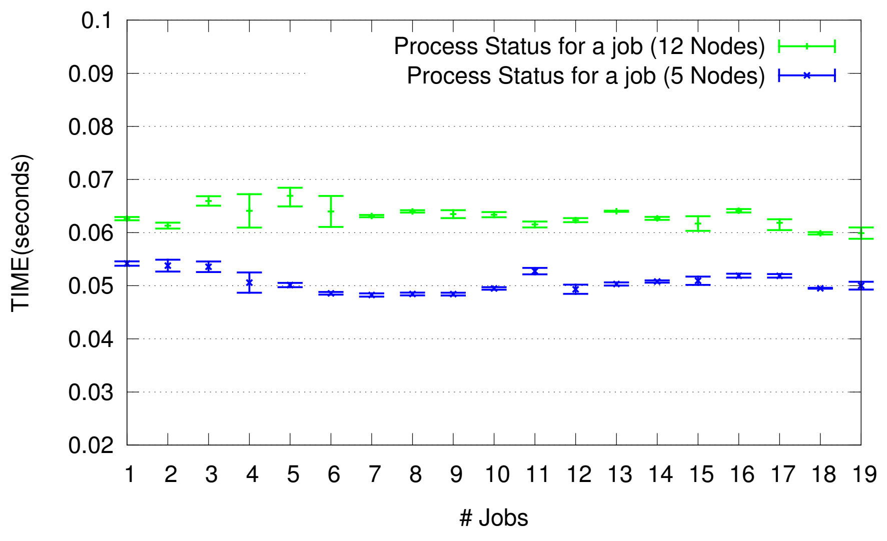


Figure 4.43: Scalability of a job Status when n jobs with 12 processes (using 12 nodes and 5 nodes) are in the system. (Test Feature 3)

4.8.8 Test Unit 08: Power computational Performance depending on client flavour

4.8.8.1 Responsibilities

WP2.3, WP3.6, WP4.2 and TID as partner, are responsible for definition and execution of this test.

4.8.8.2 Test Specification

Test Items

We test the following features of the AEM interface:

- Submission of a job that executes a C program and a bash script in a resource node.

Features to be Tested

The main goal of this test is to check the independence between the computational power of the Grid and the XtreemOS client used (PC or Mobile Device): both clients should offer a similar performance. In addition, we want to demonstrate the utility of XtreemOS on mobile devices, by comparing the results with the ones obtained with a direct execution of the tests on the mobile device, not using the Grid.

Approach Refinements

In order to execute the test, the xsub.sh XATICA command will be used to submit a job to the Grid. This job will consist of a C program (xos_performace.c) that uses the `gettimeofday` utility of `sys/time` library to control the date and get time statistics about the process executions.

A shell script will be launched from the xos-perfomance executable. This script contains a simple counter implemented with a while loop. The loop will be able to count until a numeric limit specified as a parameter has been reached.

The bash script will be invoked with a parameter indicating the numeric limit for the loop (from 5 to 200). Also, the program will save initial execution date and final execution date. For each numeric limit, the process will be repeated fifty times and the corresponding results (time taken for the execution) will be processed. This processing includes calculating the arithmetic average of the execution times and presenting them as a text file.

This test will be executed in the same XtreemOS core, from a XtreemOS-PC client (PC flavor), a XtreemOS-MD client, and directly on a Nokia N800 device. The result will be a set of average execution times for each numeric limit and for each different client type.

4.8.8.3 Test Results

This test has been executed using an internal XtreemOS testbed in TID headquarters. The testbed contains a XtreemOS 1.0 core running as a VMware virtual machine, as well as, a XtreemOS 1.0 client running as a VMware virtual machine. The XtreemOS-MD version has been run on a Nokia N800 connected to the testbed network.

The test was run following this procedure for each device (PC client, MD client on a Nokia N800 and Nokia N800 without XtreemOS):

Installation

The program xos-performance and the bash script are copied to the XtreemFS (local disk in the case of Nokia without XtreemOS) volume of the user performing the tests.

Set up

A JSDL file (test-performace.jsdl) is created to define the test job with the tag "executable" specifying a file in the current directory, `./xos-performance`. Note that this step is not necessary for the Nokia without XtreemOS.

Executation

The XATICA command "xsub" is invoked with the following parameters and options:
`xsub -vf test-performance.jsdl`
AEM returns the the ID of the job submitted.

Getting results

Once the process has finished, the file `performance-results.txt` appears in the XtreemFS volume of the user performing the tests. The file contains a table with the measured times for each iteration and the final results.

Procedure Results

The execution of this test was successful. The results related to this job correctly presented using a graph, are shown in the figure 4.44

- `Nokia` label identifies the data related to execution of the test on a Nokia device directly (without any XtreemOS flavor installed) using its own processor.
- `XOS-PC` label identifies the data set related to execution of the test using the Grid through XtreemOS client (PC flavour).
- `XOS-MD` label identifies the data set related to execution of the test using the Grid through XtreemOS-MD installed on a Nokia device.

The figure 4.44 shows the job execution performance using the Grid from XtreemOS-MD (running on the mobile Nokia N800) is similar to the performance offered by using the Grid from the XtreemOS client PC flavour (running in a standard PC). By comparing the higher performance obtained by submitting the tests to

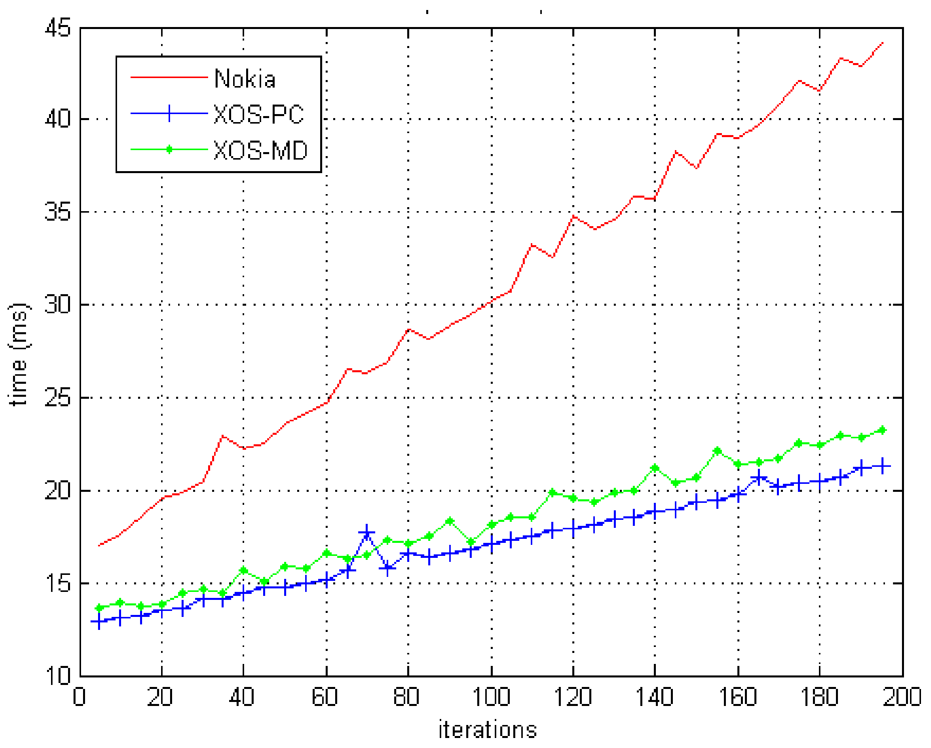


Figure 4.44: Comparison of job execution power among XOS-PC, XOS-MD and using directly the Mobile device

the Grid from XtreemOS-MD rather than running them directly on the Nokia, we can observe the benefit of using Grid solutions for executing computational jobs, especially from mobile devices.

4.8.9 Test Unit 09: Customizable SSI Scheduler

The Customizable Scheduler is the component of the LinuxSSI operating system, which manages balancing the processes between the cluster nodes. The scheduler enables the user to replace the load-balancing policies in runtime without the need of restarting the whole cluster.

4.8.9.1 Responsibilities

Both the developed code and the performance tests are the responsibility of XLAB within the WP2.2.

4.8.9.2 Test Specification

Test Items

In this test unit we test the Customizable LinuxSSI scheduler. The scheduler is installed as a part of LinuxSSI kernel and utilties, which are an installation option within the XtreemOS installation. Installing and configuring LinuxSSI is outlined in the XtreemOS Administration Guide.

Features to be Tested

As a part of this test unit, we will measure the following LinuxSSI scheduler characteristics:

- The speedup that we gain by automatically balancing processes between cluster nodes,
- The effect of the customizable scheduler overhead on the execution time of parallel jobs. The scheduler overhead includes continuous resource polling, measurements filtering and scheduling policies triggering.

Approach Refinements

For the purpose of testing the overhead induced by the LinuxSSI scheduler, we have implemented "primes-solver-parallel", a simple parallel application for searching the primes (i.e. the numbers that are divisible only by 1 and by itself) in a given interval. We chose the algorithm for searching primes because it is trivially paralellizable and it doesn't require the parallel parts to exchange any info among themselves. This way, we can get much more accurate measurements of the scheduler overhead.

The plan is to execute the parallel application in 3 types of environments:

- In an environment where no load scheduling is enabled (i.e. the “no scheduling” environment). This means all the parallel processes are executed on the same node. By executing the tests in this type of environment, we estimate the benefits of having the automatic scheduling in the first place,
- In an environment where the LinuxSSI scheduler is enabled (i.e. the “LinuxSSI scheduler” environment). The processes are started on the same node and are automatically migrated to other cluster nodes during the runtime,
- In an environment where parallel processes are already started on separate nodes (i.e. the “manual scheduling” environment). In this scenario,the LinuxSSI scheduler is not required, thus dropping any potential overhead from the LinuxSSI scheduler.

We will execute the tests on the 3 nodes of the “paradent” cluster in Rennes (this cluster is a part of the Grid5000 Grid system). The testing procedure will be as follows:

1. Boot the LinuxSSI operating system on all testing nodes, but don’t start the customizable scheduler at this time.

2. Execute the "primes-solver-parallel" program on a single node and measure its execution time. All the execution times are measured for the interval [2,1000000]:

   ```
   primes-solver-sequential 2 1000000 1
   ```

3. After the sequential test is finished, perform a manual execution of the "primes-solver-parallel" processes on separate nodes, each on a different subset of numbers in the interval:

   on node 1: `primes-solver-parallel 2 1000000 NUM_NODES`
   on node 2: `primes-solver-parallel 3 1000000 NUM_NODES`
   on node 3: `primes-solver-parallel 4 1000000 NUM_NODES`

   ...

   on node NUM_NODES:
   `primes-solver-sequential NUM_NODES+1 1000000 NUM_NODES`

   After the programs on all nodes are finished, collect all the execution times. The total execution time is equal to the longest execution (that is, we apply the "MAX" function to all the execution times).

4. Enable the LinuxSSI scheduler by executing the “start_linuxssi_scheduler.sh” command.

5. Execute the “primes-solver-parallel” on a single node. The LinuxSSI scheduler will take care of migrating the processes around the cluster nodes:

   ```
   primes-solver-parallel 2 1000000 NUM_NODES
   ```

4.8.9.3 Test Results

The results of the LinuxSSI scheduler are presented in the table 4.18 and figure 4.45. We can see that scheduling (either manual or the one performed by LinuxSSI) causes significant improvement in the parallel applications execution. With scheduling enabled the “primes-solver-parallel” program was able to finish twice as fast as with scheduling disabled.

Furthermore, we can see that the execution time with LinuxSSI scheduler enabled is only 1.06 seconds longer from the ideal case where all the processes are balanced by the user from the beginning. This presents only the 0.95% overhead when compared to the total execution time. From this data, we can observe the LinuxSSI scheduler does not impose a significant performance overhead on the cluster.

Table 4.18: Execution time for the “primes-solver-parallel” program in different environments. The execution times are presented in seconds. There were in total five repetitions of each test performed.

Testing environment	Execution time (sec)
no scheduling	226.15
manual scheduling	111.58
LinuxSSI scheduler	112.64

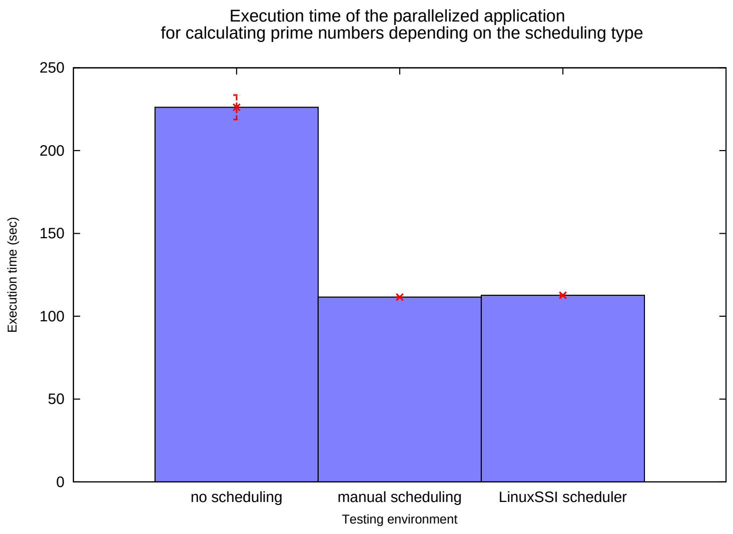


Figure 4.45: Graph of the execution times for the “primes-solver-parallel” program. On the graph, the standard deviation of the measurements is also presented.

4.8.10 Test Summary Report

4.8.10.1 Summary of Tests and Results

We performed tests to validate AEM functional requirements, evaluate its performance and compare it to the performance of Globus toolkit. Tests were performed to assess the suitability of XtreemFS as a platform for running web server We also performed tests to evaluate the difference of performance between XtreemOS PC flavor and a grid running the Condor middleware. The robustness of the AEM scheduler was also tested. Our tests also included comparative perfromance analysis of job submission responce times of XtreemOS and Globus. We also evaluated power computational performance depending on client flavour.

4.8.10.2 Conclusion and Directions for Future Work

Following our tests we conclude that that AEM in XtreemOS Release 2 are adequate. We also have proven that XtreemOS is a suitable environment for hosting web services and applications. Our tests have shown that the perfromance of XtreemOS and Condor system is comparable, and for some tests like testing management times, XtreemOS performs better. With regard to the robustness of AEM, we discovered that only a part of the jobs submitted reaches the Done final Status, which shows that the robustness of AEM scheduler still leaves the room for improvement. The results of comparative performance analysis obtained in the job submission and execution at XtreemOS and Globus show a great improvement over Globus in this basic unit.

4.9 Evaluation of Data Management

4.9.1 Test Plan

This test plan covers the distributed file system XtreemFS developed by WP3.4. and Object Sharing Service (OSS) provided by the XtreemOS release 2.0. XtreemFS functionalities include:

- high-performance distributed file system for federated installations across multiple organizations
- fully POSIX-compliant with extensions
- suitable for Wide Area Networks (WANs) with high latencies between sites
- suitable for environments with complex failure cases like network partitioning and similarities between slow and dead nodes
- support for replication and partitioning of metadata servers, replication and striping at file/object level
- integration into Virtual Organizations
- self-monitoring and autonomous optimization of file distribution, layout and access
- transparent object sharing service

kDFS functionalities include:

- kDFS aims at providing an integrated cluster file system for High Performance Computing.
- fully POSIX-compliant.
- support replication and striping at file/object level
- cooperative caching strategies.
- self-monitoring and autonomous optimization of file distribution, layout and access

4.9.1.1 Responsibilities

The test and evaluation execution is conducted under WP2.2, WP3.4 and WP4.2. Below we provide the names of partners and a brief description of their test applications and suites:

- SAP: Enterprise data search and analytics (TREX), MaxDB database system and the bonnie IO benchmark

- XLAB: Distributed financial modelling framework.
- UDUS: An interactive multi-user 3D virtual world application,
- CNR: The NTFS-3G suite for testing XtreemFS POSIX-compliance,
- ZIB: IOzone file system benchmark for parallel IO evaluation
- Kerlab: Bonnie IO benchmark to test performance of kDFS

4.9.1.2 Test Items

The test items include:

- The software to be tested is XtreemFS. Publications are available on the XtreemFS website: `www.xtreemfs.org`. The XtreemFS version tested is the release 1.1 from 2009-09-18. Source and documentation are available from the internal XtreemOS SVN.
- Distributed file system kDFS and its documentation is available at `http://www.kerrighed.org/wiki/index.php/KernelDevelKdFS`.
- Bonnie is a benchmark which measures the performance of Unix file system operations. The benchmark and its documentation is available at `http://sourceforge.net/projects/bonnie`, version 1.03.
- The MaxDB distribution and its documentation is available under `https://www.sdn.sap.com/irj/sdn/maxdb`.
- Information about TREX can be found under: `https://www.sdn.sap.com/irj/sdn/nw-search`.
- Information about Wissenheim can be found under: `http://www.wissenheim.de`.
- The Pawel Jakub DawidekŠs POSIX filesystem test suite (PJD-fstest) that is part of NTFS-3G suite. The PJD-fstest suite software and documentation are available at http://www.tuxera.com/community/posix-test-suite/.
- IOzone, a file system benchmark. Source and documentation are downloadable from http://www.iozone.org/.

4.9.1.3 Features to be Tested

The following features will be tested:

- POSIX compliance (open, read, write, close, ls, rm, touch, mv, cp, mkdir, cd, rmdir)

- stability
- I/O performance
- scalability
- runtime of application benchmarks
- striping policies
- OSD and replica selection policies
- I/O performance of read/write operations for kDFS
- I/O performance of file metadat operations for kDFS

4.9.1.4 Features not to be Tested

The following features will not be tested as they are not supported by the current official release or for other reason, that is pointed out near the feature:

- resilience to failure cases like network partitioning etc. The feature is not tested since it is based on the read/write replication which is still not available.
- replication will not be tested since the current official release supports read-only replication, while the test applications impose mixed read/write load.
- integration into Virtual Organizations will be done at the next test iteration.

4.9.1.5 Overall Approach

The purpose of the test is to provide feedback to developers about the implemented features and about the fulfilment of requirements.

The performance of XtreemFS and OSS will be evaluated using applications and benchmarcs. The tests will also provide comparative performance analysis of XtreemFS and NFS file systems. The corresponding tests will be performed at the same dedicated cluster environment without using virtualization layers, ensuring accuracy, reproducibility and fairness of the comparison.

Benchmark Focus

We will start testing XtreemFS using Bonnie benchmark. It will be performed in Test Unit 4.9.8. This benchmark is typically used for performance measurements of Unix file systems. We will use it to test the basic functionality of XtreemFS and evaluate its performance.

POSIX compliance of XtreemFS will be tested at Test Unit 4.9.6 using the Pawel Jakub DawidekŠs POSIX filesystem test suite (PJD-fstest).

The performance evaluation of the XtreemFS software stack (client and server) and the parallel I/O for striped files in a cluster environment will be done using IOzone, a file system benchmark. The evaluation will be carried out by Test Unit 4.9.7.
The perfromance of kDFS will be assessed using Bonnie benchmark. It will be performed in Test Unit 4.9.9. Bonnie benchmark will be used to test the basic functionality of kDFS and evaluate its performance.

Application Focus

After testing XtreemFS using benchmarks, we will test XtreemFS using several applications. Tests are performed by installing XtreemFS on testbeds of local machines and executing the maxDB replay and TREX (provided by SAP) and WISS (provided by UDUS). It was decided to choose maxDB replay as reference application since in typical multi-tier business solutions the great majority of file operations is transactional relying on a central database. In practical business scenarios, end-user applications access this database via the middleware WEB Application Server. In the experiments with XtreemFS, it was decided to stress the file system via the maxDB replay (recorded accesses of maxDB to the filesystem) from SAP which also allows to simulate multi-user access with parallel read and write operations. These tests will be performed at Test Unit 4.9.2.

TREX is another SAP application that is used as an indexing and enterprise search server. At the index stage, TREX typically indexes a large collection of documents, generating a significant read load on the underlying file system. After reading the input documents TREX produces a large amount of index data that is written to a distributed file system in a number of phases. Thus at our experiments TREX is used to generate two separate access loads: reading input documents and writing the index files. The corresponding tests will be carried out in Test Unit 4.9.3.

Wissenheim is a distributed interactive 3D virtual world for edutainment and entertainment. It utilises the Object Sharing Service (OSS) of XtreemOS to distribute its shared game state so that every participating node can alter the shared data directly. Conflicting accesses are synchronised by an optimistic transactional scheme provided by OSS. While the dynamic game data is distributed by OSS the static graphical data is shared via XtreemFS. The experiments with Wissenheim will be carried out at Test Units 4.9.4 and 4.9.5.

Whereas the SAP experiments test a database-centric and distributed search engine scenarios, the experiments with WISS are file-based accessing the file system via OSS. Table 4.9.1.5 summarizes different file system usage characteristics, tested by the test applications.

4.9.2 Test Unit 01: MaxDB Replay

4.9.2.1 Responsibilities

This test unit and the included tests with MaxDB replay are the responsibility of SAP (WP4.2).

	Bonnie	MaxDB	TREX	WISS
Distr. Trans. Load		X		
Distr. Random Read			X	X
Read only load				X
Sequential access			X	
Standard FS ops.	X	X		X
Size of files	X	8.5G	0.3G	<0.5MB
Overall file size	X	200G	1.3G	8MB
Number of files	X	37	85	50-100
OSS usage				X
Directory structure		tree	tree	tree

Table 4.19: File system usage characteristics

4.9.2.2 Test Specification

Test Items

The software to be tested is XtreemFS. Publications are available on the XtreemFS website: www.xtreemfs.org.

The XtreemFS version tested is the release 1.1 from 2009-09-18. Source and documentation are available from the internal XtreemOS SVN.

The MaxDB distribution and its documentation is available under https://www.sdn.sap.com/irj/sdn/maxdb.

Features to be Tested

The following features and requirements are the subject of this test design specification:

- creating directories using **mkdir** Unix command: R50, R76
- copying files using **cp** Unix command (the existing MaxDB data files are copied from local file system to XtreemFS volume): R50
- opening/reading/writing/closing existing files (MaxDB data files) using POSIX/UNIX system calls: R1, R50, R79,
- using different striping degrees and number of OSDs for stability, scalability and performance testing: R1, R50, R79
- running MaxDB tests against XtreemFS and NFS for comparative performance analysis: R1, R50, R79
- adding and removing MRCs and OSDs to and from the working system to test stability: R1, R50

Approach Refinements

The goal of the test is to test and evaluate XtreemFS file system under a transactional load that is generated by a typical business application. During the test we will perfom comparative analysis of XtreemFS and NFS.

The testbed used in these tests is a 15 nodes cluster with SuSE Linux Enterprise 11 installed at each node. The nodes are connected via standard ethernet which has a transfer rate of 1GB/s (E 1Gb/s). Performing the tests at the same dedicated cluster environment without using virtualization layers, ensures accuracy, reproducibility and fareness of the comparison. We will use standard performance metrics: latency and throughput of read and write file system operations.

This test uses recorded IO access traces that MaxDB generated while supporting a real-life application. During the test, IO load of a real-life SAP application is applied to XtreemFS by replaying the recorded IO traces over XtreemFS. Replaying the traces, especially when they are concurrently replayed from several nodes, generates a considerable amount of IO to the data files (called MaxDB volumes). In the sequel we will refer to concurrently replayed traces as *concurrent IO streams*. During the recording process, MaxDB ran over NetApp filer. We use the performance of this filer as the baseline in our experiments. Note that since the filer used SSD technology for non-volatile memory implementation to speed up write-to-log operations, the baseline latency of the filer write operations is much better than for other filesystems in our experiments. For the preparation of a MaxDB replay run on XtreemFS, one needs to copy the existing volume files created during the MaxDB installation to XtreemFS and symbolically link the MaxDB volume directory to the XtreemFS directory with the copied files.

In the experiments the number of OSDs equals the stripe width of the XtreemFS volume. The test is repeated with several stripe widths and number of concurrent IO streams to test the system's reliability, scalability and performance. For each combination of stripe width and number of concurrent IO streams, the IO latency and throughput are measured and compared to learn if increasing stripe width (and number of OSDs) improves the resulting performance of the system. Each test was performed three times and for each performance characteristic (throughput and latency) we report average value of the characteristic over the three test repetitions.

The same tests will also be conducted over two different filesystems: NFS and XtreemFS for comparative performance analysis over the file systems as well as how increasing striping width of XtreemFS affects the performance of the file systems.

4.9.2.3 Test Results

Figure 4.46 presents the read latency results for the baseline filer technology, NFS volume and XtreemFS volumes with Stripe Widths (SW) 1, 4 and 8. At our experiments the number of OSDs equals the SW of each volume. When NFS file system is compared with the XtreemFS in the case of 1 IO stream, NFS file system provides the best read latency, reflecting the fact that it is a mature system, widely spread in the industry. However when the number of concurrent IO streams grows to 4 and 8, the latency of NFS grows sharply, showing bad scalability. XtreemFS, however shows much better resilience to scaling IO load. We explain this fenomenon by the fact that the accumulated working set of the concurrent IO streams grows lineary with the number of IO streams. Each OSD server in XtreemFS stores its objects in the local file system. The local file system utilizes its own cache capabilities to cache the results of the recent read operations. Thus, when XtreemFS reads data, the read data is cached in all the caches of all OSD servers that support the XtreemFS volume. Actually XtreemFS utilizes *a distributed cache* composed of the caches of the individual supporting OSD servers. The size of the resulting XtreemFS distributed cache grows lineary with the number of OSDs and that is why this cache is able to incorporate the big working set of several IO streams. Since NFS has the same cache size for the increasing numberf of IO streams, once the size of this application work set grows beyond the NFS cache size, the IO performance of NFS drops dramatically.

Note also that for SWs 4 and 8, XtreemFS is able even to get better latency for 4 concurrent IO streams than for 1 IO stream. We explain this phenomenon by the fact that while the concurrent IO streams run the same IO trace, they work out of phase. In this situation the first IO stream that performs a specific IO access that is recorded in the trace loads the corresponding file stripe to the cache of the corresponding OSD. When the other IO streams perform the same access, they find this file stripe already in the cache, avoiding the necessity to access the disk. This interpretation gets more support from the fact that the latency improvement is better for SW 8 than for SW 4, because the bigger distributed cache size of the volume with SW 8.

Figure 4.47 shows the “mirror” effect of Figure 4.46, where the throughput of each file system increases as function of the distributed cache size of the file system.

Figure 4.48 shows write latency of the baseline filer technology, NFS volume and XtreemFS volumes with SW 1, 4 and 8. All files in those experiments were open with O_SYNC flag, blocking the calling process until the data has been physically written to the underlying hardware. Since the filer used SSD non-volatile memory for the write operations, its latency is far below the latencies of the other file systems. As with the latency of the read operations, the latency of write operations of NFS shows bad scalability with growing number of IO streams, while the latency of write operations of XtreemFS volumes is more resilient for the growing number of IO streams. Furthermore, it may be observed that a higher SW provides with better resilience to the growing number of IO streams. Our interpretation of this observation is that with growing number of SW (and respectively OSDs) XtreemFS distributes the synchronous write operations among several OSDs and thus affectively load balance the write load and reduces the collisions when several write operations are executed at the same time.

Figure 4.49 shows write throughput in our experiments. As in the case of read

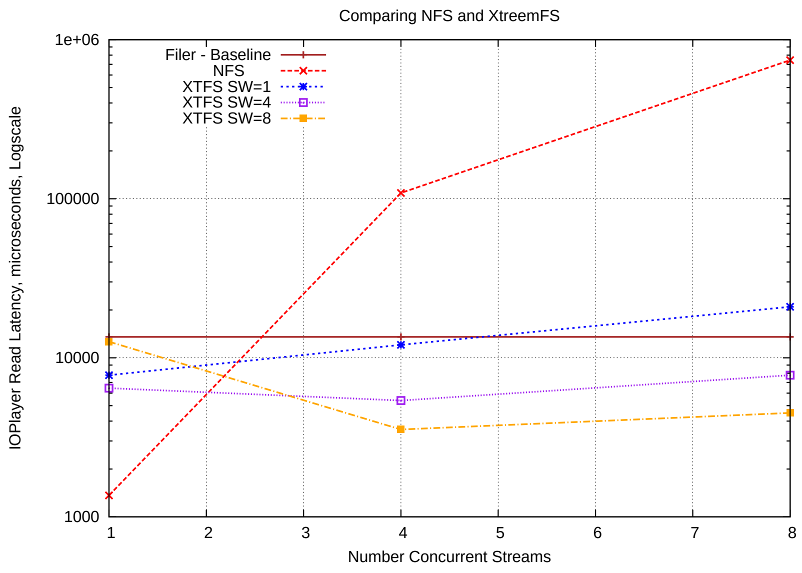


Figure 4.46: Read Latency in MaxDB experiments.

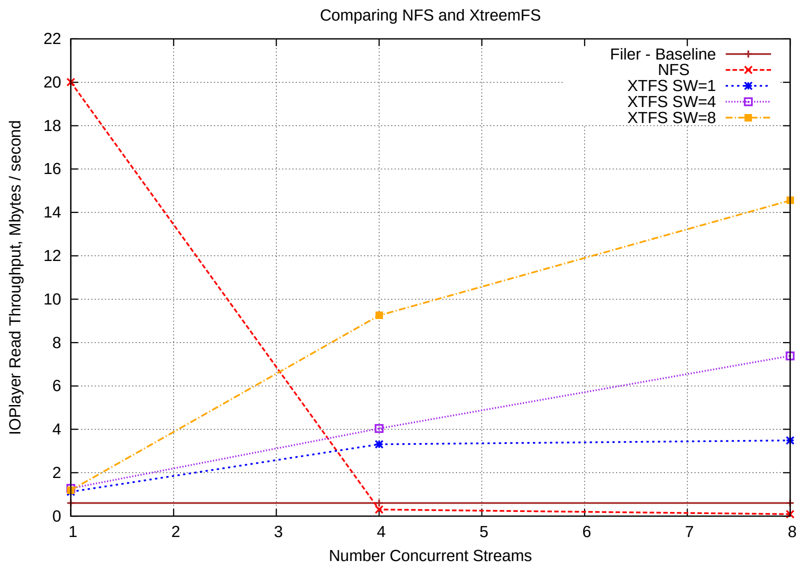


Figure 4.47: Read Throughput in MaxDB experiments.

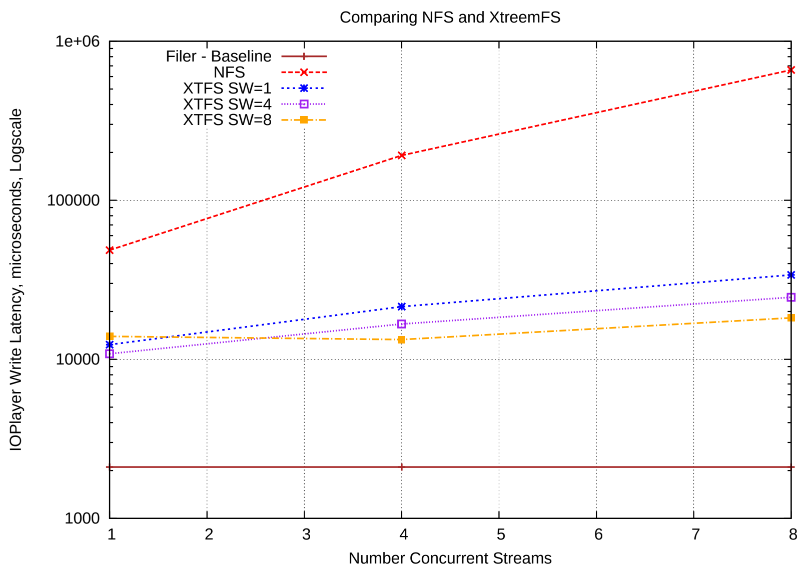


Figure 4.48: Write Latency in MaxDB experiments.

operations, the graphs of throughput of write operations provide with the "mirror" picture of the write latency. Note, however, that with SW 8, XtreamFS almost approaches the throughput of the baseline filer inspite of the fact that the filer used SSD non-volatile memory to shorten latency. This clearly shows the advantage of the distributed XtreemFS filesystem to scale out with the commodity hardware, as opposed to the scale-up capability of the filer technology.

Conclusions:

We performed the performance analysis measurements which show promising results that support the follwoing conclusions:

- XtreemFS effectively utilizes distributed cache whose size grows lineary with SW of the volume. This distributed XtreemFS cache is capable to incorporate big application work sets and thus provide the base for data volume scalability at high performance.

- Big distributed cache of XtreemFS enables low latences and high throughput for applications with large work set.

- When an application uses synchronious write operations, XtreemFS effectively distributes those write operations over several OSDs enabling low latences and high throghput.

- XtreemFS may help to reduce TCO of running business applications: XtreemFS

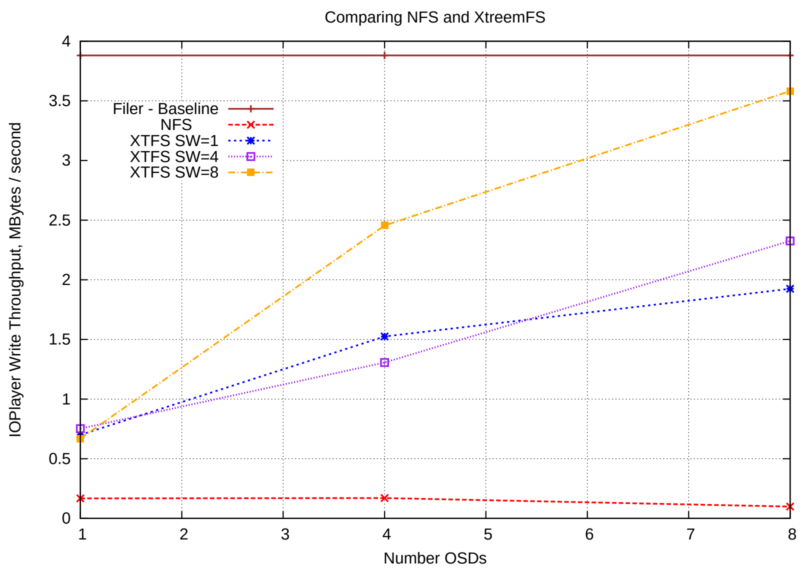


Figure 4.49: Write Throughput in MaxDB experiments.

scales out over commodity hardware - it runs over commodity hardware and its peformance scales almost lineary with stripe width.

- Overall conclusion: our experiments show the potential of XtreemFS to effectively support transactional load. However more work should be done to stabilize XtreemFS and to prove its ability to support business application transactional load. It may be also required to utilize SSD-approach at each OSD node to achieve low write latency, which is crucial for implementing transactional write load.

4.9.3 Test Unit 02: TREX

4.9.3.1 Responsibilities

This test unit and the included tests with TREX are the responsibility of SAP (WP4.2)

4.9.3.2 Test Specification

Test Items

The software to be tested is XtreemFS. Publications are available on the XtreemFS website: `www.xtreemfs.org`.

The XtreemFS version tested is the release 1.1 from 2009-09-18. Source and documentation are available from the internal XtreemOS SVN.

Information about TREX can be found under:
https://www.sdn.sap.com/irj/sdn/nw-search.

Features to be Tested

The following features and requirements are the subject of this test design specification:

- creating directories using **mkdir** Unix command: R50, R76
- opening/reading/writing/closing existing files (TREX data and index files) using POSIX/UNIX system calls: R1, R50, R76, R79
- using different striping degrees and number of OSDs for stability, scalability and performance testing: R1, R50, R76, R79
- running TREX tests against two filesystems: XtreemFS and NFS: R1, R50, R76, R79

Approach Refinements

The goal of the test is to test and evaluate XtreemFS file system under an IO load that is generated by an enterprise search application. During the test we will perfom comparative analysis of XtreemFS and NFS. The testbed used in these tests is a 15 nodes cluster with SuSE Linux Enterprise 11 installed at each node. The nodes are connected via standard ethernet which has a transfer rate of 1GB/s (E 1Gb/s). Performing the tests at the same dedicated cluster environment without using virtualization layers, ensures accuracy, reproducibility and fareness of the comparison. We will use application-oriented performence metric - the overall time to complete the indexing operation.

This test includes running TREX index server on a large collection of documents, stored at XtreemFS. Reading the documents generates read accesses to XtreemFS. The resulting index files are written back to XtreemFS. The two stages produce read and write load respectively on XtreemFS. The elapsed time of index creation operation is measured.

The test is repeated with several values of the input parameters (striping degrees and number of OSDs) to test the system's reliability, scalability and performance. For each combination of the scalability factor values, the overall time of index creation is measured and finally compared to find if changing scalability factor values improves the resulting performance of the system.

The same tests will also be conducted over two different filesystems: NFS XtreemFS for comparative performance analysis over the file systems as well as

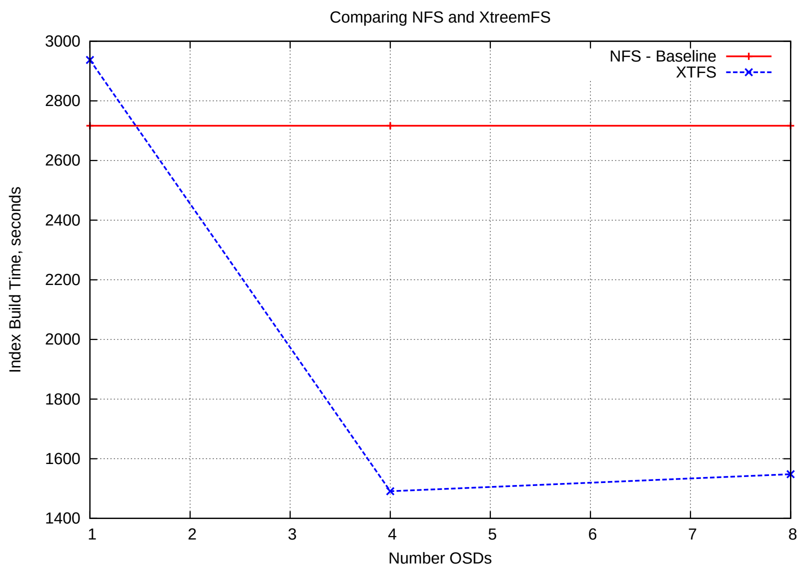


Figure 4.50: Elapsed time to create TREX index.

how varying the input parameters for XtreemFS affects the performance of the file system.

4.9.3.3 Test Results

Figure 4.50 presents the results of measuring the elapsed time to create the index under NFS volume and XtreemFS volumes with Stripe Widths (SW) 1, 4 and 8. Note that NFS file system provides a better result than XtreemFS volume with SW 1, which is explained by the fact that NFS is a mature file system that is present at the industry for several year, while XtreemFS is still a research file system that is not mature yet. However for SW 4 and 8 XtreemFS shows the improvement of almost factor 2 as compared to the performance of NFS file system. We attribute this performance improvement to two XtreemFS features:

- XtreemFS with SW at least 2 distributes IO load among several OSD servers.
- In addition to IO load distribution XtreemFS utilizes distributed cache, based on the involved OSD servers, to incorporate large application work sets.

Note also that the performance of XtreemFS slightly drops at SW 8 as compared to SW 4. We explain this phenomenon by the lease coordination mechanism that XtreemFS uses to synchronize parallel IO accesses. This mechanism starts to impose performance overhead for growing number of SW of the file system volumes.

4.9.4 Test Unit 03: Wissenheim - XFS

4.9.4.1 Responsibilities

WP4.2, UDUS

4.9.4.2 Test Specification

Test Items

The test item will be the XtreemFS component of XtreemOS which is part of the XtreemOS 2.0 distribution. XtreemFS can also be obtained from www.xtreemfs.org along with user and install documentation.

Features to be Tested
The following file system features and requirements will be tested:

1. creating XFS-volume for Wissenheim graphics data
2. initializing XFS-volume with Wissenheim graphics data (recursive directory structure, mixture of small and medium size files)
3. mounting of XFS-volume on different nodes and testing data integrity and availability using Wissenheim

Approach Refinements
The main purpose is to test XtreemFS as a mean to distribute the large amount of graphics data used by Wissenheim throughout the XtreemOS grid. Therefore, a blank volume will be created and initialized with the current Wissenheim media data set. Parallel access from Wissenheim clients will check if the XtreemFS volume is accessible and if all data is in a consistent state. Using the *netem* network simulator we will simulate packet loss and high latency to determine if access to XtreemFS can be guaranteed under wide area conditions. Wissenheim will access the data in a typical scene loading scenario generating random access to small and medium sized files. We will measure the impact of latency and packet loss to the overall load time for selected Wissenheim scenes. As an additional test we have used the packet corruption feature of netem to check if XtreemFS provides error detection for arbitrary bit errors in packets. More information about netem can be found at: http://www.linuxfoundation.org/collaborate/workgroups/networking/netem. The consistency of the data retrieved from XtreemFS will be checked by the resource management of Wissenheim. Figure 4.51 gives an overview over the scenes used in testing and about the data size and amount of files. We have selected the two big scenes RainbowIsland and FATCave which are comparable in size and file count to check if both perform similar. The DiffieHellman scene is roughly half the size of the bigger scenes so we should see comparable faster load times with increasing latency.

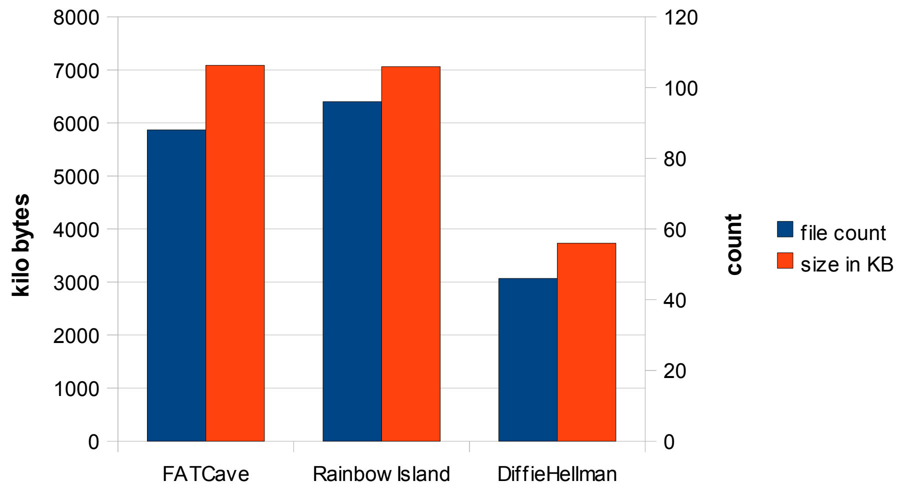


Figure 4.51: Wissenheim scenes used.

4.9.4.3 Test Results

The results of the latency test are shown in figure 4.52. The load time increases linearly with the increase in latency and both big scenes behave nearly the same. No error occurred while increasing the latency. As you can see the load time of the DiffieHellman scene is faster with increasing latency compared to the bigger scenes. With no latency the load time is not limited by the disk transfers but by the decoding process of the media data performed by Wissenheim.

The packet loss results are shown in figure 4.53. We used uniform distribution with no variation in packet loss. All data could be read and was in a consistent state throughout the test.

When activating the packet corruption feature of netem Wissenheim detected several errors. This was expected because XtreemFS does not yet including mechanism to counter such corruption.

4.9.5 Test Unit 04: Wissenheim - OSS

4.9.5.1 Responsibilities

WP4.2, UDUS

4.9.5.2 Test Specification

Test Items

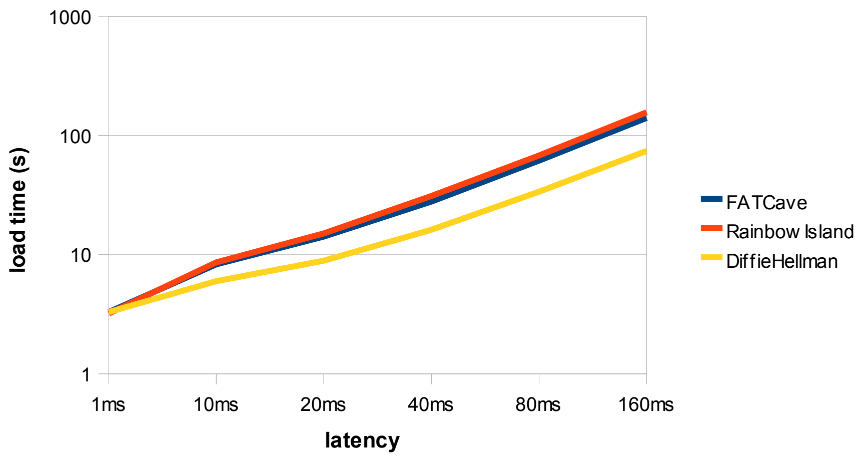


Figure 4.52: Load time with increasing latency.

Test item will be the Object Sharing Service provided by the XtreemOS release 2.0. Documentation can be found in the subversion repository of XtreemOS.

Features to be Tested

- OSS basic network setup.
- OSS distributed transactional memory.

Approach Refinements
Wissenheim is using OSS to share the game state of the world among all participating nodes. To ensure synchronized access to the data speculative transactions provided by OSS are used. All tests have been performed with the Wissenheim scene DiffieHellman and two players.

OSS uses a token based mechanism to achieve the serialization of commit requests so the latency for requiring a token will be directly responsible for the latency experienced by a player in the game. As Wissenheim is optimized to reduce transactional conflicts, aborts of transactions due to collisions can be neglected. We have measured the token latency over time using two real world setups with the following parameters:

Backbone: Connection between Düsseldorf and Ulm, both using the DFN backbone with at least 100MBit, low latency network links and public IPs.

DSL: Connection between Düsseldorf and Ulm with Düsseldorf connected via DFN and Ulm connected via DSL with a downlink of 6Mbit and an uplink

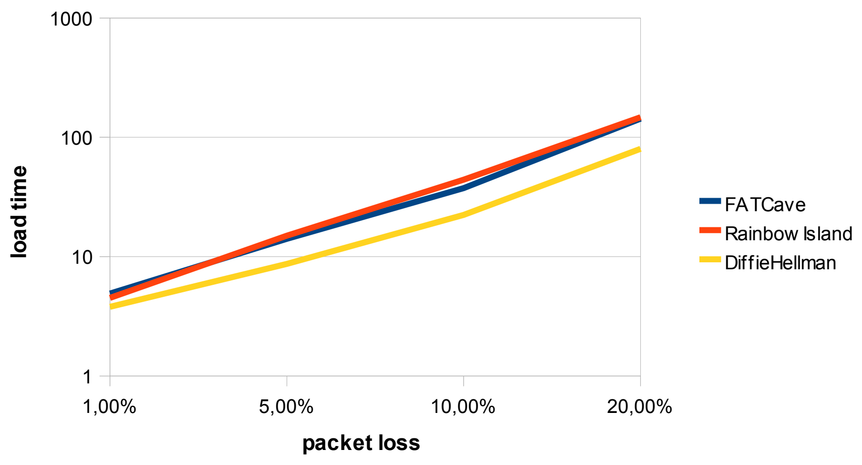


Figure 4.53: Load time with increasing packet loss.

of 1MBit. Additional the client at Ulm was connected via WLAN using the 802.11g protocol within a private network using NAT.

4.9.5.3 Test Results

The results of the wide area test are shown in figure 4.54. The peak at the beginning marks the load time of the scene when the joining node is requesting the shared game state and thus produces heavy network traffic. The measured average roundtrip time for pings between Ulm and DÃijsseldorf using the *backend* connection was about 13ms, the roundtrip time for the *DSL* connection about 80ms. After joining, the latency remained nearly constant in case of the *backbone* connection while the *DSL* connection shows significant more noise due to the DSL and WLAN error rate and latencies. In both cases Wissenheim was perfectly playable with almost no latency issues detectable.

4.9.6 Test Unit 05: POSIX Compliance

4.9.6.1 Responsibilities

This test unit and the included tests on the POSIX compliance are the responsibility of CNR (WP3.4). Testing activity has been conducted in collaboration with BSC.

4.9.6.2 Test Specification

Test Items

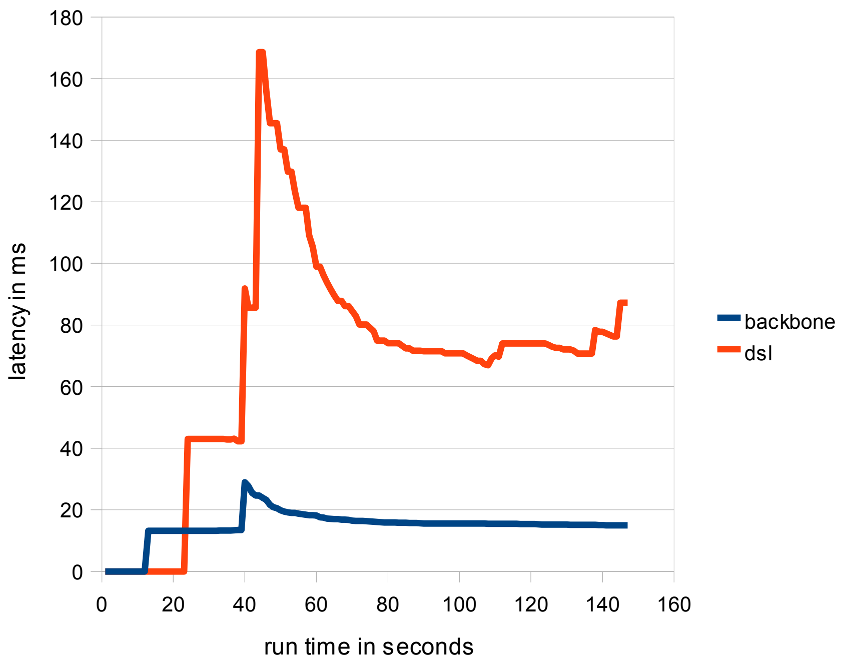


Figure 4.54: Token latency over time.

The software to be tested is XtreemFS. Publications are available on the XtreemFS website: www.xtreemfs.org.

The XtreemFS version tested is the release 1.1 from 2009-09-18. Source and documentation are available from the internal XtreemOS SVN.

The XtreemFS POSIX-compliance was tested by the NTFS-3G suite, that is a freely and commercially available and supported read/write NTFS driver for the most important operating systems. In particular, it includes a POSIX file system test environment, namely the Pawel Jakub Dawidek's POSIX filesystem test suite (PJD-fstest). The PJD-fstest suite software and documentation are available at http://www.tuxera.com/community/posix-test-suite/. The PJD-fstest used for the tests is the latest stable release pjd-fstest-20080816 (released on August 16, 2008) and downloadable from http://tuxera.com/sw/qa/pjd-fstest-20080816.tgz.

Features to be Tested

PJD-fstest suite performs 1957 regression tests that exhaustively check a wide amount of different scenarios for the following system calls:

- *chmod*: changes the permissions of files or directories

- *chown*: changes ownership of files or directories
- *link*: creates hard links
- *mkdir*: creates directories
- *mkfifo*: creates fifo files named pipes
- *open*: opens and eventually creates a file
- *rename*: changes file or directory names
- *rmdir*: removes directories
- *symlink*: creates symbolic links
- *truncate*: decrease/increase file size
- *unlink*: removes regular files, symbolic links, fifos and sockets

Approach Refinements

The goal of the test is to evaluate the POSIX compliance of XtreemFS. POSIX (Portable Operating System Interface for Unix) is the name of a family of related standards specified by the IEEE to define the application programming interface (API), along with shell and utilities interfaces for software compatible with variants of the Unix operating system.

The PJD-fstest suite performs a set of operations on files and directories, i.e. create, remove, rename, truncate, permission and owner changes, as well as operations on special files, i.e. hard/symbolic links and fifos. For each system call, the suite executes of a set of scripts. Each script performs a set of basic operations, like the creation of a directory, the change of its access rights, etc., and it evaluates, for each one, its execution and return value. If its manner of acting or its return value are different than that expected (as specified by POSIX), an error is pointed out. In particular, the suite is "system call-oriented", which means that the scripts performing the tests for a particular system call are composed of operations targeted for the evaluation of the (hopefully correct) behaviour of that system call.

To execute the tests, we implemented a tool that basically automatizes all the process of updating, compiling, installing XtreemFS and running a basic scenario with one Directory Service (DS), one Metadata and Replica Catalogue (MRC) and one Object Storage Device (OSD), and creating a volume and mounting it on a specific directory. Once this scenario is up and running, the tests are executed in the mount-point where the volume has been mounted. We decided to execute such kind of tests in such simple scenario because we consider it is sufficient for the specified test purposes. In other words, the behaviour of XtreemFS (from the POSIX compliance view point) is the same if a different number of OSD, MRC or

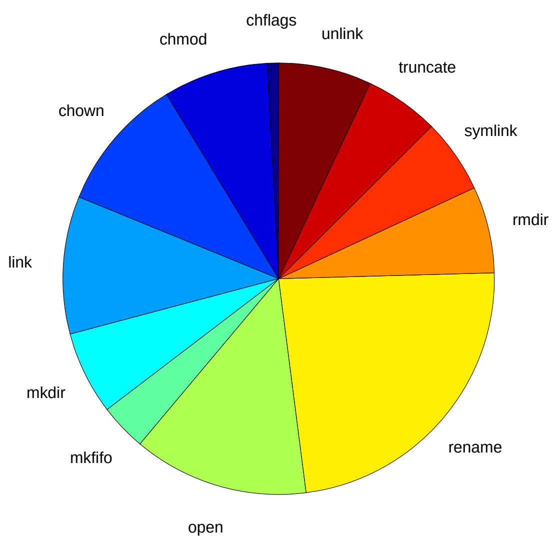


Figure 4.55: PJD test suite: number of tests performed on each system call.

DS are used. In fact, since the POSIX tests query and verify only metadata information, they are affected only by the logic implemented in the MRC. Moreover, no analysis on performances is undertaken in such kind of tests.

The testing activity consisted in the automatic execution of the scripts and in the evaluation of the failure events. Then, in order to understand the cause of each failure, it was needed to interpret the cause of the problem and reproduce manually the scenario (the sequence of operations) causing it.

4.9.6.3 Test Results

As above described, the PJD-fstest suite executes, for each system call *sc* to be tested, a set of scripts aimed at verifying the correctness of that operation. More in detail, each script executes various tests and verifies their return value. In Figure 4.55 a plot of all the system call tested by the PJD test suite is reported. Moreover, for each system call the number of tests performed on it is reported.

Before to report test results, it is relevant to make two observations. First, we do say that XtreemFS actually does not support fifos mechanisms. For this reason, in order to make our evaluation as much correct as possible, we skipped tests performed by the PJD tests suite on the `mkfifo` system call (totally, 217 tests). Second, XtreemFS supports file names and directory names longer than 256 characters. Since this is considered as an error by the PJD tests suite (despite of the fact that POSIX specifications do not limit arbitrary filename length), we removed also such kind of tests (10 tests). For such reasons, we totally removed 227 tests from the original PJD suite test set, resulting in 1730 final tests.

Now, let us report the results of POSIX-compliance evaluation on XtreemFS. By considering all the 1730 tests performed by the PJD test suite, actually XtreemFS passes exactly 1556 tests, corresponding to 89.94% of the total. In the following table, a more detailed report of the test environment and obtained results is reported. In particular, the table shows, for each system call, the number of scripts, the total number of tests performed by the scripts, the number of tests satisfied and the success rate (the tests passed w.r.t. the total number of tests executed).

system call	**no. of scripts**	**no. of tests**	**no. of successful tests**	**% success rate**
`chflags`	14	14	14	100.00%
`chmod`	12	128	127	99.22%
`chown`	11	206	164	79.61%
`link`	18	167	161	96.41%
`mkdir`	13	101	101	100.00%
`open`	24	217	213	98.16%
`rename`	21	479	379	79.12%
`rmdir`	16	109	105	96.33%
`symlink`	13	90	90	100.00%
`truncate`	14	90	90	100.00%
`unlink`	14	129	112	86.82%
Total	183	1730	1556	89.94%

The activity aimed at making XtreemFS as much POSIX-compliant as possible is a work in progress. Many errors have been detected and corrected, but others are under evaluation.

4.9.7 Test Unit 06: Parallel I/O Evaluation

4.9.7.1 Responsibilities

This test unit and the included tests on the parallel I/O for striped files in a cluster environment are the responsibility of ZIB (WP3.4).

4.9.7.2 Test Specification

Test Items

The software to be tested is XtreemFS. Publications are available on the XtreemFS website: `www.xtreemfs.org`.

The XtreemFS version tested is the release 1.1 from 2009-09-18. Source and documentation are available from the internal XtreemOS SVN.

In such kind of tests XtreemFS was tested by IOzone, a file system benchmark that generates and measures a variety of file operations. The IOzone version used in such tests is the release 3.283. Source and documentation are downloadable from `http://www.iozone.org/`.

Features to be Tested

Tests described in this unit are aimed at evaluating the performance of the XtreemFS software stack (client and server) and the parallel I/O for striped files in a cluster environment. In particular, the tests have two goals. The first one is to evaluate the performance for parallel I/O with file sizes that fit into OSD main memory. This means that the measured bandwidth is not limited by the hard disk but by the speed of the XtreemFS implementation. The second goal is to test and show the effect of caching with a single OSD and client.

Approach Refinements

The goal of the test is to evaluate the performance of the XtreemFS software stack (client and server) and the parallel I/O for striped files in a cluster environment. All tests were conducted on nodes with eight cores (Xeon E5420, 2.5GHz), 16GB RAM and a single HD with approx. 60 MB/s read/write rate. The nodes are connected via infiniband which has a transfer rate of 1.2GB/s (gigabit/s) via TCP (IPoIB). We used one node for the XtreemFS DIR and MRC, ten nodes for the IOZone clients and one to 15 nodes for the OSDs.

4.9.7.3 Test Results

As above mentioned, a first goal of these tests consists in evaluating the performance for parallel I/O with file sizes that fit into OSD main memory. The test environment contemplates the presence of twenty six physical nodes. Ten nodes are used to host a varying number of clients (as described in the following), fifteen nodes to host the OSDs and one node for both the MRC and the DIR.

IOZone is a widely used synthetic file system benchmark. It tests a file system with sequential, random and stride read/writes. In particular, out tests were aimed at evaluating performances of the *read*, *re-read*, *write* and *re-write* operations. *Read* consists in reading an existing file, *write* in writing a new file, *re-read* in reading a file that was recently read and *re-write* in writing a file that already exists.

We used a file size of 100MB per client and a record size of 256KB which is also the stripe size of the volume. We conducted the IOzone throughput test and measured the *aggregated bandwidth* of the IOzone client processes. For *aggregated bandwidth* we mean the sum of bandwidth of all client processes. In particular, we report in Figures 4.56 and 4.57 the results when 20 and 40 clients are running, respectively.

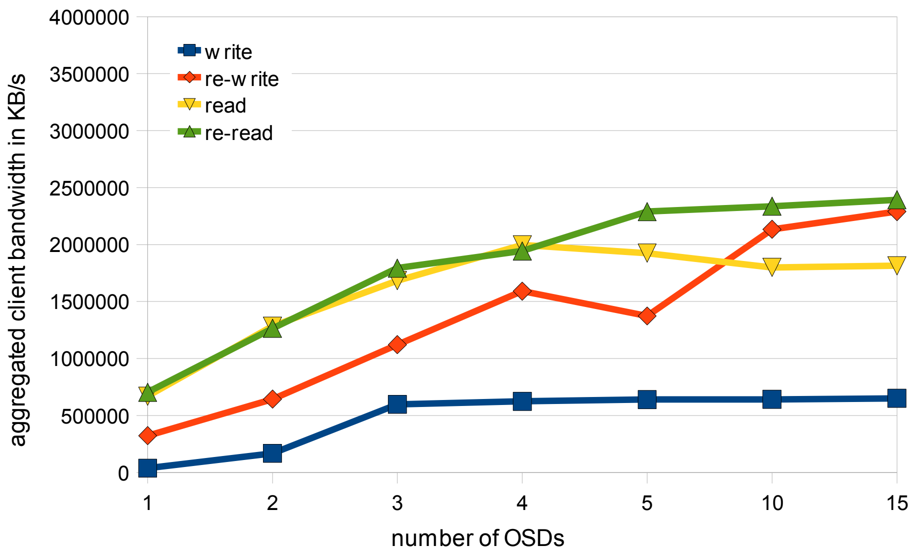


Figure 4.56: Bandwidth of clients for an IOZone throughput test with 20 clients (on 10 physical nodes) with 1-15 OSDs.

For what concerns the re-write, read and re-read performance, we notice that they do scale well with the number of OSDs in Figure 4.56 up to five OSDs. The lack of further increase in performance is due to the limits in the client. To justify this assertion, we have to take into account that each client executes its read/write operations sequentially. So, although the 20 clients are running in parallel, they do not generate enough load to reach the maximum throughput offered by a number of OSDs greater than five. Accordingly, the bandwidth scales better with number of OSDs when using 40 client processes (Fig. 4.57). In both experiments, the initial write bandwidth is much lower than the re-write or both read bandwidths. This is due to the fact that for the initial write of a file, the OSDs need to allocate files on disk and the client has to update the file size at the MRC. This is not necessary for re-writing or reading the file.

For what concerns the write operation, we have to notice that write performance does not scale at all with #OSDs greater than 3. This is due to the fact that the current version of the XtreemFS client updates the file size after each write operation, which results in the MRC being the bottleneck for writes. We guess that this limit will be overcome in the upcoming XtreemFS release, which is expected to cache file size updates and consequently achieve more scalable performance for the write operation.

In a last experiment we measured the bandwidth for a single client process using a single OSD for increasing file sizes. The results are reported in Figure 4.58. As expected, the read bandwidth assumes values under 60 MB/s (maximum throughput of the HD) when the file size is larger than the available cache. The

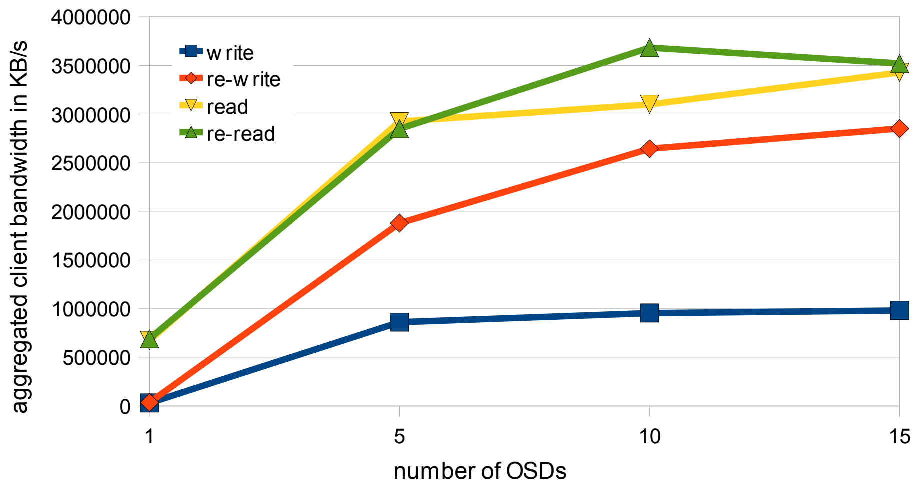


Figure 4.57: Bandwidth of clients for an IOZone throughput test with 40 clients (on 10 physical nodes) with 1-15 OSDs.

bandwidth for write decreases earlier as the Linux kernel allows only a pre-defined fraction of the page cache to be dirty. In fact, the Linux cache has a limit of pages which can be dirty (which is expressed as a fraction of the total cache). If this limit is exceeded, writes have to be flushed to disk before they are acknowledged to the application.

4.9.8 Test Unit 07: Non-Parallel I/O Evaluation

4.9.8.1 Responsibilities

This test unit and the included tests using the bonnie IO benchmark are the responsibility of SAP (WP4.2).

4.9.8.2 Test Specification

Test Items

The software to be tested is XtreemFS. Publications and installation files are available on the XtreemFS website: `www.xtreemfs.org`.
The XtreemFS version tested is the release 1.1 from 2009-09-18. Source and documentation are available from the internal XtreemOS SVN.
At this test unit we used the bonnie++ benchmark to test the performance of XtreemFS. The bonnie++ version used in our tests is the release 1.03. Source and

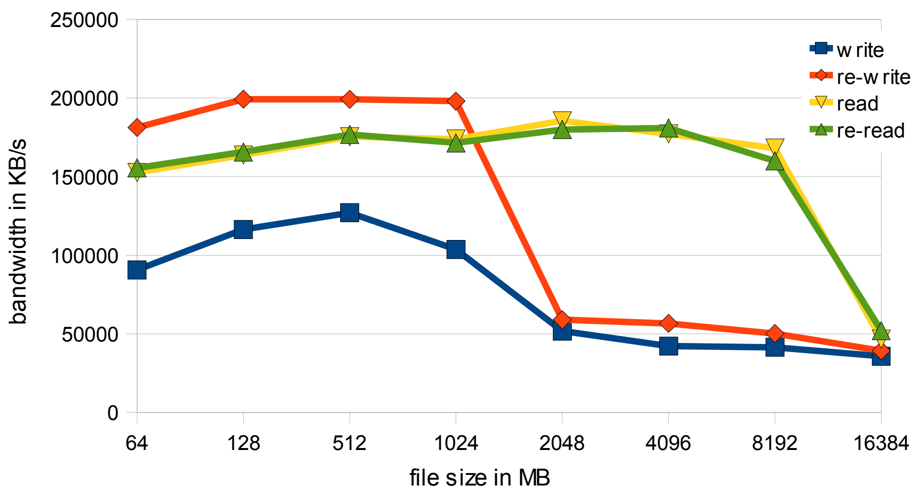


Figure 4.58: Bandwidth of a single client and a single OSD for increasing file sizes.

documentation are downloadable from http://sourceforge.net/projects/bonnie/.

Features to be Tested

Bonnie++ tests the types of filesystem activity that have been observed to be bottlenecks in I/O-intensive applications. The generated IO load is non-parallel. Tests described in this unit serve for comparative performance analysis of NFS and XtreemFS file systems. They are composed of the following tests:

- Sequential per-character output performance
- Sequential block output performance
- Rewrite performance. The data is read, dirted and written back. This should test the effectiveness of the filesystem cache and the speed of data transfer.
- Sequential per-character input performance
- Sequential block input performance
- Random seeks performance - several parallel processes are doing a total of 8000 lseek()s to locations in the file specified by a random function. In 10% of cases, the block is dirtied and written back.

Approach Refinements

The goal of the test is to evaluate the performance of the XtreemFS filesystem and provide performance comparative analysis in a cluster environment. All tests were conducted on nodes with two cores (Duo CPU E8400, 3GHz), 3.8GB RAM and a single HD with approx. 60 MB/s read/write rate. The nodes are connected via standard ethernet which has a transfer rate of 1GB/s (E 1Gb/s). We used one node for the XtreemFS DIR and MRC, one node for the Bonnie client and one to 3 nodes for the OSDs.

4.9.8.3 Test Results

Bonnie provides performance tests for several filesystem operations over files of the specified aggregated size. In our experiments the aggregated size of the files is 16G. Initially Bonnie performs the sequential per-character writes, where each character is written separately. Then it issues the test with sequential block writes. Next it uses rewrite tests to check the effectiveness of the filesystem cache and the speed of data transfer. Following write tests, Bonnie starts read tests that consist of sequential per-character read and sequential block read tests. Finally is runs random seek tests, using several parallel processes. In our experiments we used three parallel processes for seek tests. In the most of the tests Bonnie imposes non-parallel IO load on the filesystem in question. As Figure 4.59 suggests, in the case of non-parallel IO load XtreemFS, as distributed-architecture file system, performs worse than NFS. NFS is a mature filesystem that is present at the industry for several years and it is well tuned for performance. Since in read/write tests there is no a big and stable application work set that could fit into the distributed cache of XtreemFS, such a cache is helpless.
Note also that the performance of XtreemFS degrades when the stripe width (and correspondilgy the number of OSDs) grows. We attribute this observation to the lease-based synchronization meachanism that XtreemFS uses to synchronize IO accesses.
In contrast to the read-write tests, the seek test is performed using three processes and in this case we anticipate that the distributed nature of XtreemFS architecture and implementation will manifest in better performance as compared to NFS. And indeed, looking at Figure 4.60 one can see that the performance of XtreemFS grows sharply with the increase of stripe width (and number of OSDs), outperforming NFS at stripe width 3.

4.9.9 Test Unit 08: kDFS

4.9.9.1 Responsibilities

WP2.2, Marko Obrovac (INRIA)

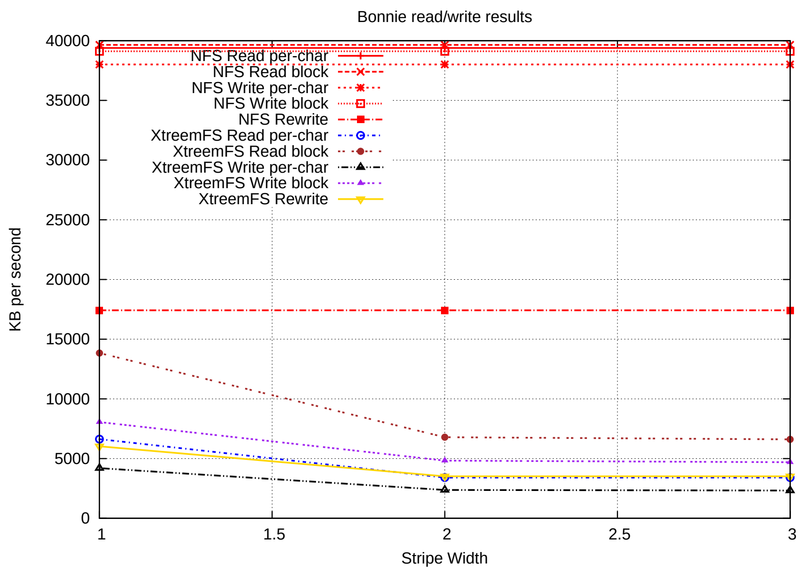


Figure 4.59: Performance of read / write operations under NFS and XtreemFS filesystems.

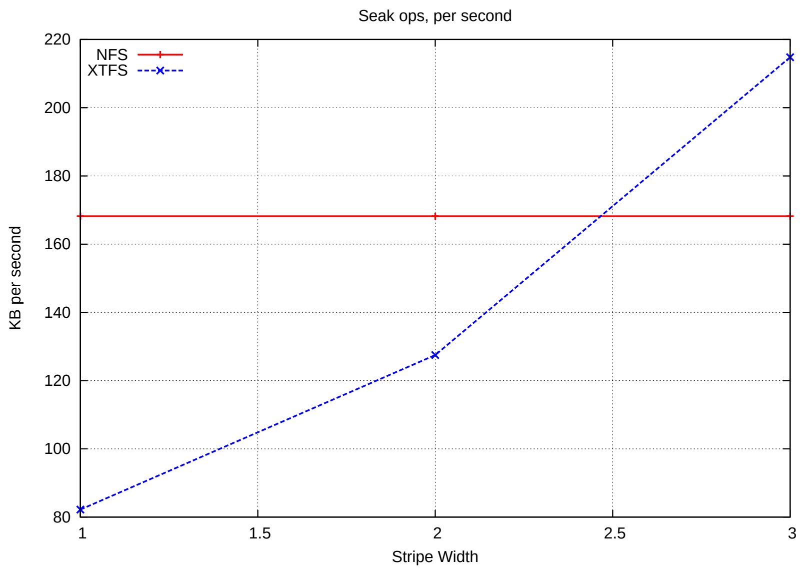


Figure 4.60: Performance of seek operations under NFS and XtreemFS filesystems.

4.9.9.2 Test Specification

Test Items

In this test unit, we will test kDFS - the fully distributed symmetric cluster file system which ships with LinuxSSI.

Features to be Tested

The main concern of this test is the performance of kDFS, specifically, the read/write performance using different access patterns. In order to ensure unbiased and reproducible test results, we used Bonnie++ - an industry-standard, POSIX-compatible I/O benchmarking tool. Bonnie++ conducts the tests in a few steps:

- read/write access:
  - sequential writing: per character, per block, rewrites
  - sequential reading: per character, per block, random seeks
- file operations:
  - sequential create, read, delete
  - random create, read, delete

To make a qualitative evaluation of kDFS' performance and capabilities, Bonnie will be run multiple times with different block sizes, which means kDFS will not only be tested with different access patterns in mind, but it will also be evaluated for different computing application types.

Approach Refinements

The purpose of this experiment is to benchmark kDFS' performance and to show its stability. The tests are carried out in the Grid5000 environment. Concretely, the tests were performed on the Rennes cluster. Eight physical machines were used, each having the following specifications:

- Dell PowerEdge 1950
- Intel Xeon 5148 LV 2.33 GHz x 2
- 8 GB RAM
- 2 x 300GB Raid0 / SATA
- Gigabit ethernet

Although the nodes are equipped with 8 GB of RAM, we used only 4GB in our tests due to the kernel's memory addressing limitations.

Setting up the environment

On those eight nodes we installed an XtreemOS 2.0 - LinuxSSI environment adapted for the Grid5000 platform. The deployment was carried out with the help of the `xos-ssi-deploy` script available in the XtreemOS GForge repository. The script performs the following steps:

- deploy the *xos20-i586-ssi* environment
- configure the nodes:
  - set up correct node and session ids
  - set up the host names
  - deliver a list of nodes involved in the cluster to all of the nodes
  - change the default run level
  - enable the `nfs-server` service on the head node and disable it on all the other nodes
- reboot the nodes
- start the cluster services and mount NFS partitions
  - on the head node: `start_linuxssi`
  - on the worker nodes: `start_linuxssi_as_worker`

Next, we need to create and mount kDFS partitions. Again, a helper script is used: `ssi-mkfs-kdfs`. The steps performed on every node are the following:

- create a directory to hold the partition
- run `mkfs_kdfs`
- after `mkfs_kdfs` has completed creating all of the partitions, mount every one of them

Finally, we need to run the benchmark tool - Bonnie. We launch eight instances of Bonnie in parallel - one on each node. That way, we can exploit kDFS' cooperative cache. On every node, we launch the following command:

- `bonnie++ -d /media/kdfs/tmp-node_id/ -s 8192:block_size -n 1 -r 4096 -u root`

The command launches the execution of Bonnie with the following parameters:

- directory to write to/read from: `/media/kdfs/tmp-node_id/` (`node_id` is substituted with the actual node's id - that way any possible naming clashes are avoided)

- file size: `8 GB`
- block size: it changes with every iteration. We tested kDFS with the following values: `512 B`, `4096 B`, `32768 B` and `1048576 B`
- number of files for the files creation test: `1024`
- quantity of physical RAM: `4 GB`
- the uid of the user which will execute the tests: `root`

4.9.9.3 Test Results

All the proposed Bonnie tests were successfully completed. They were performed on the eight nodes in parallel and the results shown here represent the median values obtained from the tests on each node.

Table 4.20 shows the results of sequential write operations for different block sizes. As expected, when the data is written character by character, there are almost no variations in the results. When testing character write operations, Bonnie ignores the block sizes. Thus, since the tests were executed on matching machines, similar results have been obtained.

However, there is great disproportion in block writes, as well as in rewrites (which are also block-size-dependant). With a 512-byte block size the performance is really poor - not even one megabyte of data gets to be written per second. On the other hand, we can see performance improvement with higher block sizes (13.3MB/s with a 4096-byte block size, 13.4MB/s with a 32678-byte block size and 12.7MB/s with a 1048576-byte block size). Although it seems there is almost no difference in performance when (re)writing data with a 4096-byte, 32768-byte or 1048576-block block size, we should keep in mind that all of the figures presented in these tables are medians over all tested nodes. Since kDFS is a parallel cluster file system, we should keep in mind that all those writes happen in parallel. kDFS handles all of the writes - it processes eight times the data presented in the tables. Of course, the disks themselves have to handle the actual writes.

Table 4.20: Sequential write operations for different block sizes (in kB/s)

Block Size	Character Write	Block Write	Rewrite
512	11403	803	805
4096	11443	13553	12944
32768	11263	13652	13312
1048576	11306	13003	13098

Although write operations seem somehow limited, we can see a real performance boost in sequential read operations, for which the results are presented in table 4.21. As in the case of single character write operations, the performance of single character read operations are independent of the block size.

Naturally, block sizes higher than 1 give us better performance. With as low as a 512-byte block size, the read performance has almost doubled (from 45.4MB/s to 84.2MB/s). It continues to rise and peeks at 241.8MB/s with a block size of 32768 bytes. Again, keep in mind kDFS actually reads eight times the size of data shown in the table (since the results are per node).

Table 4.21: Sequential read operations for different block sizes (in kB/s)

Block Size	Character Read	Block Read	Random Seek (per sec)
512	46468	86139	138.4
4096	48341	233111	349.9
32768	49438	247645	119.4
1048576	49124	238819	51

File creation and deletion tests are Bonnie's last benchmark. They control system stability and POSIX compliance. The results of those tests, presented in table 4.22, show kDFS has passed all of the tests, which means it is a stable and fully POSIX-compliant file system.

Table 4.22: File creation and deletion tests

Access Type	Create	Read	Delete
Sequential	OK	OK	OK
Random	OK	OK	OK

To put the figures represented in tables 4.20 and 4.21 into perspective, we ran the same Bonnie tests (with more or less the same parameters) on NFS partitions within the same environment. However, since we use NFS for comparison only, Bonnie tested only the 4kB block size performance. The side-to-side comparison between kDFS and NFS can be found in table 4.23. As the table shows, kDFS outperforms NFS by over a double in block reading operations, and by almost a double in block writing operations. Moreover, the difference between the two becomes larger and larger as we add more machines to the cluster: while kDFS has a global cooperative cache and is able to write data locally on disks, NFS not only adds network traffic overhead, it also reads to/writes from only one disk.

Table 4.23: Side-to-side comparison of I/O characteristics of kDFS and NFS (in kB/s)

File system	Block Read	Block Write
kDFS	48341	13553
NFS	21770	7298

Finally, we would like you to keep in mind that the figures presented in this test are relative and are highly influenced by the backbone exploited by kDFS. The more important include:

- *hardware itself* - any test, especially a file system performance test, cannot be reproduced with the exact same results in a different environment
- *underlying file system* - kDFS is not a low-level file system. It does not write nor read directly to/from the disk. It uses the VFS and the underlying file system to do disk-level access. As such, even in the same environment, the results could be quite different depending on which file system was chosen as the local one.

4.9.10 Test Summary Report

4.9.10.1 Summary of Tests and Results

We performed tests to validate XtreemFS functional requirements, evaluate its performance and compare it to the performance of NFS. The functional requirements of XtreemFS were also tested in the simulated at real wan environment. Tests were also performed to assess functional requirements of Object Sharing Service provided by the XtreemOS. We also evaluated the performance of Object Sharing Service provided by XtreemOS. The perfromance of kDFS was evaluated using the Bonnie benchmark.

It was found that POSIX complience tests passes 89.4% of the adequate tests. Non-parallel IO tests reveal that for sequential access pattern XtreemFS does not provide performance improvements as compared to NFS. Parallel IO tests reviel that the performance of XtreemFS scales well with the increase in the number of OSDs. XtreemFS also shows very good scalability under transactional load and the load generated by enterprise search application. It effectively caches big application IO work set, utilizing its de-facto distributed cache based on OSDs. Even without using SSD non-volatile memory, it enables to almost reach the throughput of the baseline filer technology that uses SSD non-voletile memory. We also found that applications using XtreemFS are capable to work under wan-simulated latency. It was found that Object Sharing Service may be used to share the application state among nodes in the wan conditions. With regard to kDFS perfromance evaluation we found that the read/write performance improves when the block size grows. kDFS successfuly passed the file creation and deletion tests of the Bonnie benchmark, which control system stability and POSIX compliance. It means that kDFS is a stable and POSIX-compliant file system. Tests were also performed to compare the perfromance of kDFS and NFS. The tests show that kDFS outperforms NFS almost by factor 2.

4.9.10.2 Conclusion and Directions for Future Work

Based on the tests we conclude that POSIX Compliance leaves the room for improvement. While currently the MRC is the bottleneck for writes, this limit will probably be overcome in the upcoming XtreemFS release, which is expected to cache file size updates. Write latency under the transactional load is still very low as compared to the baseline NetApp filer technology. And most probably this shortcoming may be overcome only by means of using SSD non-voletile memory at OSD nodes. Our experimental results under transactional load looks very promising and suggest that XtreemFS may support transactional load. However more experiments need to be performed to support this conclusion. We also experimentally assessed that XtreemFS does not yet include mechanism to counter packet corruption in the wan conditions.

Based on the tests perfromed for kDFS we conclude that it is POSIX compliant. We also conclude that the global cooperative cache of kDFS enables it to cache the application work set and write the data locally to disk. As the result kDFS outperforms NFS filesystem at our tests.

We can conclude that XtreemFS, Object Sharing Service in XtreemOS Release 2 and kDFS are adequate, with all major functional requirements met but performance and some functional requirements leaving some room for improvement. Further testing is required for features skipped in this iteration, most of which relate to comparative performance analysis of XtreemFS with kDFS and with other advanced research file systems such as CEPH. Another set of tests to be performed in the next iteration will target read/write replication and using it to overcome failures such as network partitioning and to improve performance of applications in wan conditions.

4.10 Evaluation of Security Services

4.10.1 Test Plan

This test plan evaluates the security services of XtreemOS, developed in WP 3.5. The features and components that have been evaluated are as follows:

- Identification of the major performance costs (e.g. issuing and verifying of delegation tokens) of a new delegation technology, named DToken, being developed for XtreemOS and preliminary evaluation of the technology
- Policy management including the insertion, removal, saving, evaluation and storage
- Issuing of XtreemOS Certificates

The focus of testing in this deliverable is on performance and impact on compute resources. There are various reasons for focusing on these two attributes:

- Security is often identified as a bottleneck and overhead in a system's operation and maintenance
- Security services can add additional load to messages and interactions
- Security services may be targets of denial of service attacks, which seek to exhaust the resources of the host the services are running on

4.10.1.1 Responsibilities

The tests reported were carried out within WP3.5 by STFC and XLAB, the developers of the security services evaluated here. SAP was responsible for review of the experiments and interpretation. The experiments were also done in parallel with the vulnerability assessment of XtreemOS in deliverable D3.5.14, performed by SAP.

4.10.1.2 Test Items

The test items have been selected based on a) the availability of meaningful evaluation results and the nolvety of the technology; or b) the impact they have on a node's performance and resource usage.

- **DTokens** is a delegation certificate format and protocol designed for XtreemOS but is not currently packaged in release 2.0 of XtreemOS. It has been published by Yang and Matthews [55] at the 2009 IEEE Symposium on Reliable Distributed Systems (SRDS)

- **CDA (Certificate Distribution Authority)** server, version 0.2.4. This is included in release 2.0 of XtreemOS and is available as a patch for XtreemOS 1.0. The CDA server is a critical component of the XVOMS component configured on a core node.

- **VOPS (VO Policy Service)** is a service under development at XLAB. It is designed to support local and VO-wide policy administration, filtering and multiple decision requests.

4.10.1.3 Features to be Tested

The DTokens will be tested against another more popular form of delegation certificate, which is incorporated in other Grid middleware. The essential features of VOPS to be tested surround the operations to be performed by and administrator for policy administration. However, the focus is on the compute time associated with these operations being performed. Finally, the performance of certificate issuing by the CDA will be tested, focusing on the impact of handling parallel requests.

4.10.1.4 Features not to be Tested

All notable features of VOPS are to be tested, as well as of the CDA. However, given that DTokens is still under development, there are some features excluded from this testing:

1. Communication overhead of dtokens

2. Chained delegation (delegation that involves more than two parties)

3. Multiple delegation (concurrent delegations to multiple parties)

4. Scalability, how does DToken scale with regards to:

   (a) the number of processes within a single-level delegation to one delegatee

   (b) the number of processes within a single-level delegation to more than one delegatees

5. The relationship with SSL (DToken is built upon SSL, this is to investigate whether the use of DToken has any implications to SSL and vice versa)

6. The detailed cost breakdown of various DToken protocols

7. The performance of this technology when packaged with XtreemOS

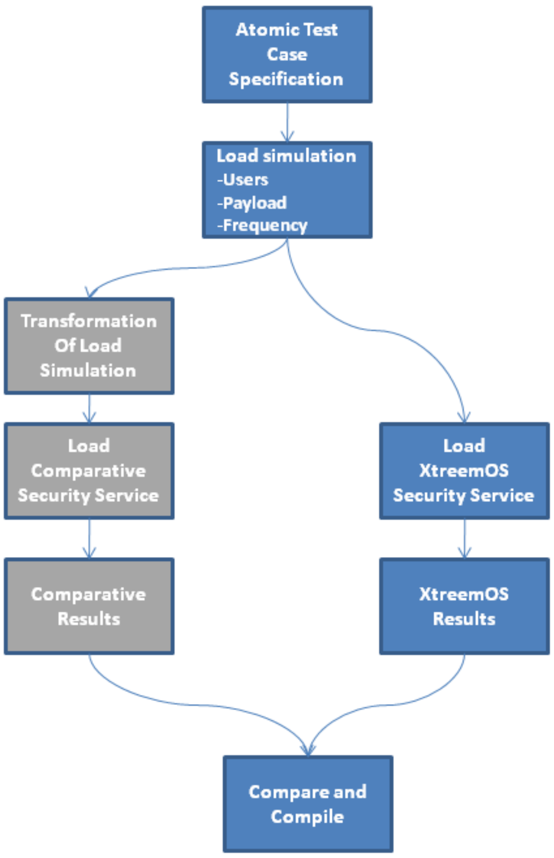

Atomic Test Case Specification
Load simulation
-Users
-Payload
-Frequency
Transformation Of Load Simulation
Load Comparative Security Service
Comparative Results
Load XtreemOS Security Service
XtreemOS Results
Compare and Compile

Figure 4.61: Overall approach for testing security services

4.10.1.5 Overall Approach

The approach to evaluating XtreemOS Security services is shown in Figure 4.61.

The first stage is the specification of atomic test cases. These define a single usecase for the service under normal conditions. For example, the atomic test case specification checks that a specific function or operation responds correctly to a request. Secondly, the atomic test case is used to create a simulated load on the component. This will include simulation of several users making the same request, varying payloads in the request or varying frequencies of the request. There are then two branches of assessment that may be done; firstly, the simulated load is used to test the performance of the XtreemOS security service under the load conditions. The results are response times, CPU usage and memory consumption. Secondly, depending on the test case, the simulated load is transformed for a different environment, against which the XtreemOS security service is to be compared. For example, the DToken technology was compared against another form of delegation technology (i.e. Proxy Certificate). However, it is not always relevant or feasible to have comparative results of security services. In such cases the goal of the test case is to show that the service will not be disruptive under normal, production loads. Secondly, in the case of abnormal loads, the goal is to better predict the changes in operational behaviour that will be observed.

4.10.2 Test Unit 01: DTokens vs. GSI Proxy Certificates

4.10.2.1 Responsibilities

We describe here a test carried out within WP3.5 by STFC comparing the DTokens approach with GSI Proxy Certificates.

4.10.2.2 Test Specification

Test Items

In XtreemOS, a delegator and a delegatee jointly generate a a delegation token, named dtoken, to facilitate delegation. Through the token, a machine is able to check whether a user has delegated to one (or more) machine(s) without directly contacting the user or any trusted on-line third-party service. The whole approach has been called DTokens, it is under development in WP3.5 and will be fully reported in deliverable D3.5.9. Description of the DTokens approach and explanation of the test was published in [55].

Features to be Tested

We compare here XtreemOS DTokens with Proxy Certificates [41], a RFC standard proposed as part of the Grid Security Infrastructure (GSI). We test the following features for managing delegation tokens/proxy certificates:

- Creation of a delegation token/proxy certificate.
- Verification of authenticity of a delegation token/proxy certificate.

The evaluation focuses on a single-level delegation, in which the size of delegation tokens is trivial (less than 3kb). Therefore, in this case, the communication costs involved in the delegation is clearly not significant.

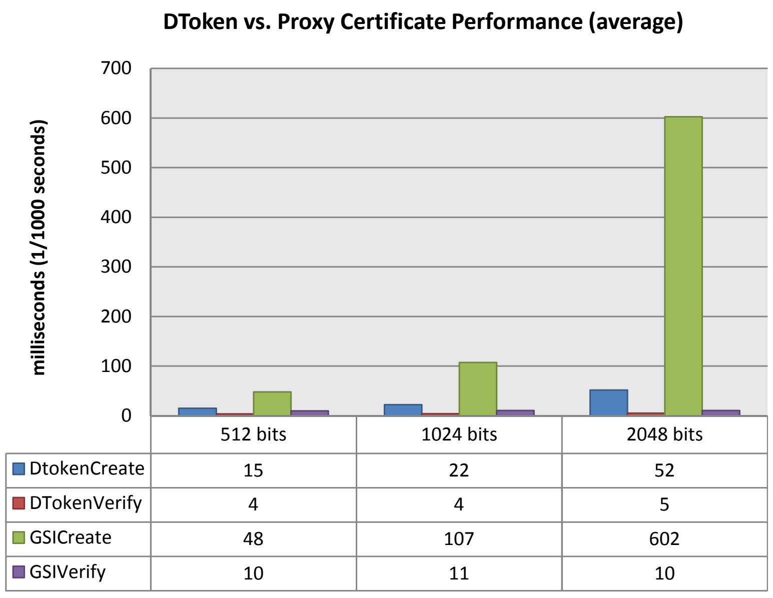
Approach Refinements

All the experiments are conducted on a Ubuntu OS (8.10) running on top of a virtual box (2.1.4). The prototype is written entirely in C, is compiled using GCC (4.3.2) and depends solely on the OpenSSL toolkit (0.9.8). The GSI proxy certificate performance data was collected using the command `grid-proxy-init` of the latest globus toolkit (4.2.1). The source program of this command was slightly modified to include performance measuring procedures. All the performance data was collected by repeatedly executing the programs for one hundred times and the average costs of the one-hundred runs are presented here.

4.10.2.3 Test Results

Figure 4.62 illustrates the major performance costs, namely creation and verification of delegation tokens, involved in a single-level delegation scenario. The overall cost of creating a DToken, the major cost of the DToken delegation architecture, is roughly 1/3, 1/5, and 1/10 of that of creating a proxy certificate when the certificate key size is 512, 1024, and 2048 bits, respectively. These results demonstrate that our proposal provides significant performance gains than the proxy certificate solution. They also show that the verification cost involved in both solutions are significantly less than that of the creation cost.

The figure also indicates that the creation cost of a DToken of key size 2048 bits is comparable to that of a proxy certificate of key size 512 bits (a typical setting in the current Grid deployment environment). From a performance perspective, it suggests that it is much more affordable to employ highly secure cryptography keys (e.g. of key size 2048 bits) in the DToken architecture than in GSI. From a security point of view, the vulnerability introduced by employing the DToken architecture (i.e. all delegation chains share the long-term credentials of a principal) can be lessened by the use of highly secure keys.



	512 bits	1024 bits	2048 bits
🟦 DtokenCreate	15	22	52
🟥 DTokenVerify	4	4	5
🟩 GSICreate	48	107	602
🟪 GSIVerify	10	11	10

Figure 4.62: Overall Performance Cost Comparison between DToken and GSI (Proxy Certificate).

4.10.3 Test Unit 02: VOPS evaluation

Virtual Organisation Policy Service (VOPS) is a server serving requests to other VO services which take part in resource selection process. For such a service it is essential to provide

- proper and effective policy administration,
- effective operations in policy filtering,
- be scalable and able to serve multiple request effectively.

In this section we present VOPS scalability tests and try to substantiate VOPS as efficient entity providing upper requirements.

4.10.3.1 Responsibilities

VOPS is under development within WP3.5 under responsibility by XLAB.

4.10.3.2 Test Specification

We provide a list of tests which we have conducted with VOPS. The VOPS development went two separate but side by side implementations. VOPS in the second release of XtreemOS is implemented as a part of DIXI framework (see section 4.4: Evaluation of the DIXI Message Bus) and its performance depends mainly on the aforementioned framework. It does not implement any special database, policies are stored internally in the memory. The second implementation is using XML database implementation and can be used as a standalone service or as a part of DIXI. In this section we provide tests of the former, the later is still in development and therefore not stable and thoroughly tested.

Test Items

The focus of the tests is the main VOPS library, packaged in the `security-services-vops` package. The client-side component in XATI is packaged in the `dixi-xati` package, which provides client side methods.

Features to be Tested

We will test following features of VOPS:

- Policy insertion and deletion (locally on the same node and in two separate nodes).
- Access request time while incrementing the number of policies on the server (XATI client invoking will reside on a different node than the server).

- The invocation of the service calls defined by the service interface in an synchronous manner from client service.
- Two services, one of which will act as a server (the invoked service), another service, which will act as a client (the invokee service) and both will reside on the same node.
- Two services, one of which will act as a server (the invoked service), another service, which will act as a client (the invokee service), residing on two separate nodes.
- Loading policies from policy storage and saving the policies from storage in memory to disk.

VOPS	Description
Policy insertion	Inserting policies into VOPS policy storage
Policy removal	Removing policies from VOPS storage
Access request	Providing PDP to other VO services
Policy Save/Restore	Dumping and loading policies into the storage

Table 4.24: Summary of main features tested in VOPS provided by the VOPS server API.

Approach Refinements

Our assumption is that the system is properly installed and set up. Tests are conducted in a way that the server is placed in different situations of congestion. Various operations have been executed by the one client (repeated queries against the server), as well as the juxtaposition of multiple clients creating queries in parallel to create racing conditions on the server.

Policy insertion

- Goal of the test is to benchmark performance of the policy insertion process. We provide two benchmarks: first is running several numbers of benchmarks while inserting different number of policies on the server. Benchmarks are performed in two set-ups: client and server both reside on the same machine, client and server are on different machines.
- Hardware: workstation and a laptop, both with the same DIXI installation, second release.
- Test techniques: VOPS server running on the workstation, client running on the laptop running XATI (VOPS client side). On the workstation there were running **mpstat** and **vmstat** with a period of 1 second, which were doing benchmarking of the CPU and memory usage.

- Input parameters: generic policies, generated by testing environment written in Java.
- Metrics to be used: on the client side we are measuring time between several operations, on the server we are measuring user CPU time, system CPU time, and number of interrupts per second.

Policy deletion Is following the same techniques as described in Policy insertion section.

Access request

- Goal of the test is to benchmark performance of the VOPS' Policy Decision Point. Benchmarks are performed in two set-ups: client and server both reside on the same machine, client and server are on different machines.
- Hardware: workstation and a laptop, both with the same DIXI installation, second release.
- Test techniques: VOPS server running on the workstation, client running on the laptop running XATI (VOPS client side). On the workstation there were running **mpstat** and **vmstat** with a period of 1 second, which were doing benchmarking of the CPU and memory usage.
- Input parameters: generic policies on the server side, generated by testing environment written in Java. We also need generic user and resource certificates and job description also provided by the testing environment.
- Metrics to be used: on the client side we are measuring time between operations, on the server we are measuring user CPU time, system CPU time, and number of interrupts per second.

Policy Save/Restore This is a feature which enables VOPS to dump current storage into XML files and vise versa - to load XML policies into VOPS memory.

- Goal of the test is to benchmark performance of the VOPS policy load/write feature.
- Hardware: workstation with DIXI installation and VOPS running, second release.
- Test techniques: VOPS server running on the workstation. On the workstation there were running **mpstat** and **vmstat** with a period of 1 second, which were doing benchmarking of the CPU and memory usage.
- Input parameters: generic policies on the server side, generated by testing environment written in Java. We also need generic user and resource certificates and job description also provided by the testing environment.
- Metrics to be used: on the server side we are measuring time between operations (load policies, save policies), we are measuring user CPU time, system CPU time, and number of interrupts per second.

4.10.3.3 Test Results

Here we present test results for tests specified in former sections. We have already mentioned the two set-ups of test: local machine running both server and client, and two machines (one running VOPS server and other running client, XATI instaces). In tables 4.25 and 4.26 1 client refers to tests on local machine meanwhile 5 and 10 clients refer to tests conducted on two machines, simulating several clients.

Num. of policies	1 client	st.dev	5 clients	st.dev	10 clients	st.dev
10	1147.2	418.37	2328.2	1923.1	5118.8	1867.8
100	24824.6	4510.5	107077.4	10704.1	353193.2	1299.5
1000	922788.2	9972.6				

Table 4.25: Inserting policies into VOPS server simultaneously from different number of clients. Numbers are in milliseconds. There were totally 10 repetitions of each test performed.

Num. of policies	1 client	dev	5 clients	dev	10 clients	dev
10	1763.2	840.6	1376.6	447.4	2639.4	665.86
100	35099.4	306.2	112928.0	12724.2	336516.6	1280.84
1000	930784.6	19206.1				

Table 4.26: Deleting policies from VOPS server simultaneously from different number of clients. Numbers are in milliseconds. There were totally 10 repetitions of each test performed

In tables 4.25 and 4.26 we conclude, that comparing times to insert and delete policies is linear (approximately linear). Inserting 1000 generic policies into VOPS database from a client takes about 37.1 times more time than inserting 100 generic policies (comparing 100 to 10 policies it makes 21.6 times). Similar behaviour is with deleting policies.

We have also measured other metrics:

- CPU utilization that occurred while executing at the user level (user),
- CPU utilization that occurred while executing at the system level (this does not include time spent servicing interrupts or softirqs) (sys),
- percentage of time that the CPU or CPUs were idle and the system did not have an outstanding disk I/O request (idle),
- total number of interrupts received per second by the CPU or CPUs (intr/s).

Measurements are showed in figure 4.63 while inserting and deleting 1000 policies on one machine (server and client running on one machine). From time to time we can see decrease in user CPU time and an increase in idle time. This happens while the number of policies stored in the policy storage is decreasing/increasing. Period between decrease/increase in figure 4.63 takes around 2000 seconds (1000 for insertion and 1000 for deletion, see tables 4.25 and 4.26).

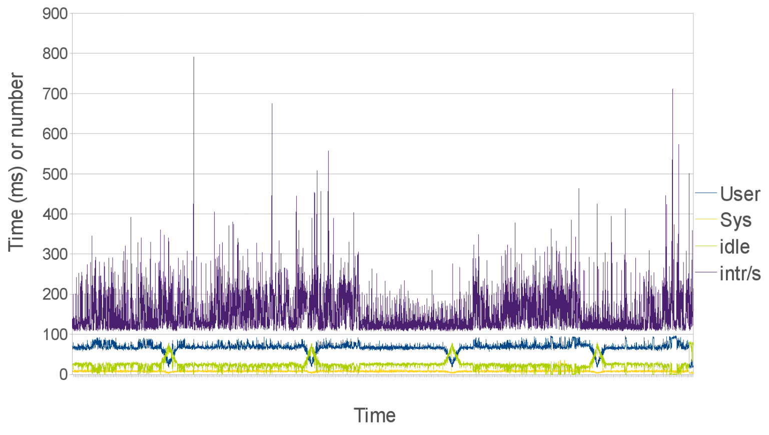


Figure 4.63: user CPU time, system CPU time, idle CPU time and number of interrupts per second obtained with *mpstat* while repeatedly inserting and deleting 1000 policies (x axis presents seconds, the whole axis presents 8065 seconds)

Figure 4.64 depicts memory consumption while inserting and deleting 100 policies. This experiment has been conducted on two machines with running server on one workstation and client on different machine. It is expected that the memory usage is greater while entering new policies than while deleting policies from the database (memory). Deleting policies is releasing memory since the garbage collector frees dynamically allocated memory (policies in the memory). Test was running for 67 seconds, in first 31 seconds 100 policies were inserted into database from a single client. In next 36 seconds these policies were deleted.

Num. of policies	Time (ms)
100	615
500	3587
1000	15288

Table 4.27: Time per request when number of policies on the server incrementally increases.

In figure 4.65 we see time to evaluate a request towards VOPS Policy Decision Point depending on the number of policies residing in the policy storage already. Time is computed as an average time of 10 access requests containing 10 generic

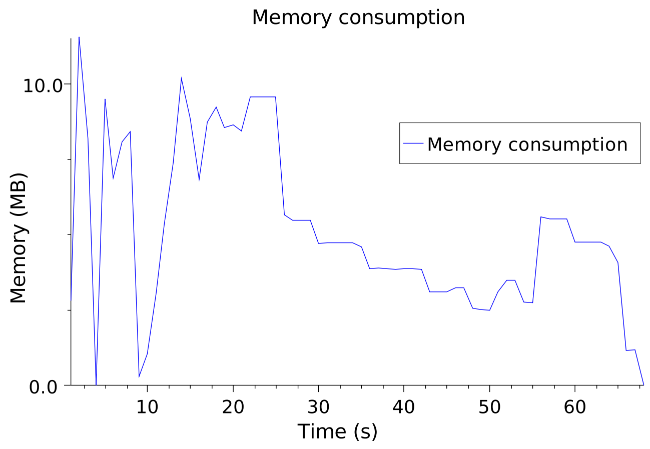


Figure 4.64: Figure memory consumption while running 100 policy insertions (first 31 seconds) and deletion (next 36 seconds) respectively from a single client.

resources and job description. In table 4.27 we can see some actual figures taken from the benchmark.

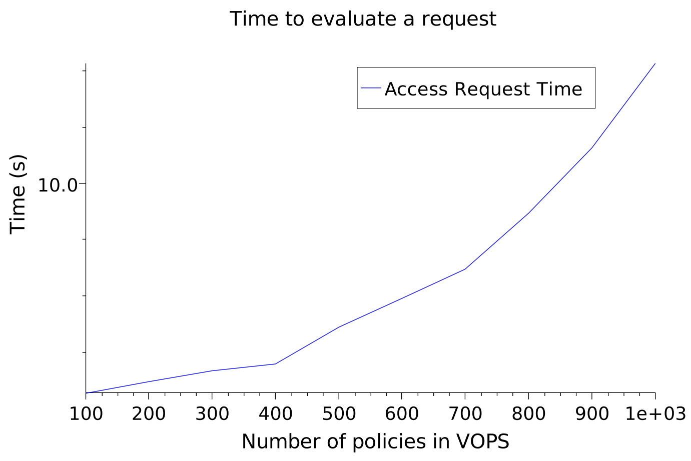


Figure 4.65: Time per request when number of policies on the server incrementally increases.

We have conducted test for loading and saving different number of policies on the server. Table 4.28 gives us figures for different number of policies.

Num. of policies	Time to load (ms)	Time to save (ms)
10	234	196
100	832	789
1000	6271	7576

Table 4.28: Time for an operation of loading and saving specified number of policies into and from storage respectively.

In figures 4.66 and 4.67 memory consumption and CPU utilization with the same testing environment as in the tests before are depicted respectively. In figure 4.66 memory consumption is showed from starting the VOPS server. The reason for larger consumption in the first 20 seconds is loading of the JVM and starting the server in Java environment. Afterwards, during saving and reloading the 1000 policies, some variations can be observed. The reason is manly working of the Java's garbage collector, nevertheless memory consumption, while reloading the policy storage, stays in the same interval through time.

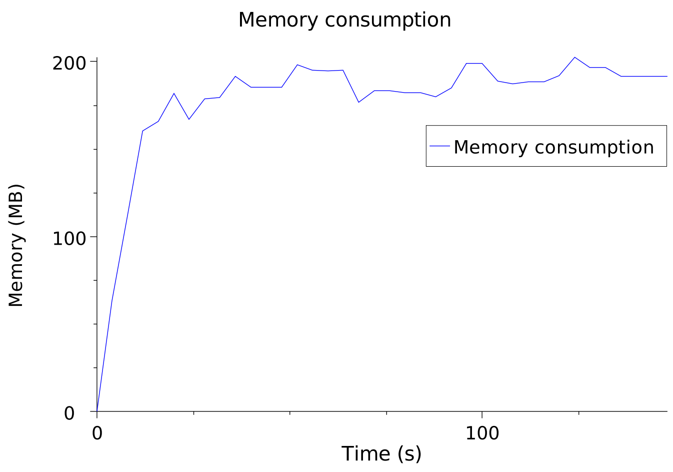


Figure 4.66: Memory consumption from starting a VOPS server (first 20 seconds), after that 10 iterations of reloading policy storage occurred.

In figure 4.67 CPU utilization, while performing saving and restoring the policy storage, is depicted. This test has also been conducted on two machines, one triggering the save/restore operations, other machine running the server. 10 repetitions of operations with 1000 policies are performed through the time of 151 seconds. First 20 seconds CPU is occupied for the reason of loading the server and

JVM. After loading the server, periodic CPU consumption can be seen and these operations are triggering periodically:

- *PolicyFactory.initializePDP()*,
- *PolicyFactory.updatePolicyStorage()*.

In first operation, Policy Decision Point, is initialized from the policy storage as defined in VOPS configuration file. In our example policy storage points to a directory of 1000 generic policies. This way 1000 generic policies are stored into the memory and decision point is initialized with these policies. In the second operation, *PolicyFactory.updatePolicyStorage()*, policies are copied back on the disk storage (into policy storage as defined in VOPS configuration file) and afterwards first operation is triggered again. This way CPU occupation is larger as expected and similar samples of CPU utilization can be seen afterwards.

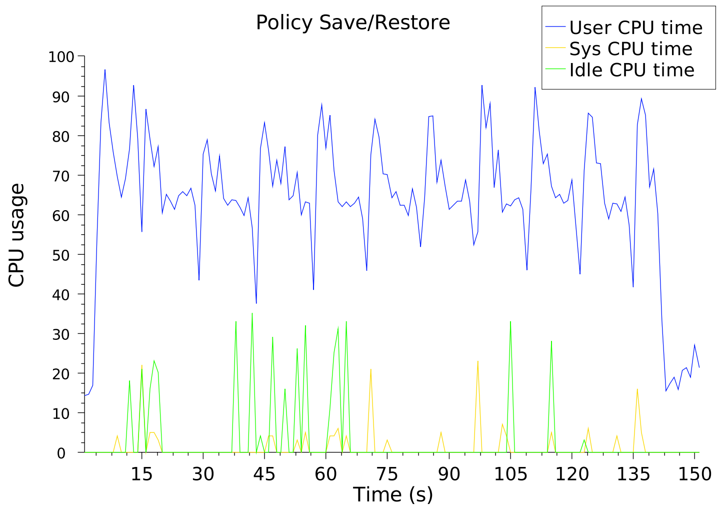


Figure 4.67: CPU utilization while performing save/restore operation on VOPS server. 10 repetitions of loading policies from XML file into memory and saving policies back to XML files are depicted.

4.10.4 Test Unit 03: Evaluation of CDA Server

4.10.4.1 Responsibilities

We describe here a test carried out within WP3.5 by STFC recording the time taken by the CDA server to process 100 requests by the "get-xos-cert" command for an XOS-Certificate.

4.10.4.2 Test Specification

The CDA server, version 0.2.4, runs on a node configured with the "task-xtreemos-coreservices" package. The hardware is an Intel dual core processor running at 2.40GHz with 3GB RAM. XtreemOS 1.0 is installed directly onto this machine, with the latest patches applied. There is no virtualisation software used.

Test Items

This test unit sends a large number of requests from a client node running the CDA client to the CDA server.

Features to be Tested
The overall time taken by the CDA server to satisfy these requests is recorded. The CPU load and memory utilisation of the CDA server while serving these requests is recorded.

Features not to be Tested
The response time to individual CDA client requests is not recorded. The CPU load and memory utilisation on the CDA client node is not recorded.

Configuration

1. XtreemOS must be installed and configured on the two test nodes. One node is set up as a core node with the CDA server installed. The other node is set up as a client node.

Test method The following steps are executed during the test:

1. On the core node, the "dstat" command is started. The invocation is "dstat -tcm 5 > file.out". This records CPU load and memory utilisation figures every 5 seconds, along with a timestamp.
2. Wait for two minutes before starting the test from the client node.
3. On the client node, the "get-xos-cert" command is executed 100 times in a shell "for" loop, under the control of the "/usr/bin/time" command.

4. Wait until this “for” loop has finished. The time taken is recorded.

5. Wait for two minutes.

6. On the core node, the dstat command is stopped and the output file preserved.

4.10.4.3 Test Results

The test was executed by Ian Johnson, STFC on 03-12-2009. The times taken on the CDA client node for three successive runs of the test are shown in the table below:

Run no.	Num Requests	Elapsed time (s)
1	100	101
2	100	97
3	100	99

The parameters measured on the CDA server for one of these test runs is shown in the graph below:

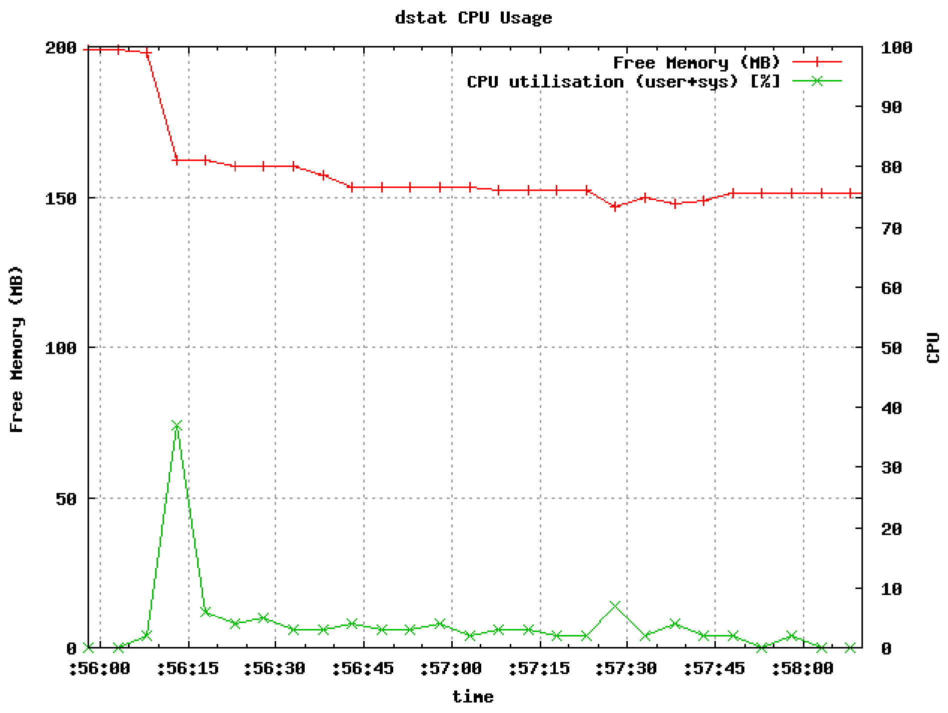


Figure 4.68: CPU load (right axis) and reduction in free memory (left axis) while CDA server is processing 100 requests.

Procedure Results The CDA client node was able to send 100 requests to the CDA server in around 100s. This is in the expected range, as the CDA client command "get-xos-cert" takes around 1 second to execute. The "get-xos-cert" command is loaded and initialised every for every iteration of the invoking "for loop".

The CDA server CPU load and reduction in free memory during a test run are shown in figure 4.68. (The reduction in free memory corresponds to an increase in used memory.)

Before the test was started, CPU utilisation was around 3%. The CPU load while the test was running was around 8%, apart from an initial spike of around 40%, which may be associated with the CDA server servicing the first request (involving loading classes and making the initial database connections). This represents an additional CPU load of 5% due to the CDA server processing a single stream of requests.

The amount of free memory before the test was around 200MB. During the test, this reduced to around 150 MB. Hence, operation of the CDA server during this test consumed about 50MB of extra memory.

4.10.5 Test Summary Report

4.10.5.1 Summary of Tests and Results

The goal of the security tests was to make sure that the security services do not become bottlenecks and unjustified overheads for the other features of XtreemOS. Moreover, it is critical to ensure that security services do not introduce security problems for the overall system. Firstly, the DTokens used for delegation proved to be more efficient than the standard proxy certificates used by more established Grid infrastructures. Delegation is an important feature in any outsourced, service oriented (*aaS) model of computation, as the owner of processes and data has to delegate privileges to the owner of resources that executes the processes and manipulates/stores data. Secondly, the VOPS has linear performance with regards to the number of simultaneous policies that can be handled. Finally, the impact on CPU and memory of operating the CDA server was also tolerable, with consumption between 3 and 8% during the servicing of 100 sequential requests. We do not yet have reportable data giving an indication of the scalability of the CDA server.

4.10.5.2 Conclusion and Directions for Future Work

Further evaluation and ongoing improvement of the security services can only be accomplished through the use of rigorous, runtime vulnerability assessment. There is still some work to be done in the area of managing the processes of issuing and updating policies and certificates. In addition, there is more scenario-based testing to be done of the VOPS, where different cases like monitoring and usage control will be investigated. This work will be done in conjunction with the demonstrator activities.

4.11 Evaluation of Mobile Device Flavor

The component to be evaluated in this case is XtreemOS-MD, the XtreemOS Mobile Device flavor, let's say, the XtreemOS version for MDs. XtreemOS-MD software includes:

- XtreemOS-MD F-layer, for VO support in Mobile Devices.
- XtreemOS-MD G-layer, including three main services: XtreemFS, AEM, and CDA.
- The IMA and JobMA applications will be used for testing, but are not part of XtreeemOS-MD software.

While the mentioned applications (IMA and JobMA) are described by this work package (WP4.2), the use cases, architecture and features are described by WP2.3 (focused on XtreemOS-MD F-layer) and WP3.6 (focused on XtreemOS-MD G-layer). In every case, TID is responsible for the components, being also WPLeader of the WPs in charge of the different parts of XtreemOS-MD software.

4.11.1 Test Plan

4.11.1.1 Responsibilities

WP2.3, WP3.6, WP4.2 and TID as partner, are responsible for defining and executing the tests related to XtreemOS-MD and the reference applications associated (IMA and JobMA)

4.11.1.2 Test Items

The software to be tested is XtreemOS-MD. The current version of the software (as of the time of writing this test plan) is XtreemOS-MD Release 1.0, from April '09. Source and documentation are available on the internal XtreemOS SVN. IMA and JobMA applications are still not released with XtreemOS-MD, but the development versions could be used to run the tests if final versions of those applications are not ready at the time agreed for testing. The advanced features of XtreemOS-MD that will be included in next releases at that time will also be tested.

4.11.1.3 Features to be Tested

The following features will be tested:

- XtreemOS support in PDA and mobile phones (ARM arquitectures).
- VO support of XtreemOS-MD.
- Lightweight security support of XtreemOS-MD (considered also as a performance test).

- XtreemOS-MD support of main services (XtreemFS, AEM, CDA).
- Typical application support including Grid features (the application to use will be IMA).
- Support of specific applications created inside XtreemOS project (the application in this case will be JobMA).
- Performance of AEM and XtreemFS in XtreemOS-MD.

4.11.1.4 Features not to be Tested

- Mobility and behavior when connectivity is lost
- Resource sharing and context awareness support

These features will be tested on later versions of the software, as those are not requirements of the basic version, but the XtreemOS-MD advanced version, currently being implemented. Other proposed requirements like Java or web service support will not be tested, as they are not natively supported by the terminals planned to be used for testing. Those requirements will be considered as not applicable.

4.11.1.5 Overall Approach

The purpose of these tests is to evaluate the current version of Mobile-Device software (1.0), provide feedback to developers, and improve the next version which fulfills the requirements listed in XtreemOS deliverable D4.2.5

We will thus focus on evaluating the higher-level design, features and usability of each module rather than bugs in the implementation. Some additional performance tests will be executed, in order to compare XtreemOS-MD and the XtreemOS PC client version and also to demonstrate the benefits of the use of Grid services from mobile devices respect to the alternative of not using a Grid (in this case, there are no competitors to compare to, as the access to Grid services from mobile devices, without using portal solutions, is something not covered yet in the market)

The tests will be done on a Nokia N8x0 PDA and could also be passed on a PC using Qemu to emulate an Angstrom platform, also supported by XtreemOS-MD. Installation of XtreemOS-MD versions is really simple, so there is no need to describe the installation procedure here. Nevertheless, in case of any modification needed with respect to the installation instructions supplied with the software, a detailed description reporting those modifications will be included in the test documentation.

The test preparation requires the following tasks:

1. Install the software XtreemOS-MD on the terminal.
2. Install the IMA and JobMA applications that will be used for testing purpose.

Once installed the mentioned software, each test then consists of performing the test, and documenting the full procedure. The individual tests can be executed in any order, but the order of tasks during preparation and during execution of tests must be as given here.

4.11.2 Test Unit 01: XtreemOS support on ARM architectures

4.11.2.1 Responsibilities

WP2.3 and WP3.6, and TID as partner, are responsible for definition and execution of this test

4.11.2.2 Test Specification

Test Items
XtreemOS-MD software is in this case the item tested. XtreemOS-MD release 1.0 could be downloaded from the project repository (an advanced version will be released together with XtreemOS release 2.1). The users guide and installation guide could also be found in the project repository.

Features to be Tested
This test case tests the support of XtreemOS-MD on ARM architectures, the typical used for PDAs and mobile phones. The test will basically verify the installation, configuration and first use of XtreemOS-MD in one initially-clean MD.

Approach Refinements

The goal of this test is to verify that XtreemOS-MD software is supported by ARM architecture devices. A Nokia N800 will be used for this test. No need of special software pre-installed in the Nokia N800. The installation of XtreemOS-MD will be part of the test, so this test case is also valid for installation testing purposes.

The input of this test case consists of the XtreemOS-MD software that will be installed in the mobile device.

A command `uname -a` will be executed as part of the test and the output should be the usual one showing info about the system where the request is being executed (name, OS, etc.). When executing this first command, the password of the user configured by the installation will be requested.

Then the command `date` will be executed and the current date and time will be shown. In this case, the user's password will not be requested again, as the certificate should have been already created in the previous step.

4.11.2.3 Test Results

The procedure was run following this procedure:

Installation

1. First, we should open the website where XtreemOS-MD software can be downloaded.
2. Then, we will proceed to the download and installation of XtreemOS-MD just clicking on the correspondent link.

Set Up No special configuration after the installation is needed.

Start The test is run by typing the command `uname -a` and then the command `date`. In the first operation, the user's password should be requested to generate a new certificate. For the second operation, it should do it directly without any furthers user's interaction, as the credential should be already stored.

Procedure Results The installation of XtreemOS-MD from the web server has been done succesfully. Then we have verified the installation following the procedure steps described previously and the commands have been executed correctly, requesting the user's password to generate the certificate just the first time.

4.11.3 Test Unit 02: VO support by XtreemOS-MD

4.11.3.1 Responsibilities

WP2.3 and TID as partner, are responsible for definition and execution of this test

4.11.3.2 Test Specification

Test Items
XtreemOS-MD software is in this case the item tested. XtreemOS-MD release 1.0 could be downloaded from the project repository (an advanced version will be released together with XtreemOS release 2.1). The users guide and installation guide could also be found in the project repository.

Features to be Tested
This test case will test the management of VOs using VOLife from the mobile device. Only authorized users shall be able to manage VOs from MDs.

Approach Refinements
The goal of this test is to verify that it's possible to manage VOs using VOLife from a mobile devices. A Nokia N800 with XtreemOS-MD pre-installed will be used for this test.

This test requires as input a user certificate to authenticate the user in the Grid and the output of the test will be the VOLife GUI for managing the VOs.

4.11.3.3 Test Results

The procedure was run following this procedure:

Installation XtreemOS-MD software should be previously installed in the terminal used, but no other installation is requiered for this test case.

Set Up No special configuration needed.

Start The test is run by connnecting to VOLife web interface from the mobile device (the Nokia N800 in this case), trying the different links and verifying that the GUI is correctly displayed in the terminal screen.

Procedure Results The pre-installation of XtreemOS-MD was done successfully. The navigation throw VOLife GUI from the Nokia N800 has been successful, and the interface is correctly displayed and adapted to the device screen.

4.11.4 Test Unit 03: Lightweight security for mobile devices

4.11.4.1 Responsibilities

WP2.3, WP3.6, WP4.2 and TID as partner, are responsible for definition and execution of this test.

4.11.4.2 Test Specification

XtreemOS-MD software is in this case the item tested. XtreemOS-MD release 1.0 could be downloaded from the project repository (an advanced version will be released together with XtreemOS release 2.1). The users guide and installation guide could also be found in the project repository. Also CDAProxy, included as part of XtreemOS release (For more information: https://gforge.inria.fr/plugins/scmsvn /viewcvs.php/*checkout*/distribu
tion-mobile/doc/XtreemOSMDUserDocumentation.pdf?rev=2712&root=xtreemos
), will be needed in this test.

Features to be Tested
This test case tests the benefits of using a CDAProxy to obtain a new certificate instead of doing it directly contacting the CDAServer. This is related with the requirement of having some light security on Mobile Devices, taking into account their CPU limitations and the high cost of resources associated to the processes for generating a certificate. This requirement is fulfilled with the inclusion of the CDA-Proxy in the XtreemOS-MD distribution.

The test will compare the time spent for the operation of generating a new certificate when using CDAProxy and when contacting directly the CDA.

Approach Refinements
The goal of this test is to demonstrate the benefits of CDAProxy. A Nokia N800 with XtreemOS-MD pre-installed will be used for this test.

There are no special input parameters needed and the output of the test will be the time spent in generating a certificate in each case.

4.11.4.3 Test Results

The test was run following this procedure:

Installation XtreemOS-MD software should be previously installed in the terminal used, but no other installation is requiered for this test case.

Set Up No special configuration needed.

Start Configure XtreemOS-MD to contact directly the CDA server. Obtain a new certificate and take note of the time spent in the operation. Repeat the process at least 3 times to obtain an average time spent on the operation.

Configure XtreemOS-MD to contact the CDA through a CDAProxy that will be already running. Obtain a new certificate and take note of the time spent in the operation. Repeat the process at least 3 times to obtain an average time spent on the operation.

Compare the average times in each case.

Procedure Results The commands used for measuring the time for obtaining a new credential contacting directly with the CDA or through the CDAProxy are `time xos_getdumpcred cdabench` and `time xos_getdumpcred cdaproxybench` respectively.

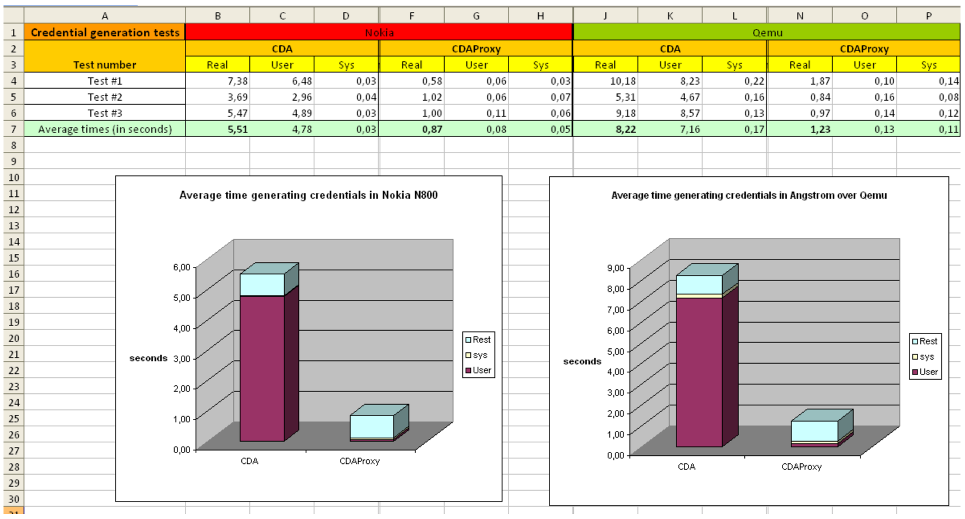
We have executed 3 series of requests executing the commands in the previous order. It's important to alternate the obtainment of the credential with and without the CDAProxy, in order to force the generation of a new credential each time (as the command `xos_getdumpcred` will obtain the credential from the cache if possible, which is not what we want to test).

The tests have been executed directly on a Nokia N800 and also on a PC running Qemu to emulate Angstrom, also supported by XtreemOS-MD.

An additional set of tests from the Nokia N800 have been carried out, to obtain a value significant from a statistic point of view

The results obtained are shown in the next figures, where a table shows the different times for obtaining the credential in each test, including also a graphic to make more clear the advantage of using CDAProxy, reducing the average time for obtaining the credential from more than 6 seconds (9 seconds in Angstrom over Qemu) to around just one second.

Additionally, there are the figure 4.69 and 4.70 including a graphical representation of a large series of tests comparing the time for obtaining a credential with and without the CDAProxy from the Nokia N800



Credential generation tests	Nokia						Qemu					
	CDA			CDAProxy			CDA			CDAProxy		
Test number	Real	User	Sys	Real	User	Sys	Real	User	Sys	Real	User	Sys
Test #1	7,38	6,48	0,03	0,58	0,06	0,03	10,18	8,23	0,22	1,87	0,10	0,14
Test #2	3,69	2,96	0,04	1,02	0,06	0,07	5,31	4,67	0,16	0,84	0,16	0,08
Test #3	5,47	4,89	0,03	1,00	0,11	0,06	9,18	8,57	0,13	0,97	0,14	0,12
Average times (in seconds)	**5,51**	4,78	0,03	**0,87**	0,08	0,05	**8,22**	7,16	0,17	**1,23**	0,13	0,11

Figure 4.69: CDAProxy comparison

4.11.5 Test Unit 04: Performance comparison with XtreemOS PC flavor and no-Grid solutions

4.11.5.1 Responsibilities

WP2.3 and WP3.6, and TID as partner, are responsible for definition and execution of this test

4.11.5.2 Test Specification

Test Items

XtreemOS-MD software is in this case the item tested. XtreemOS-MD release 1.0 could be downloaded from the project repository (an advanced version will be released together with XtreemOS release 2.1). The users guide and installation

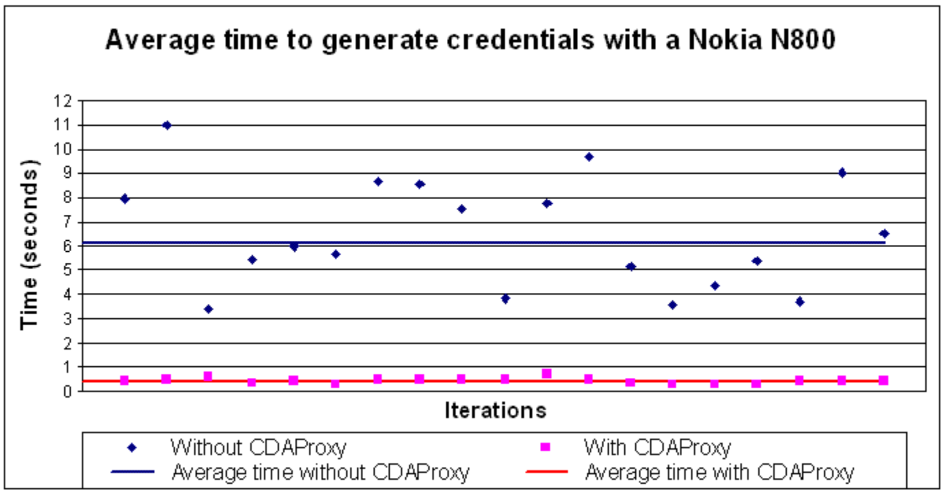


Figure 4.70: CDAProxy comparison results

guide could also be found in the project repository.
For this test, XtreemOS-MD will be installed in a mobile device: Nokia N800. Also, a XtreemOS PC client and a Core included as part of XtreemOS 1.0 release, will be needed for this test. Both nodes (core and client) are running as VMware virtual machines on top of a Pentium IV machine.

Features to be Tested
This test case checks the performance of XtreemOS-MD compared to XtreemOS PC flavor and direct operation on the MD. Some commands will be executed from an MD using and not using XtreemOS-MD and from a PC connecting to a XtreemOS node respectively. The same node will be used for every subtest. Those tests will be based on some simple jobs executions (to test AEM) and file system operations (to test XtreemFS), and finally a video conversion job, allowing us to test simultaneously the performance of AEM + XtreemFS in a more useful scenario.

Approach Refinements
The goal of this test is to demonstrate the benefits of using the Grid through XtreemOS-MD from a mobile device, and also the fact that the performance of XtreemOS-MD as a client is similar to the one offered by the XtreemOS PC client.

Some scripts for executing a simple job and for executing usual file system operations (writing and removing operations) will be used for these tests. Finally, some video files of different size will be used as input for the final test about video conversion.

The time spent for the different operations (simple job execution and video conversion), the number of bytes written for writing test and the number of files created and removed in a concrete period of time will be the output of each of these subtests, generating finally some comparison graphics. The shown data in these graphics are the result of a statistic process consists of average of the obtained values in different repetitions (10) for each subtests.

4.11.5.3 Test Results

The test was run following this procedure:

Installation
XtreemOS-MD software should be previously installed on the Nokia terminal used, and also an XtreemPS client on a PC.

Start
The command used for these tests is `xsub -vf conversion.jsdl`, where `conversion.jsdl` defines a job that converts an AVI video stored in the XtreemFS and stored also in the XtreemFS the resulting converted video (optimized for a Nokia N800 terminal).
We will convert 5 video files between 3 and 30 MB. The tests will be executed first in the PC (testing performance in XtreemOS-PC flavour) and then on the Nokia (XtreemOS-MD client). In both cases the same node in the Grid will be used for executing the conversion.

Procedure Results
The results obtained are presented in the figure 4.71, where a graphic shows the different times for video conversion depending on the size of the input file, it shows clearly that the result are similar using XtreemOS and XtreemOS-MD. The obtained results take into consideration the conversion time plus the transfer to the XtreemFS volume of the user. It serves us to conclude that the performance of AEM and XtreemFS is independent of the client used, either it's XtreemOS or XtreemOS-MD.

- `Nokia` label identifies the data set related to the video conversion on a Nokia device directly (without any XtreemOS flavor installed) using its own file system and processor.
- `XOS-PC` label identifies the data set related to the video conversion using the Grid through XtreemOS client(PC flavour)
- `XOS-MD` label identifies the data set related to the video conversion using the Grid through XtreemOS-MD client installed on a Nokia device.

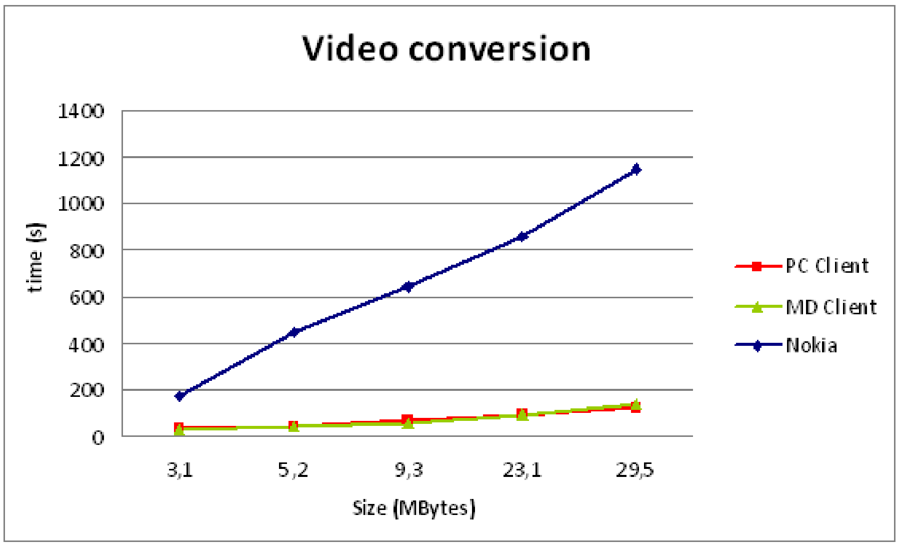


Figure 4.71: Video conversion performance

Anomalous Events
It seems that XtreemFS is a bit unstable in XtreemOS-MD 1.0, the version used for these tests, as there were some problems forcing us to restart the system when converting very heavy video files.

File creation performance
We will execute a bash script creates empty files in the XtreemFS volume. The test will be executed first in the PC (testing performance in XtreemOS-PC flavour) and then on the Nokia (XtreemOS-MD client). In both cases the same node in the Grid will be used for executing the test. Moreover, the test will be repeated on the Nokia device without XtreemOS in order to get the performance of the Nokia native file system and to compare it with the two previous cases. The results are shown in the figure 4.72

- `Nokia` label identifies the data set related to execution of the test on a Nokia device directly (without any XtreemOS flavor installed) using its own file system.
- `XOS-PC` label identifies the data set related to execution of the test using the XtreemFS system through XtreemOS client(PC flavour)
- `XOS-MD` label identifies the data set related to execution of the test using the XtreemFS system through XtreemOS-MD client installed on a Nokia device.

File removal performance
We will execute a bash script removes all empty files (created in the previous sub-

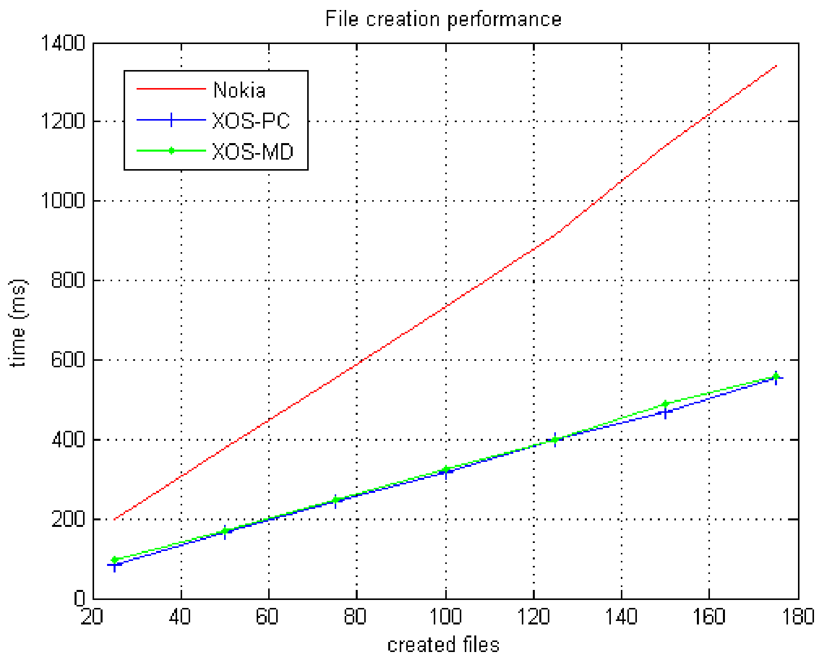


Figure 4.72: File creation performance

test) in the XtreemFS volume. The test will be executed first in the PC (testing performance in XtreemOS-PC flavour) and then on the Nokia (XtreemOS-MD client). In both cases the same node in the Grid will be used for executing the test. Moreover, the test will be repeated on the Nokia device without XtreemOS in order to get the performance of the Nokia native file system and to compare it with the two previous cases. The results are shown in the figure 4.73

- `Nokia` label identifies the data set related to execution of the test on a Nokia device directly (without any XtreemOS flavor installed) using its own file system.
- `XOS-PC` label identifies the data set related to execution of the test using the XtreemFS system through XtreemOS client(PC flavour).
- `XOS-MD` label identifies the data set related to execution of the test using the XtreemFS system through XtreemOS-MD client installed on a Nokia device.

Writing operation performance We will execute a bash script writes information (zeros), in a text file of the XtreemFS volume, in a concrete period of time. The test will be executed first in the PC (testing performance in XtreemOS-PC flavour) and then on the Nokia (XtreemOS-MD client). In both cases the same node in the Grid will be used for executing the conversion. Moreover, the test will

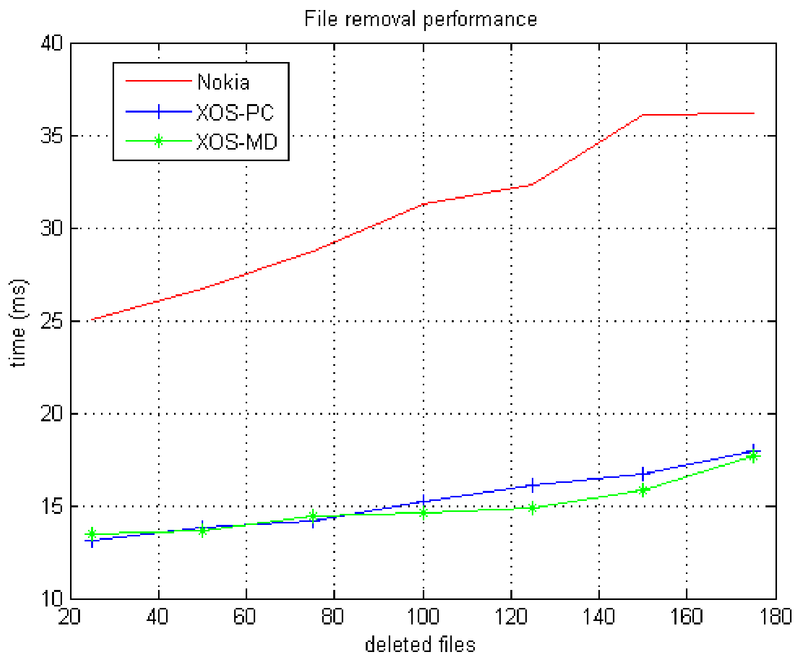


Figure 4.73: File removal performance

be repeated on the Nokia device without XtreemOS in order to get the performance of the Nokia native file system and to compare it with the two previous cases.

The results are shown in the figure 4.74

- `Nokia` label identifies the data set related to execution of the test on a Nokia device directly (without any XtreemOS flavor installed) using its own file system.
- `XOS-PC` label identifies the data set related to execution of the test using the XtreemFS system through XtreemOS client(PC flavour).
- `XOS-MD` label identifies the data set related to execution of the test using the XtreemFS system through XtreemOS-MD client installed on a Nokia device.

Note that the shown data in these graphics are the result of a statistic process consists of average of the obtained values in different repetitions(10) for each subtests.

Each previous subtests result serves us to conclude that the performance of AEM and XtreemFS is independent of the client used, either it's XtreemOS or XtreemOS-MD. In addition, the inclusion of the executed tests directly in the Nokia and their results permit us to remark the benefits (in terms of performance) of using grid solutions, as XtreemOS-MD, instead of native executions (directly in the Nokia hardware) for operations on the mobile.

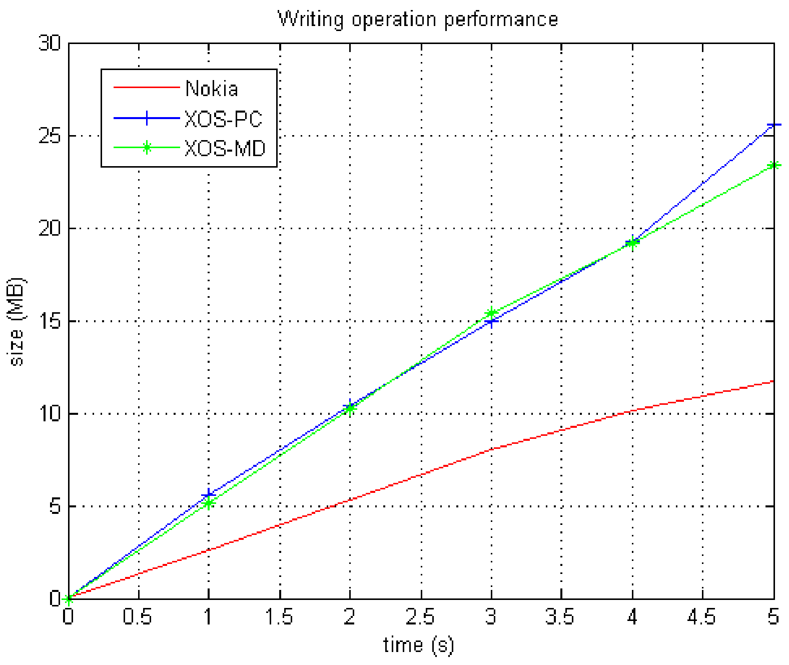


Figure 4.74: Writing operation performance

4.11.6 Test Unit 05: Creation of new jobs using JobMA application

4.11.6.1 Responsibilities

WP3.6 and WP4.2, and TID as partner, are responsible for definition and execution of this test

4.11.6.2 Test Specification

Test Items
JobMA application in in this case the item tested. As the rest of the reference applications, it's not included at this moment as part of any XtreemOS release, but, it's possible to be downloaded from the project repository.

Features to be Tested
This test case tests the creation of new jobs using the JobMA application

Approach Refinements
The goal of this test is to demonstrate that it's possible to create new jobs using the GUI offered by the JobMA application. A Nokia N800 with XtreemOS-MD and JobMA applications pre-installed will be used for this test.

This test requires a JSDL file where a job (that will be created) is described.

We will check that the JobMA GUI works correctly and that it's possible to load JSDL files from JobMA.

4.11.6.3 Test Results

The test was run following this procedure:

Installation Once installed XtreemOS-MD on the mobile device used, JobMA application should also be installed in the terminal following the installation and configuration instructions.

Set Up No special configuration after the installation is needed.

Start Once JobMA application is installed, the next step is to start the JobMA. To do this, type the next sentence in a X-Terminal:

```
# jobma
```

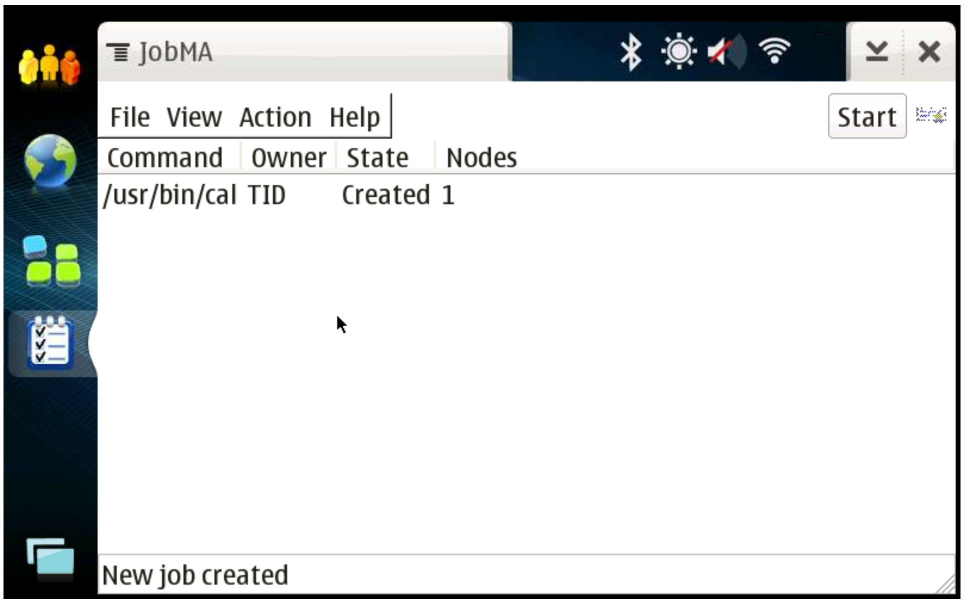
A new window will appear in the terminal, requesting the CDA password, which the user has in the VOlife. Then, the user has to click on the "OK" button in this window to send the password.

After introducing the password, the JobMA user interface will appear on the X-terminal (not needing to load manually the certificate in this version)

Load jsdl It's needed to load the job that the user wants to create. This job should be in a .jsdl file and, to load it, is necessary to select the "File" tab in the menu and then click on the "Open JSDL file..." option.

A new window will appear, where the user has to browse to select the job and, after clicking on the "Open" button, a new message will appear at the bottom of the JobMA window informing that the new job has been created automatically if the load is successful.

Procedure Results The execution of this test was successful. The JobMA GUI was correctly presented and a JSDL file was loaded using it.



```
Command        Owner  State    Nodes
/usr/bin/cal   TID    Created  1
```

New job created

4.11.7 Test Unit 06: Defining new jobs using JobMA application

4.11.7.1 Responsibilities

WP3.6 and WP4.2, and TID as partner, are responsible for definition and execution of this test

4.11.7.2 Test Specification

Test Items
JobMA application in in this case the item tested. As the rest of the reference applications, it's not included at this moment as part of any XtreemOS release, but, it's possible to be downloaded from the project repository.

Features to be Tested
This test case tests the definition of new jobs using the JobMA application

Approach Refinements
The goal of this test is to demonstrate that it's possible to define new jobs using the GUI offered by the JobMA application. A Nokia N800 with XtreemOS-MD and JobMA applications pre-installed will be used for this test.

This test requires that the user introduces some fields to generate the JSDL file that will create the job

We will check that the JobMA GUI works correctly and that the .jsdl file is automatically created when the job is defined filling the needed fields in the window opened by the JobMA GUI.

4.11.7.3 Test Results

The test was run following this procedure:

Installation Once installed XtreemOS-MD on the mobile device used, JobMA application should also be installed in the terminal following the installation and configuration instructions.

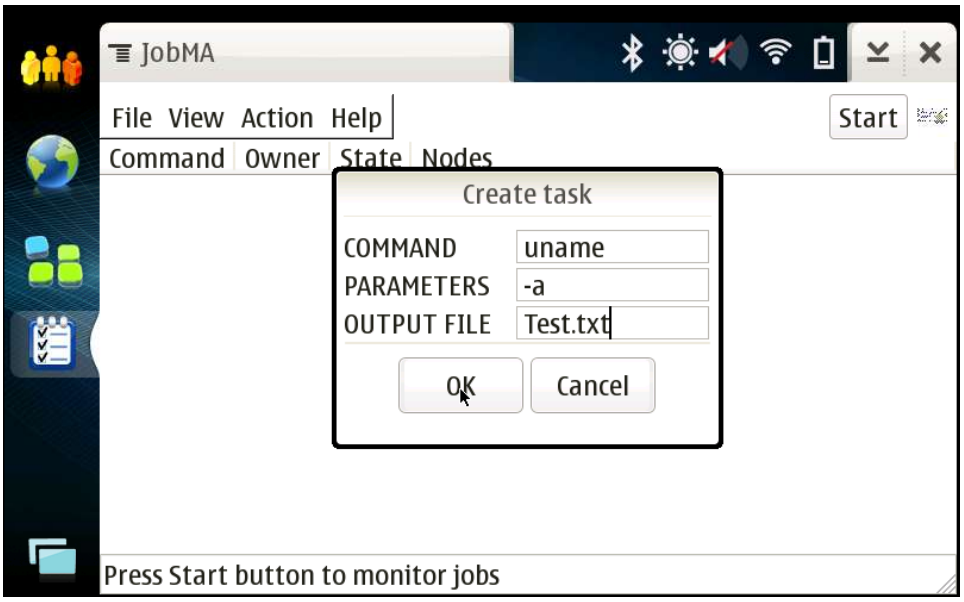
Set Up No special configuration after the installation is needed.

Define job Once installed XtreemOS-MD and JobMA application in our mobile device, and after launching this JobMA application, user can define a job that will be created. To do this, user has to do click on the "File" tab in the top menu and then on the "Define job" option. After this, a new window will appear requesting three fields to fill:

- Command: is the name of the job we want to create. E.g: "cal" is the command to create a calendar.
- Parameters: is the specification that we want to add to the job. E.g: "-3" in the previous command will show the three last months.
- Output file: is the file in which the job will be stored once it will be created and run.

After clicking on the "OK" button, the job will be created and a new message will appear at the bottom of this JobMA window ("New job created"), also the column "State" is set to "Created", explaining that the job has been created succesfully.

Procedure Results The execution of this test was successful. The JobMA GUI was correctly presented and a JSDL file was loaded using it.



COMMAND uname
PARAMETERS -a
OUTPUT FILE Test.txt

4.11.8 Test Unit 07: Using JobMA for monitoring jobs

4.11.8.1 Responsibilities

WP3.6 and WP4.2, and TID as partner, are responsible for definition and execution of this test

4.11.8.2 Test Specification

Test Items
JobMA application in in this case the item tested. As the rest of the reference applications, it's not included at this moment as part of any XtreemOS release, but, it's possible to be downloaded from the project repository.

Features to be Tested
This test case tests the monitoring of jobs using the JobMA application

Approach Refinements
The goal of this test is to demonstrate that it's possible to monitor the current jobs using the GUI offered by the JobMA application. A Nokia N800 with XtreemOS-MD and JobMA applications pre-installed will be used for this test.

We will check that the JobMA GUI works and that the list of works in correctly shown.

4.11.8.3 Test Results

The test was run following this procedure:

Installation Once installed XtreemOS-MD on the mobile device used, JobMA application should also be installed in the terminal following the installation and configuration instructions.

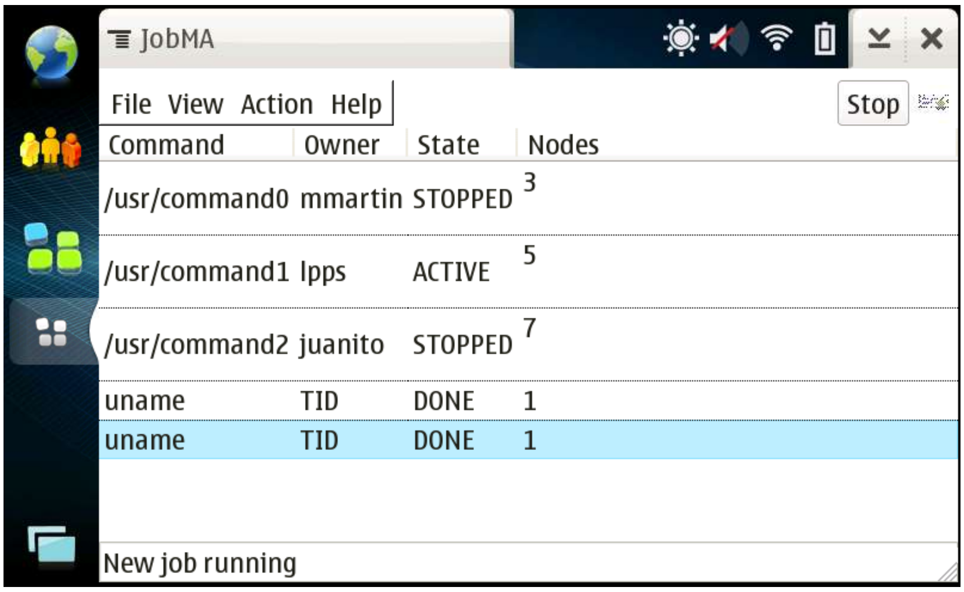
Set Up No special configuration after the installation is needed.

Start Once installed XtreemOS-MD and JobMA application in our mobile device, and after launching this JobMA application, user can consult all the existing jobs in the Grid. To do this, user has to do click on the "Start" button in the top right corner of the JobMA window. Note that in the JobMA window will appear a message at the bottom explaining that in order to monitor jobs, user has to press "Start" button.

After clicking on this button, a job list will be shown, displayed in rows, one for each job. The number of columns can be different depending on the options selected by the user. This is, in the "View" tab in the menu, users can select the information about the job that they want to know. In this tab, users can consult the "Job ID", "Command", "Name", "Owner", "State", "Sub. time" and the "Nodes" related to each job.

The "Start" button is as well replaced by a "Stop" button, that will serve to stop monitoring the jobs.

Procedure Results The execution of this test was successful. The list of jobs was correctly presented using the JobMA GUI.



Command	Owner	State	Nodes
/usr/command0	mmartin	STOPPED	3
/usr/command1	lpps	ACTIVE	5
/usr/command2	juanito	STOPPED	7
uname	TID	DONE	1
uname	TID	DONE	1

New job running

4.11.9 Test Unit 08: Using JobMA for viewing info about a job

4.11.9.1 Responsibilities

WP3.6 and WP4.2, and TID as partner, are responsible for definition and execution of this test

4.11.9.2 Test Specification

Test Items
JobMA application in in this case the item tested. As the rest of the reference applications, it's not included at this moment as part of any XtreemOS release, but, it's possible to be downloaded from the project repository.

Features to be Tested
This test case tests the "view info" feature offered by the JobMA application

Approach Refinements
The goal of this test is to demonstrate that it's possible to view additional info about a concrete job using the GUI offered by the JobMA application. A Nokia N800 with XtreemOS-MD and JobMA applications pre-installed will be used for this test.

We will check that the JobMA GUI works and that the job additional info is presented when requested.

4.11.9.3 Test Results

The test was run following this procedure:

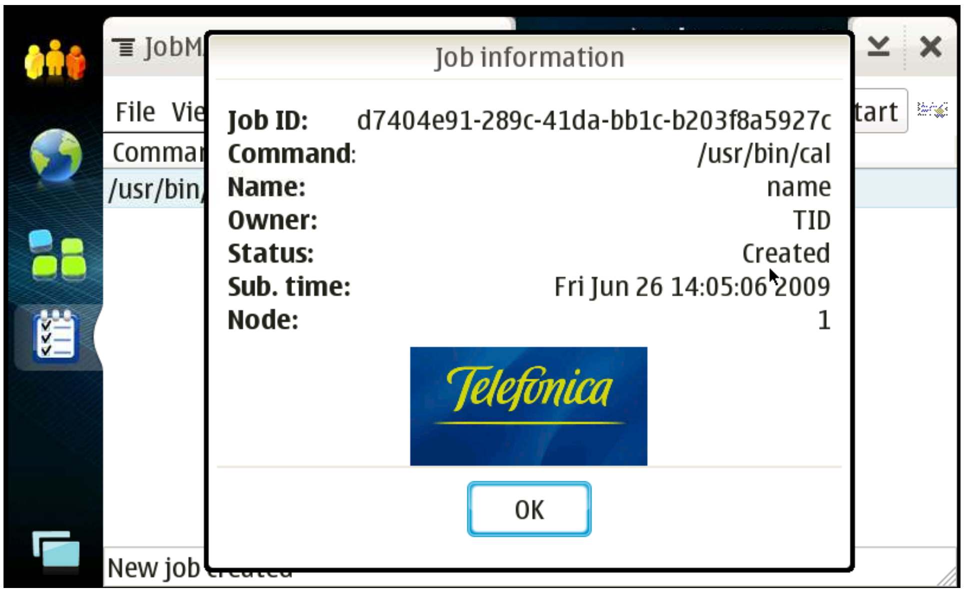
Installation Once installed XtreemOS-MD on the mobile device used, JobMA application should also be installed in the terminal following the installation and configuration instructions.

Set Up No special configuration after the installation is needed.

Start Once installed XtreemOS-MD and JobMA application in our mobile device, after launching this JobMA application, user has to create a job to view info about it. This can be done from two different ways, loading a JSDL file or defining a job.

After the job is created, user can consult all the information about it doing double click on the created job or clicking on the "Action" tab in the top menu and then in the "View info" option in it. After doing this, a new window will appear in which some info about the job is shown, like "Job ID", "Command", "Name", "Owner", "Status", "Sub.time" or "Node".

Procedure Results The execution of this test was successful. The additional info related to a job correctly presented using the JobMA GUI.



```
Job ID:      d7404e91-289c-41da-bb1c-b203f8a5927c
Command:                              /usr/bin/cal
Name:                                         name
Owner:                                         TID
Status:                                    Created
Sub. time:                Fri Jun 26 14:05:06 2009
Node:                                            1
```

4.11.10 Test Unit 09: Using JobMA for running a job

4.11.10.1 Responsibilities

WP3.6 and WP4.2, and TID as partner, are responsible for definition and execution of this test

4.11.10.2 Test Specification

Test Items
JobMA application in in this case the item tested. As the rest of the reference applications, it's not included at this moment as part of any XtreemOS release, but, it's possible to be downloaded from the project repository.

Features to be Tested
This test case tests the execution of jobs using the JobMA application

Approach Refinements
The goal of this test is to demonstrate that it's possible to run jobs using the GUI offered by the JobMA application. A Nokia N800 with XtreemOS-MD and JobMA applications pre-installed will be used for this test.

An already created job is needed, as we will check that the JobMA GUI works and that the job is correctly executed.

4.11.10.3 Test Results

The test was run following this procedure:

Installation Once installed XtreemOS-MD on the mobile device used, JobMA application should also be installed in the terminal following the installation and configuration instructions.

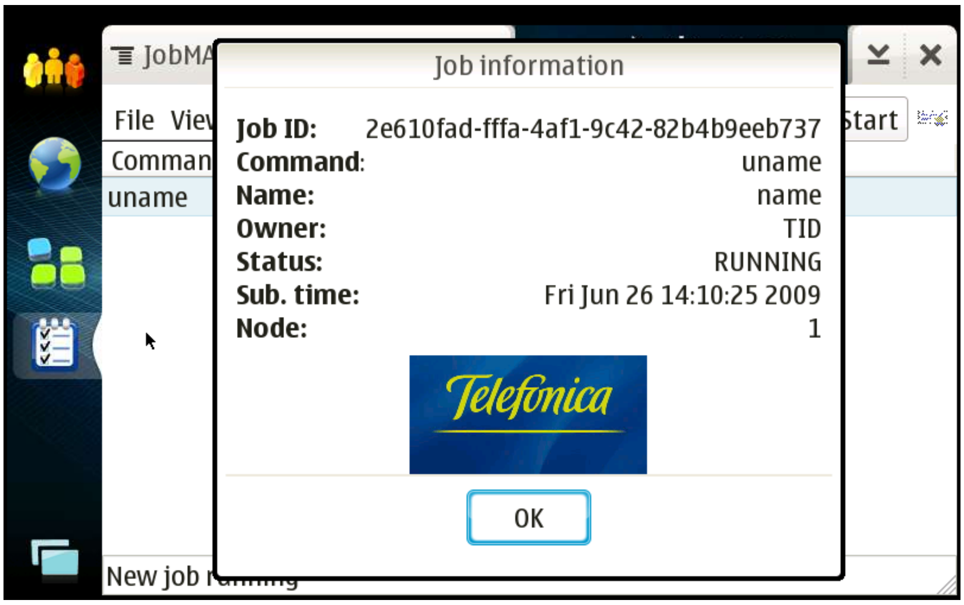
Set Up No special configuration after the installation is needed.

Start Once installed XtreemOS-MD and JobMA application in our mobile device, after launching this JobMA application and after creating a job, user has to run it to get the expected result.

The first step to do this is selecting the job the user wants to run and then clicking on the "Action" tab in the top menu and then in the "Run" option in it. After doing this, column "State" change from the previous state to "Running" and a new message will appear at the bottom of this JobMA window ("New job running") explaining that we have run a created job.

If everything goes OK, a new file will be created in the XtreemFS with the result of the execution.

Procedure Results The execution of this test was successful. The additional info related to a job correctly presented using the JobMA GUI.



Job information

Job ID:	2e610fad-fffa-4af1-9c42-82b4b9eeb737
Command:	uname
Name:	name
Owner:	TID
Status:	RUNNING
Sub. time:	Fri Jun 26 14:10:25 2009
Node:	1

New job running

4.11.11 Test Unit 10: Using JobMA to suspend running a job

4.11.11.1 Responsibilities

WP3.6 and WP4.2, and TID as partner, are responsible for definition and execution of this test

4.11.11.2 Test Specification

Test Items
JobMA application in in this case the item tested. As the rest of the reference applications, it's not included at this moment as part of any XtreemOS release, but, it's possible to be downloaded from the project repository.

Features to be Tested
This test case tests the suspension of running jobs using the JobMA application

Approach Refinements
The goal of this test is to demonstrate that it's possible to suspend a running job using the GUI offered by the JobMA application. A Nokia N800 with XtreemOS-MD and JobMA applications pre-installed will be used for this test.

An already running job is needed, as we will check that the JobMA GUI works and that the job is correctly suspended.

4.11.11.3 Test Results

The test was run following this procedure:

Installation Once installed XtreemOS-MD on the mobile device used, JobMA application should also be installed in the terminal following the installation and configuration instructions.

Set Up No special configuration after the installation is needed.

Start After launching JobMA application and clicking on the "Start" button in the top right corner of this application, all the users jobs will be displayed. One of the actions that the user can do with these jobs, but only with active jobs, is to suspend them. This action could be done from the "Suspend" option in the "Action" tab of the menu of the JobMA. After clicking on this option, "State" column of the job will change to "SUSPEND" instead of the previous state.

Procedure Results The execution of this test was successful and the running job was correctly suspended.

4.11.12 Test Unit 11: Using JobMA to resume a suspended job

4.11.12.1 Responsibilities

WP3.6 and WP4.2, and TID as partner, are responsible for definition and execution of this test

4.11.12.2 Test Specification

Test Items
JobMA application in in this case the item tested. As the rest of the reference applications, it's not included at this moment as part of any XtreemOS release, but, it's possible to be downloaded from the project repository.

Features to be Tested
This test case tests the feature of resuming a suspended job using the JobMA application

Approach Refinements
The goal of this test is to demonstrate that it's possible to resume a suspended job using the GUI offered by the JobMA application. A Nokia N800 with XtreemOS-MD and JobMA applications pre-installed will be used for this test.

An already suspended job is needed, as we will check that the JobMA GUI works and that the job is correctly resumed.

4.11.12.3 Test Results

The test was run following this procedure:

Installation Once installed XtreemOS-MD on the mobile device used, JobMA application should also be installed in the terminal following the installation and configuration instructions.

Set Up No special configuration after the installation is needed.

Start After launching JobMA application and clicking on the "Start" button in the top right corner of this application, all the users jobs will be displayed. One of the actions that the user can do with these jobs, but only with jobs that have been suspended or stopped, is to resume them. This action could be done thanks to the "Resume" option in the "Action" tab in the menu of the JobMA application window. After clicking on this option, "State" column of the job will change to "RUNNING" instead of the previous state in which the job was.

Procedure Results The execution of this test was successful and the suspended job was correctly resumed.

4.11.13 Test Unit 12: Using JobMA to cancel a job

4.11.13.1 Responsibilities

WP3.6 and WP4.2, and TID as partner, are responsible for definition and execution of this test

4.11.13.2 Test Specification

Test Items
JobMA application in in this case the item tested. As the rest of the reference applications, it's not included at this moment as part of any XtreemOS release, but, it's possible to be downloaded from the project repository.

Features to be Tested
This test case tests the cancellation of a job using the JobMA application

Approach Refinements
The goal of this test is to demonstrate that it's possible to cancel an existing job (independently of its current status) using the GUI offered by the JobMA application. A Nokia N800 with XtreemOS-MD and JobMA applications pre-installed will be used for this test.

An already existing job is needed, as we will check that the JobMA GUI works and that the job is correctly canceled.

4.11.13.3 Test Results

The test was run following this procedure:

Installation Once installed XtreemOS-MD on the mobile device used, JobMA application should also be installed in the terminal following the installation and configuration instructions.

Set Up No special configuration after the installation is needed.

Start After launching JobMA application and clicking on the "Start" button in the top right corner of this application, all the users jobs will be displayed. One of the actions that the user can do with these jobs (active, stopped, suspended and resumed), is cancel them. This action could be done thanks to the "Cancel" option in the "Action" tab in the menu of the JobMA application window. After clicking on this option, the job will appeared as "Canceled" in the list of jobs shown.

Procedure Results The execution of this test was successful and the job was correctly canceled.

4.11.14 Test Unit 13: Communications using IMA application

4.11.14.1 Responsibilities

WP4.2, and TID as partner, are responsible for definition and execution of this test

4.11.14.2 Test Specification

Test Items
IMA application, based on the well-known Pidgin messaging application, is in this case the item tested. As the rest of the reference applications, it's not included at this moment as part of any XtreemOS release, but, it's possible to be downloaded from the project repository.

Features to be Tested
This test case tests the communications using the IMA application

Approach Refinements
The goal of this test is to demonstrate that is possible to reuse common applications, not coming directly from XtreemOS project, let's say, "native" applications, even if enhanced to make use of the Grid capabilities. A Nokia N800 with XtreemOS-MD and IMA applications pre-installed will be used for this test.

A jabber account is needed, and we will check that the GUI and behavior of the IMA application is the usual one out of the XtreemOS world.

4.11.14.3 Test Results

The test was run following this procedure:

Installation Once installed XtreemOS-MD on the mobile device used, IMA application should also be installed in the terminal following the installation and configuration instructions.

Set Up Log in in the Grid with a user/password valid.

Start Launch the IMA application, connect using a jabber account, and check that it's possible to receive messages, create conversations, etc. let's say, check that the messaging application behave in the same way that it will do out of XtreemOS-MD.

Procedure Results The whole test was executed successfully, connecting with a jabber account and being able to start a conversation with a contact, to exchange messages, etc.

4.11.15 Test Unit 14: IMA and XtreemFS integration

4.11.15.1 Responsibilities

WP3.6 and WP4.2, and TID as partner, are responsible for definition and execution of this test

4.11.15.2 Test Specification

IMA application, based on the well-known Pidgin messaging application, is in this case the item tested. As the rest of the reference applications, it's not included at this moment as part of any XtreemOS release, but, it's possible to be downloaded from the project repository.

Features to be Tested
This test case tests the integration of IMA application and XtreemFS

Approach Refinements
The goal of this test is to demonstrate that it's possible to integrate existing applications with the Grid services. In this case the integration of the IMA application with the XtreemFS service, so that the configuration of the application and the conversation logs can be stored in the Grid, in the XtreemFS. 2 Nokia N800 with XtreemOS-MD and IMA applications pre-installed will be used for this test.

A jabber account is needed. We will check that the configuration of the application (concretely the jabber account configured in the first terminal) is available when launching the application from the second terminal and the logs of the conversations kept using the first terminal are available from the second terminal.

4.11.15.3 Test Results

The test was run following this procedure:

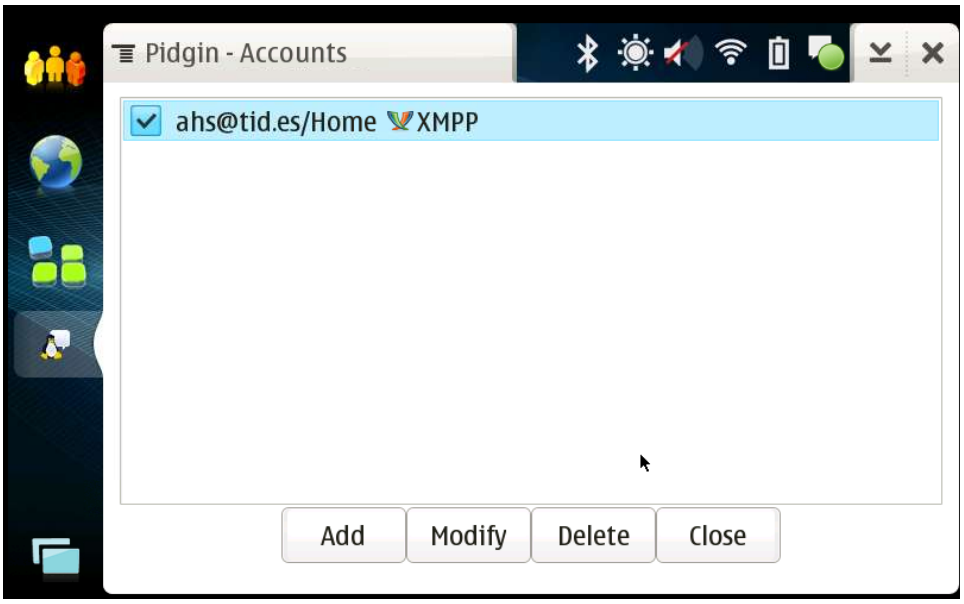
Installation Once installed XtreemOS-MD on the mobile device used, IMA application should also be installed in the terminal following the installation and configuration instructions.

Set Up Log in in the Grid with a user/password valid.

Start The user will start the IMA apllication and will configure a new jabber account. Then, he will connect using the new account and will launch a conversation with one of the contacts of his contacts. Finally he will log out.

Then he will use a different Nokia N8x0 device, also provided with XtreemOS-MD and IMA application. After logging in the Grid with the same username used previously, he will launch the IMA application. The jabber account previously configured in the first terminal should be already available. The user will connect using that account, and he will check the logs of the conversations. The previous conversation from the first terminal should appear in the logs.

Procedure Results The test was executed succesfully. After moving to a different device (the second Nokia N800), we had available the account created from the first terminal and once connected we could see the logs of the conversations kept with this first terminal.



4.11.16 Test Summary Report

4.11.16.1 Summary of Tests and Results

The tests executed have been passed without major problems. The functionalities promised by XtreemOS-MD, IMA and JobMA applications have been verified and from the performance tests we can conclude that the performance of XtreemOS-MD as a Grid client executed from a mobile device like a Nokia N800 is similar to the one achieved by the XtreemOS PC client. On the other hand, execution of jobs in the Grid and even access to the Grid file system are faster than the local execution or local file system access to the terminal, making clear the benefits of using the Grid from a mobile device (specially when thinklng on scenarios like the video conversion one).

4.11.16.2 Conclusion and Directions for Future Work

XtreemOS-MD advanced version, including support for smartphones and some additional features where the context awareness and resource sharing are the most important ones, is currently under development. This new release will extend XtreemOS-MD not only in the number of devices supported, but also on the scope of the software, as the mobile device will become a *light* resource of the Grid and not just a mere client. New tests will be designed and executed to evaluate the new functionalities, specially concerning the “offline-mode” operation and the resource sharing from the mobile device.

Chapter 5

Comparison of XtreemOS with other Grid Solutions

This chapter presents the comparison of XtreemOS with alternate Grid solutions. The subsequent sections start with an overview of the previous and current assessments followed by a theorectical comparison of XtreemOS with a variety of other Grid middleware solutions and Grid operating systems. Finally, a summary of the comparative experiments is given.

5.1 Overview

The comparison between XtreemOS and other Grid approaches is conducted in two ways:

1. Theoretical comparison (see Section 5.2)

2. Experimental comparison (see Section 5.3)

The theoretical comparison assesses the various appproaches first from a general perspective, then XtreemOS is contrasted to various Grid middlware solutions and related Grid operating systems.

The experimental comparison summarizes the results of the comparative experiments with 1) XtreemOS-AEM and Globus-GRAM and 2) XtreemOS-DTokens and Globus-GSI. Finally, the setup of currently executed application-centric large-scale tests (on Grid5000) with Galeb (from XLAB) executed on XtreemOS and Globus GT4.0 is described.

5.2 Theoretical Comparison

Table 5.1 compares the usage characteristics and the properties of classical Grid computing tools with the extended scope of XtreemOS. It is differentiated between application landscape, reach, job characteristics, data characteristics, performance requirements, Service Level Agreements, legacy and main drivers.

Table 5.1: Comparison between classical Grid Computing tools and the extended scope of XtreemOS

	Classical Grid Infrastructures	Extended Scope of XtreemOS
application landscape	• automated batch processing • file-based	• multi-tier solutions • database
reach	• targeting the world-wide thing	• many administrative domains within a single administrative super-domain (hosting) • dynamic VOs across administrative domains
job characteristics	• stateless batch jobs • short deployment times (< secs) • limited execution times • highly mobile	• stateful sessions, services and jobs • long deployment times (mins, hrs) • possibly unlimited execution time • restricted mobility(licensing, sessions)
data characteristics	• relatively flat data structures • no transactional data	• large amounts of highly structured data • transactional data
performance requirements	• high throughput • high resource utilization	• good response time for all interactive requests • high resource utilization

Service Level Agreements	• performance isolation of individual applications	• performance isolation across applications, middleware and DB
legacy	• do not care about legacy	• legacy support is a must
main drivers	• get the compute power • get it done at all/quicker	• improve administrative flexibility • get it done efficiently (cost vs. customer-specific quality)

The computation problems, which are typically tackled with classical Grid tools require large amounts of either computation time or data, and they can be reduced into several small parallel processes with only little inter-process communication and a limited execution time (as opposed e.g. to interactive applications). The majority of the computations can be characterized as stateless batch jobs mainly performing file-based input and output operations. Such jobs can be deployed in a relatively short time as they are submitted in a self-contained manner along with all input data files and executables required. Usually, the jobs do not depend on locally available license files or user interactions and are therefore highly mobile.

The applications characterized mainly stem from the scientific computing domain. XtreemOS tries to also address far more advanced requirements of applications from the business domain, where solutions often possess a multi-tier architecture and interact with a (normally central) database. Applications are executed within a company consisting of many administrative domains belonging to a single administrative super-domain. Here, the dynamic VO support offered by XtreemOS targets business requirements and also allows to control VOs across administrative domains. As mentioned above, to address business needs the Grid infrastructure must be able to deal with databases and all related requirements. Thus, it is not possible to hide the transport of data to the executing node by prefetching all data before the job execution starts. In business scenarios, small sets of data must be loaded from and stored to a database with random access patterns. This does not only affect the execution time of the application itself, but also the scheduling decisions as the node that executes a certain application cannot be allocated too far from the database. Depending on its size the database remains static at a certain location. Furthermore, many business applications are interactive, i.e. interaction takes places within open and stateful sessions, small data packets are exchanged frequently with the database, and no delays in data transmissions can be tolerated. Application components can be migrated but the interactive session must remain open.

Data management in classical Grid environments focuses on data modeling, data movement, and handling of distributed and replicated files. Typically, data is modeled in comparatively flat data structures, and data access is performed in a non-transactional fashion. Resource management puts the main emphasis on high throughput and high resource utilization whereas low response times are often considered as less important. Security issues are commonly of minor interest since confidentiality and integrity of data are not critical or computation resources and networks are assigned exclusively to scientific project members.

XtreemOS, however, aims at the far more challenging security constraints as imposed by business applications. Thus, the execution of the application itself must be secured and also all the related data before, during, and after the processing. Such security constraints also limit the allocation possibilities and the mobility of data. XtreemOS allows to manage secured isolation on shared resources where the degree and kind of isolation can be customized by the user ranging from name space separation, container-based isolation up to physical isolation.

The following sections re-capture, update and extend the initial comparison discussed in the XtreemOS Vision Paper [13]

5.2.1 Comparison with Grid Middleware

Figure 5.1 gives an overview of a classical Grid architecture consisting of multiple layers and middleware levels [2].

The fabric level is composed of networked and distributed resources ranging from computers, networks, storage systems, data sources to scientific instruments. The computational resources can be very diverse like simple PCs, clusters or supercomputers. Local resource managers control the access to and provide information about operating system, queuing system libraries and application or protocols used.

The core Grid middleware level includes services for remote process management, co-allocation of resources, storage access, registration and discovery of information, security and QoS-aspects like resource reservation and trading. Such services hide he complexity and heterogeneity of the fabric level and provide a consistent interface for the access to distributed resources.

On the user level, the middleware utilizes these interfaces in order to deliver higher level abstractions and services including e.g. application development and programming tools, resource brokers for resources management and scheduling of tasks to be executed on global resources.

Finally, applications and portals make use of programming languages and utilities. Grid portals offer Web-enabled application services which allows the user to submit jobs and receive results from applications running on the web.

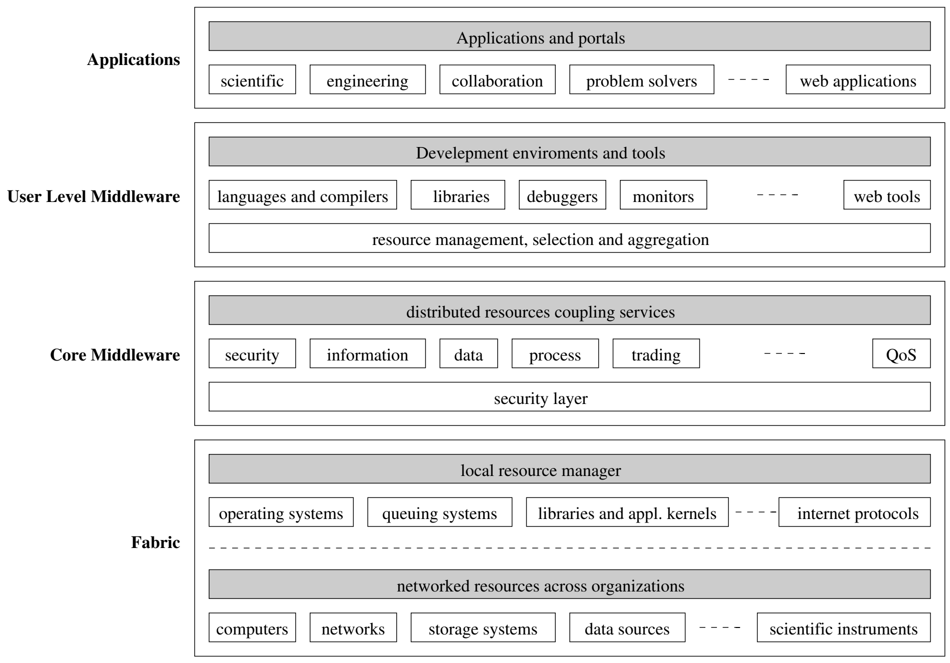


Figure 5.1: Layers and components of a typical Grid environment (based on [2])

XtreemOS primarily deflates the core middleware level into the operating system which in turn is extended by kernel patches or kernel modules. These extensions represent services on the foundation layer (XtreemOS-F) providing for a unified access to physical nodes through the node-level VO concept, and on the grid layer (XtreemOS-G) offering higher level features for highly available and scalable services, data management, application execution management and security. Unified access to these services can be provided by the XOSAGA interface. Application users and developers can use XtreemOS via six different ways: executing legacy application by issuing the familiar Linux commands, submission into a VO with the ssh-xos command, submission through XOSAGA, submission through AEM with xsub or submission though AEM using the SAGA AEM bindings. The variety of interfaces makes the usage of XtreemOS very versatile for different needs of the various applications. A Grid architecture as shown in 5.1 consists of many layers and tools which are often not well-integrated and a lot of information is often lost on the way through the sub-subsystems or correlation is lost which renders usage of such information very difficult. For instance, in such Grid systems it may be difficult to determine why an application failed and on which resources this application was executed actually. XtreemOS aims at integrating all services in a single OS which shall facilitate usage and give the user an execution environment with rich monitoring information and a powerful execution control. The classical approach also implies many scheduling levels which may not be sufficiently coordinated such that decision made on one level may contradict the decisions on another level. XtreemOS aims at reducing the amount of scheduling levels which should result in overall more efficient execution schedules. Finally, the classical multi-layered approach complicates accurate accounting, whereas XtreemOS will provide for detailed accounting information of the resources used which is a more useful basis for billing or compensation.

Globus [16] offers an open source software toolkit used to construct computational Grids and Grid-based applications. Amongst others Globus provides services for job submission, file staging, replica location and management, publishing and querying of resource information etc. Installation, management and usage of Globus is comparatively difficult due to the vast number of services and due to the lack of consistent interfaces and the lack of a unifying model of the interaction between services. Such issues must be treated by the programmer who spends valuable time on basic Grid functions thereby increasing development cost needlessly. XtreemOS has been built with a completely different goal in mind to develop a Grid-enabled OS. XtreemOS shall remove the burden from the application programmer who can rely on the native services of the operating system for tasks like resource or process management. After the installation of XtreemOS on a machine, the machine is ready to join a VO without the need to install additional software. Nevertheless, modifications to the underlying OS are done carefully in order to maintain backward compatibility with standard Linux. The services provided by XtreemOS are roughly equivalent to Globus, however, they are offered in a more consistent and integrated manner. For example, data management in XtreemOS is based on the Grid file system XtreemFS which makes data available through a POSIX API in a transparent manner and avoids the need to move files explicitly. Also file replication is supported to improve performance and fault tolerance. One further major advantage of XtreemOS is the support of interactive applications whereas Globus executes application in batch mode. Further large production Grid system are built on Globus including, e.g. gLite [7] and the EU-project EGEE [6].

Zorilla [15] is a java-based Grid middleware which combines locality-aware P2P resource coallocation with a scheduling system. A further P2P middleware solution is Vishwa [37], which provides two layers, a structured and an unstructured layer. The structured layer is responsible for reconfiguring applications for masking failures. The unstructured layer reconfigures applications to adapt them to varying load. Common to these P2P middleware solution is that they do not offer the complete software stack which would support all needed functionalities.

5.2.2 Comparison with other Grid Operating Systems

The first comparison shall be done with Legion [19] which is an object-based wide-area operating system. Legion aims to be a virtual operating system (OS) for distributed resources and delivers a uniform API and object space for users and developers. Even though Legion exposes itself like an OS it still is a middleware running on top of the OS hosting it. Furthermore, Legion can execute of various different operating systems running on heterogeneous resources. Also XtreemOS provides a common API (POSIX API), however, all resources must run XtreemOS with one of the three different flavors: for PCs, clusters or mobile devices. Interoperability and homogeneous programmability is offered by the XOSAGA interface. XtreemOS and Legion also have several requirements in common with respect to security services, global name spaces, ease of programming, interactivity, fault tolerance, persistence, dynamicity, scalability and autonomy of sites. Different approaches are followed regarding user and resource management. XtreemOS implements a native support for Virtual Organizations (VOs) where users can be members of multiple VOs, resources can be shared among multiple VOs with customizable mechanisms for secure isolation. Legion supports the concept of autonomous domains managing their local users and resources. Larger systems can be created by combining such domains where objects from one domain can call services of objects from other domains.

Mosix2 [3] has many similarities with LinuxSSI, the cluster flavor of XtreemOS. As management system Mosix2 targets high performance computing on Linux clusters and multi-cluster organizational Grids. Both, Mosix2 and LinuxSSI virtualize cluster nodes giving the impression of working with a single OS on a single powerful machine. It is not required to modify applications, to link them to special libraries, to copy files to remote nodes or to login to such nodes. The two operating system also share similar approaches to dynamic resource management and automatic workload distribution. Opposed to Mosix, LinuxSSI is enabled with VO support such that a cluster can join an XtreemOS Grid thereby appearing like a large SMP node.

Also GridOS [34] is based on a Linux operating system extending it with features of classical Grid middleware. It offers modules for communication, resource management, process management, high performance I/O and a kernel ftp server and client. These are basic functionalities integrated in the kernel which shall facilitate middleware support and also focuses on performance gains in data management.

WebOS [42] provides services for resource discovery, global name spacing, remote process execution, resource management, authentication and security. The main difference to XtreemOS is that WebOS primarily aims at applications in wide-area scenarios whereas XtreemOS also targets local or mixed scenarios and with different flavors.

Globe [43] follows an approach similar to Legion as both are grid middlewares providing classes and objects to abstract from the host operating system and the physical resources. Whereas Legion aims to be a virtual operating system for Grids, Globe offers a distributed application environment.

9Grid [30] aims at extending the distributed operating system Plan 9 [36] into the area of Grid Computing. Plan9 provides support for user authentication, resource discovery and data management in distributed environments. Whereas in Plan9, administration is limited to a single domain, 9Grid also supports multi-domain name spaces and remote authentication agents which resembles XtreemOS managing multiple VOs on shared resources.

Running on Linux, Vigne [21] offers services for resource discovery, distributed application management and automatic application life cycle management. Vigne also aims at high scalability and transparency, however, it does not provide support for VOs, security and no Grid file system.

5.3 Experimental Comparison

The following two sections describe the current status of the experimental comparisons between XtreemOS and Globus. A first set of experiments has been carried out with the job submission and delegation features of XtreemOS and the Globus Toolkit. Furthermore, the setup of a running application-centric comparison using the Galeb application is described. Forthcoming evaluations will execute extended scalability comparison tests on large scale systems.

5.3.1 Comparison of Job Submission and Delegation on Globus and XtreemOS

Here the test results are briefly summarized. Detailed descriptions of the test specifications and interpretation of test results are given in sections 4.8.6 and 4.10.2. In Section 4.8.6, job submission with XtreemOS through AEM was compared with job submission with Globus using GRAM4. Here job submission through AEM was more than six times faster than through GRAM4. One further test (see Section 4.10.2) examined the time needed for the creation and the verification of a token (in the case of DTokens from XtreemOS) or of a proxy certificate (in the case of GSI Proxy Certificates from Globus). The results demonstrate that DTokens provide significant performance gains over the proxy certificate solution. They also show that verification cost involved in both solutions are significantly less than creation cost. From a performance perspective, the results suggest that it is much more affordable to employ highly secure cryptography keys in the DToken architecture than in GSI.

5.3.2 Comparison of Galeb on Globus 4.0 and XtreemOS

Galeb is one of the applications that have been previously adapted to run on Globus Toolkit 4.0 and have been or are being ported to XtreemOS in WP4.2. As such it is an appropriate choice for comparison of these two Grid paradigms, particularly because performance tests on Globus have also been performed in the past [28]. In this deliverable we will compare Globus and XtreemOS from the perspective of the developers and users of Galeb.

5.3.2.1 Application Summary

Galeb is a tool to fit analytical functions to an arbitrary set of data, primarily developed for financial analysis. It constructs functions from the basic unary (log, exp, sqrt) and binary operators (+, -, *, /). It uses the genetic algorithm from the GaLib library[1] to minimize the mean squared error of the fitted function.

From the developer's aspect, Galeb's computational core is a C++ library with one public function whose parameters include the name of the input data file and multiple genetic algorithm parameters (number of generations, mutation probability etc).

Being based on a genetic algorithm, Galeb can be trivially parallelized in a master-with-multiple-slaves fashion by simply splitting it into one or more independent runs on each processor and finally selecting the best solution obtained. While this approach is in general not an optimal parallelization of the genetic algorithm, our tests have shown that it performs well in case of Galeb. It also requires no communication apart from the initial input distribution and final collection of results.

The command-line version of Galeb simply calls the library function with the supplied parameters. This version has also been parallelized for SMP machines using multiple processes communicating through System V IPC message queues. The latter version has in the past been successfully run on LinuxSSI flavour of XtreemOS, although with some problems related to checkpoint/restart and migration [10].

5.3.2.2 Porting to Globus

Galeb was ported to Globus Toolkit 4.0 following [38]. The computational core was exposed as a grid service (GS), which has to be deployed on all nodes contributing to the calculation. An alternative approach would be to install the commandline version of Galeb to all nodes and submit jobs running it, as described in [17]. Both cited tutorials suggest implementing job scheduling within the application.

Galeb Grid Service Because the individual runs of the genetic algorithm are independent the GS can be stateless, thus the so called *single resource* implementation suffices, in contrast to a stateful *multiple resource* implementation that would have each client communicate with its own instance of the GS and thus its own version of internal service state. Our grid service consists of the multiple components, each of them implemented by a Java class. Most of these components are related to GS state management and thus trivial in our case. The only non-trivial component is the GS implementation itself, which calls the Galeb library through JNI.

[1] `http://lancet.mit.edu/ga/`, the GAlib genetic algorithm package, written by Matthew Wall at the Massachusetts Institute of Technology.

Deployment of the Grid Service in Globus Even such a simple GS requires some effort to build and deploy. Source files of all the components have to be put in appropriately named directories, as well as the web service definition file (wsdl) and deployment parameter files (xml and wsdd files), after which the scripts `globus-build-service` and `globus-deploy-gar`, both supplied with Globus, are used to build and deploy, respectively. The process creates tens of other files and directories. Although these files are automatically generated and do not need any editing they still clutter the project.

Parallelization The described GS, even when deployed to many nodes, is not distributed at all and cannot be called by the end user in a simple fashion. A proxy application was thus written that takes care of both.

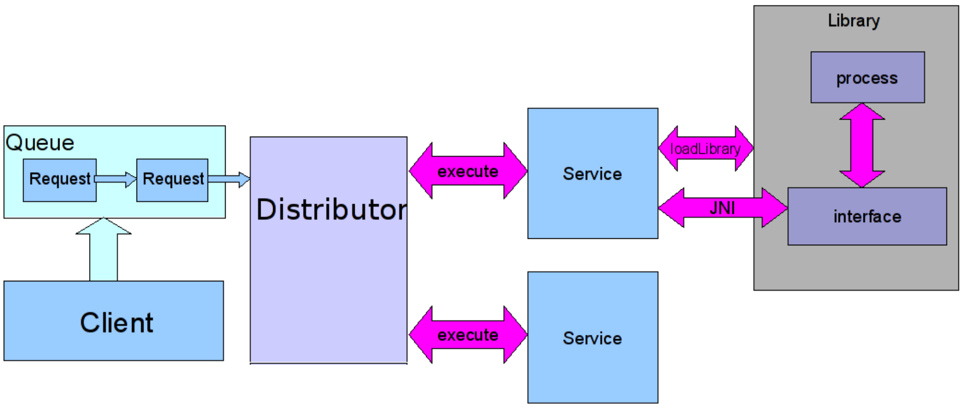


Figure 5.2: Architecture of Grid-Service-based implementation of Galeb on Globus

The application acts as a distributor. It first copies the input file using GridFTP to all the nodes where the GS is deployed. Then it distributes jobs, each of which represents a single run of the genetic algorithm, to the nodes. The results are, again, transferred to the master node using GridFTP, and the best of them is finally returned. The architecture of the whole application is shown in Figure 5.2.

5.3.2.3 Porting to XtreemOS

Similarly to Globus, XtreemOS offers multiple ways to *gridify* a master-slave type parallel calculation such as Galeb:

- running the slaves on selected resource nodes using `ssh-xos`,
- submitting the calculation jobs on selected nodes through XOSAGA, the XtreemOS implementation of SAGA, the official XtreemOS API,

- submitting the jobs through AEM using the `xsub` utility, leaving resource discovery and scheduling to XtreemOS rather than re-implementing it within the application,
- submitting the jobs through AEM using the SAGA AEM bindings.

The first approach has been implemented in Galeb some time ago and is similar to the Globus approach recommended in [17]. The second not only uses a different API but also allows submission to any node accessible through SAGA, not just XtreemOS nodes[2]. This approach has also been implemented in Galeb. These two approaches do not differ significantly in terms of programming and deployment effort. Both duplicate XtreemOS functionality, i.e. resource selection and scheduling, in the application.

The third and fourth approaches are thus recommended. Likewise, they differ significantly in the API used but not in effort required. The interoperability advantage of SAGA does not apply here because XOSAGA is the only SAGA implementation that includes the AEM functionalities. The SAGA-AEM implementation of Galeb is currently under development.

5.3.2.4 Galeb for XtreemOS vs. Galeb for Globus

In terms of programming effort the advantage of XtreemOS is that resource selection and scheduling does not have to be implemented in the application. Apart from this the effort is similar, except if grid services are used in Globus, which increases the effort significantly but offers other advantages. Note that such service-oriented approach could also be used in XtreemOS and would even offer fault-tolerance [11]; however, we did not explore this further because fault-tolerance of the slaves is not beneficial in case of Galeb.

The benefit of XtreemFS, compared to GridFTP, is the lack of need to explicitly copy files to any certain node. Moreover, in case of simple applications and homogeneous hardware architecture deployment to any resource nodes is not required. Instead the executable is simply placed on VO user's XtreemFS home volume, where it can be read by any resource selected by AEM.

In a Globus grid the grid map file on each resource contains an explicit mapping to a local account for each certificate subject, i.e. for each grid user. This presents a significant administration burden for resource administrators. Mapping multiple grid users to the same local account prevents isolation between users. XtreemOS avoids these problems elegantly with the dynamic account mapping.

To be fair, one has to mention that Globus is currently a much more mature and better documented platform than XtreemOS. For users outside the XtreemOS consortium the apparent risk of the platform being simply discontinued in the future is also greater for XtreemOS than for Globus.

[2]Note that SAGA for Globus is not yet available.

Chapter 6

Usability evaluation of XtreemOS for Mobile Devices

WP4.2 evaluated the usability of XtreemOS for Mobile Devices. A first approach consists of the usability evaluation of the XtreemOS-MD installer and the JobMA application, which are probably the most graphical elements, and then the easiest in order to apply the usual criteria for usability evaluation.

6.1 Introduction and goals

The main objective of the 1st XtreemOS usability report is to establish an initial state of JobMA application and XtreemOS-MD installer in some mobile devices, such as Nokia N800 and OpenmMoko Neo Freerunner, and to determine the level of difficulty associated to the installation and management of these applications in the tested devices.

Moreover, some problems about the installation and management will be explained and some possible solutions to them will be provided, besides other possible improvements or future work.

6.2 Methodology

In this 1st iteration the methodology followed consist in the analysis of the ease of use of JobMA application (focusing on the main window and the operations provided) and the ease of installation of XtreemOS-MD using the installer provided. In particular, these applications will be analyzed over the Nokia N800 and OpenMoko Neo Freerunner devices.

The methodology followed to is based on a "Heuristic Evaluation", which is a method to evaluate the possible usability problems of a user interface.

6.3 Results

6.3.1 JobMA

6.3.1.1 JobMA application Strengths

- Easy installation
- Graphical appearance is good
- Easy management of jobs
- Intuitive operation
- Contextual help and information

6.3.1.2 JobMA application Areas of improvement

- Concrete small details of the graphical interface
- Size of the main window depending on the mobile device
- Lack of tutorials or on-line help

6.3.1.3 Summary of the evaluated scenarios

The following figure shows the level of ease under six different scenarios, depending on the action the user wants to do: loading the certificate to authenticate, opening a JSDL with the definition of a job, defining a new job, viewing more info related to a concrete job, running the job or monitoring the status of a job. All these aspects are shown in the next figure with values between 1 and 5 points, depending on the ease of use in each case.

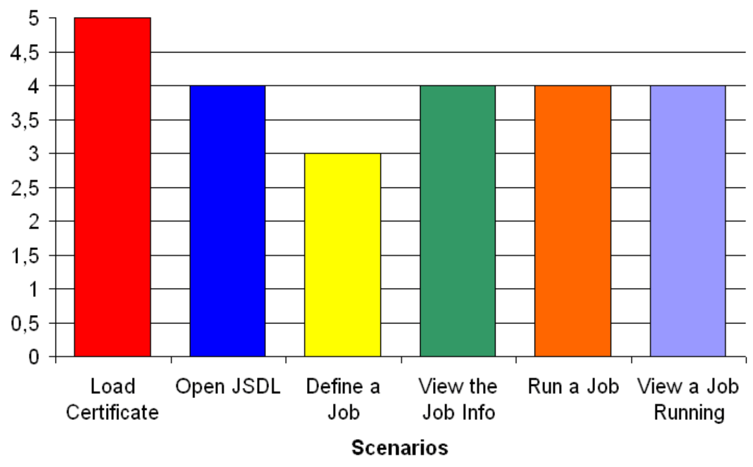


Figure 6.1: JobMA application result, scale 1-5 (1 very difficult and 5 very easy)

6.3.1.4 Initial User perception of JobMA over Maemo

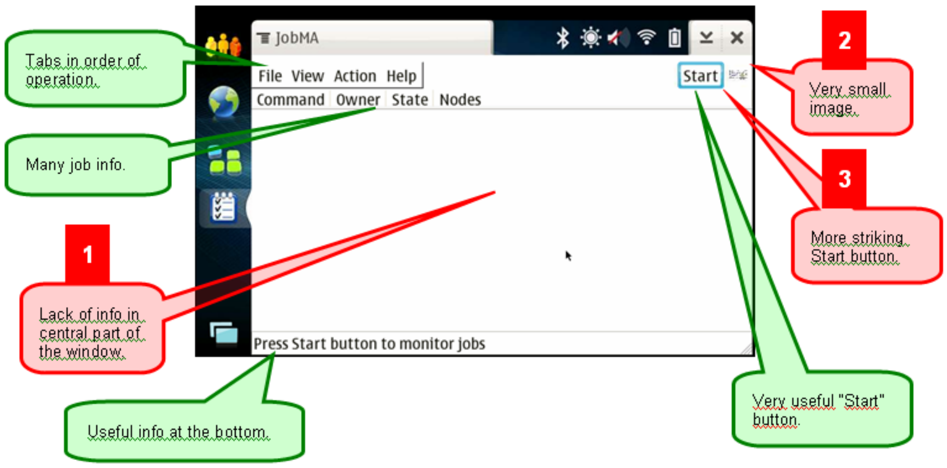

Tabs in order of operation.
Many job info.
1
Lack of info in central part of the window.
Useful info at the bottom.
JobMA
File View Action Help
Command Owner State Nodes
Start
Press Start button to monitor jobs
2
Very small image.
3
More striking Start button.
Very useful "Start" button.

Figure 6.2: JobMA main window on Nokia N800

1	First screen is too simple and a big part in the center of the window is blank	Minor: It's not really good-looking
2	XtreemOS icon too small, the user cannot see it	Minor: No function associated to this icon
3	Start button should be more visible and emphasized	Minor: Aspect improvement recommended

Table 6.1: JobMA main window weaknesses

How this could be improved?

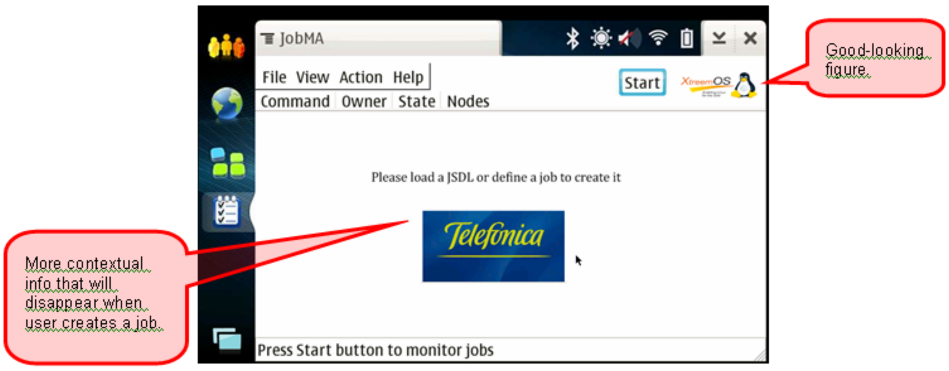

JobMA
File View Action Help
Command Owner State Nodes
Start
XtreemOS
Please load a JSDL or define a job to create it
Telefonica
Press Start button to monitor jobs
Good-looking figure.
More contextual info that will disappear when user creates a job.

Figure 6.3: JobMA main window improved on Nokia N800

6.3.1.5 Opening a JSDL file with JobMA application

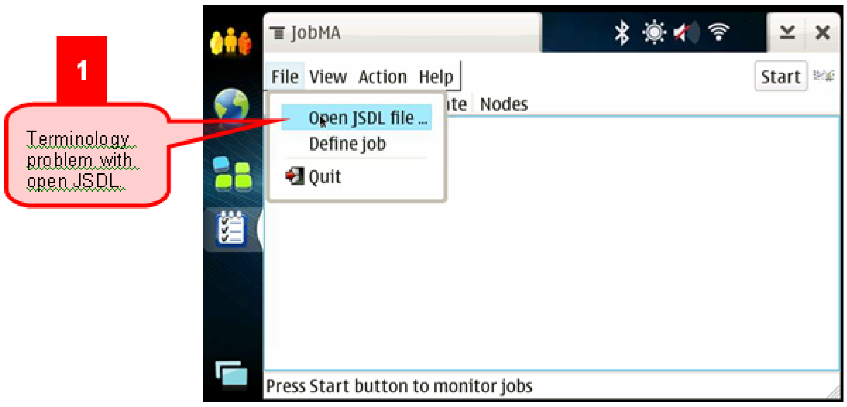


Figure 6.4: JobMA "Open a JSDL" window on Nokia N800

1	User has to load a JSDL file to create a new Job, but end-user doesn't know the purpose of this file	Medium: Users could load a JSDL file, without knowledge of the job that will be created

Table 6.2: JobMA "Open a JSDL" window weaknesses

How this could be improved?

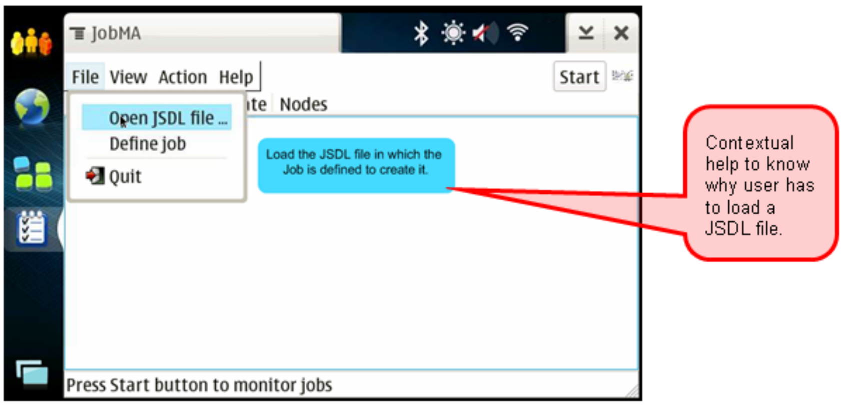

JobMA
File View Action Help
Start
Open JSDL file ...
Define job
Quit
Nodes
Load the JSDL file in which the Job is defined to create it.
Contextual help to know why user has to load a JSDL file.
Press Start button to monitor jobs

Figure 6.5: JobMA "Open a JSDL" improved window on Nokia N800

6.3.1.6 Defining a job with JobMA application

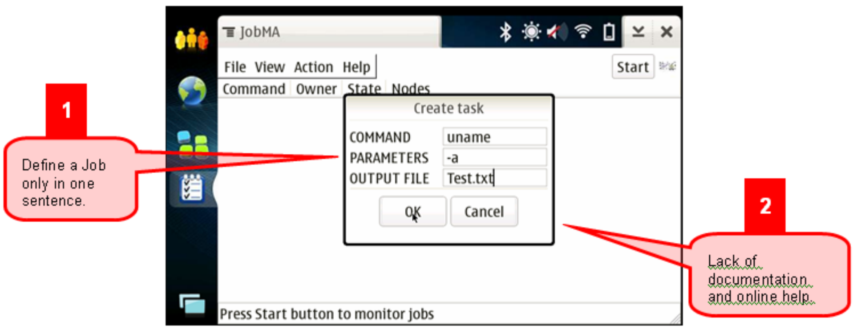

JobMA
File View Action Help
Start
Command Owner State Nodes
Create task
COMMAND uname
PARAMETERS -a
OUTPUT FILE Test.txt
OK
Cancel
Press Start button to monitor jobs
1
Define a Job only in one sentence.
2
Lack of documentation and online help.

Figure 6.6: JobMA "Define a job" window on Nokia N800

1	User has to type three sentences to create a basic Job, but it will be more useful if he could do it only with one sentence	Minor: It will be easier for the user
2	More documentation is needed for some parts of the application, explaining what he is doing in each step	Minor: Lack of documentation

Table 6.3: JobMA "Define a job" window weaknesses

How this could be improved?

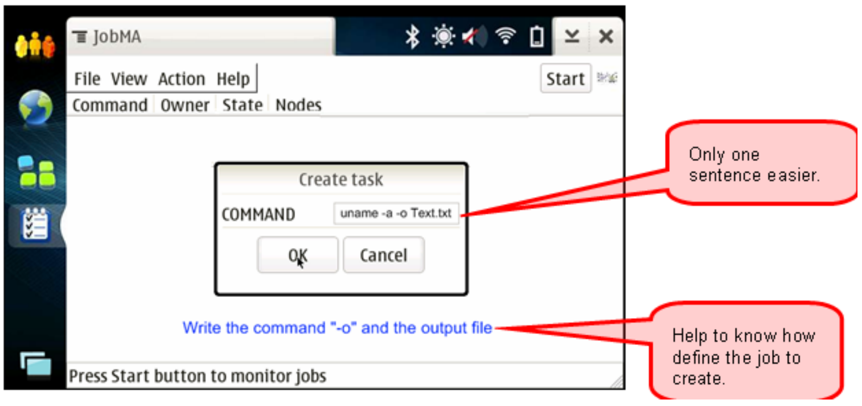

JobMA
File View Action Help
Start
Command Owner State Nodes
Create task
COMMAND uname -a -o Text.txt
OK
Cancel
Write the command "-o" and the output file
Press Start button to monitor jobs
Only one sentence easier.
Help to know how define the job to create.

Figure 6.7: JobMA "Define a job" improved window on Nokia N800

6.3.1.7 Initial User perception of JobMA over OpenMoko

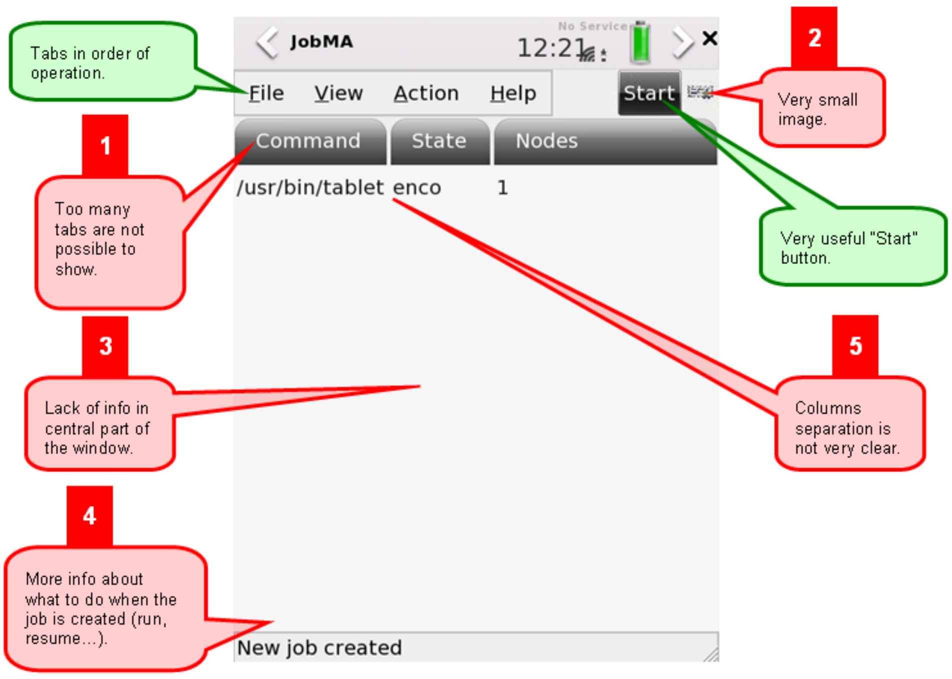
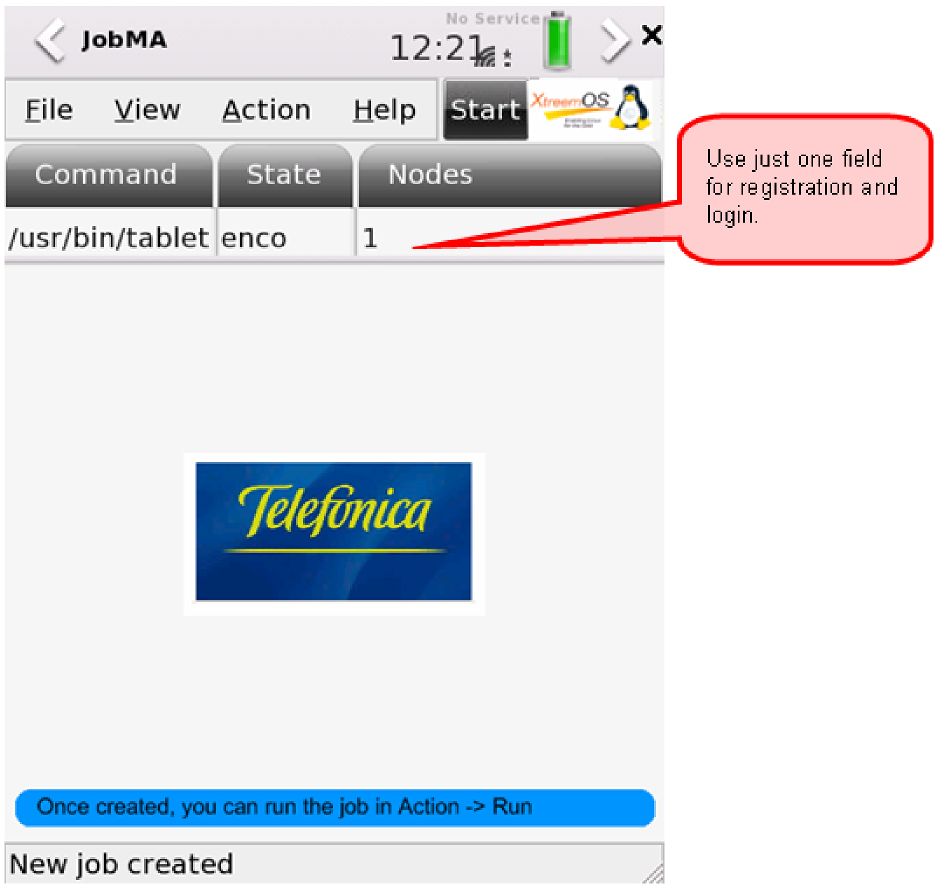

Tabs in order of operation.
1
Too many tabs are not possible to show.
3
Lack of info in central part of the window.
4
More info about what to do when the job is created (run, resume...).
JobMA
No Service
12:21
File View Action Help
Start
Command State Nodes
/usr/bin/tablet enco 1
New job created
2
Very small image.
Very useful "Start" button.
5
Columns separation is not very clear.

Figure 6.8: JobMA main window on OpenMoko NeoFreeRunner

1	"View" tab allows selecting the job info shown. If many fields are selected, not all of them will be displayed	Major: The user will miss some information
2	XtreemOS icon too small, the user cannot see it	Minor: No function associate to this icon
3	Initial screen is too simple and a big part in the center of the window is blank	Minor: It is not really good-looking
4	After creating a job, the user doesn't know what can be done with this created job	Medium: Some contextual help could be useful to solve this problem
5	Field limits are not well differentiated	Minor: One vertical line could solve this problem

Table 6.4: JobMA main window weaknesses over OpenMoko

How this could be improved?



/usr/bin/tablet enco 1

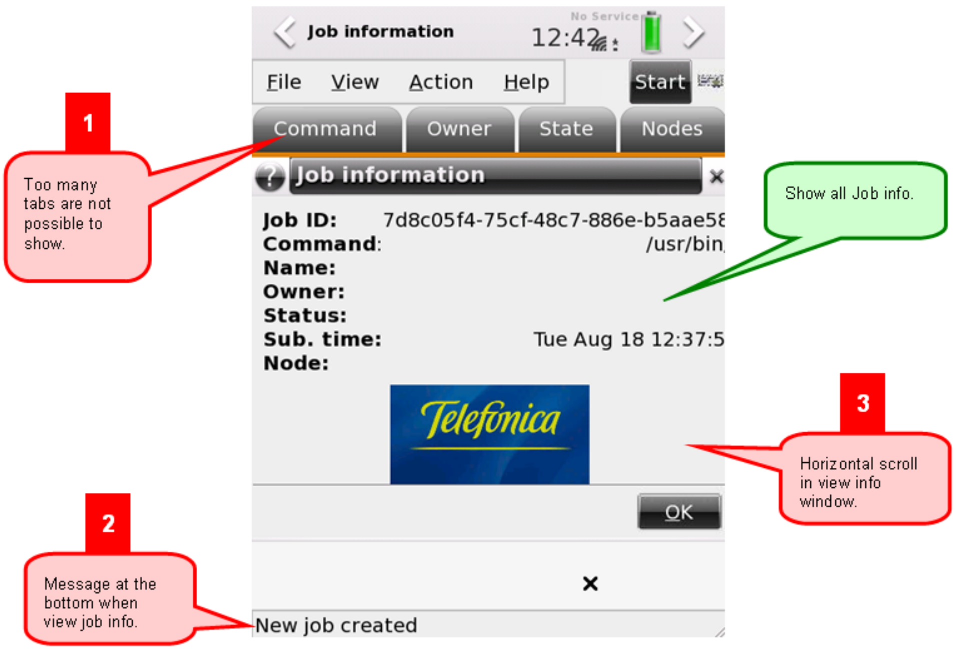
Use just one field for registration and login.

Once created, you can run the job in Action -> Run

New job created

Figure 6.9: JobMA main window improved on OpenMoko NeoFreeRunner

6.3.1.8 Viewing job info with JobMA over OpenMoko


Job information
12:42
No Service
File View Action Help
Start
Command Owner State Nodes
Job information
Job ID: 7d8c05f4-75cf-48c7-886e-b5aae58
Command: /usr/bin
Name:
Owner:
Status:
Sub. time: Tue Aug 18 12:37:5
Node:
OK
New job created

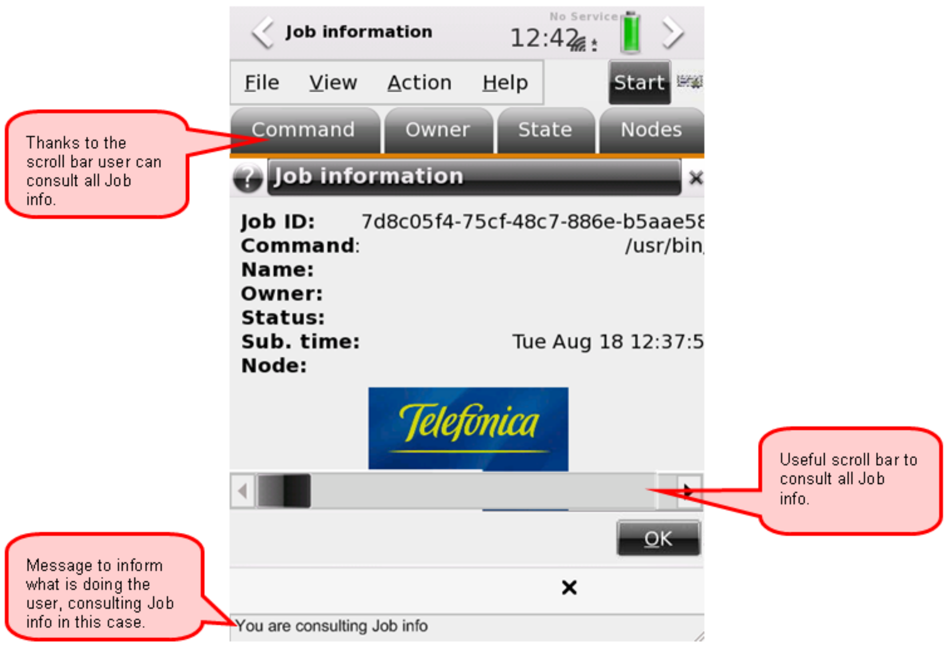
1
Too many tabs are not possible to show.
Show all Job info.
Telefonica
3
Horizontal scroll in view info window.
2
Message at the bottom when view job info.

Figure 6.10: JobMA "View info" window on OpenMoko NeoFreeRunner

1	"View" tab allows selecting the job info shown. If many fields are selected, not all of them will be displayed	Major: The user will miss some information
2	When a user selects "View" Job info option, a message at the bottom of the window is missed	Minor: It is not very important, but it could be useful to know the selected action
3	In View info window all details about the job do not appear because of the screen size	Medium: It would be very useful to include a horizontal scroll bar to be able to browse across all the information provided

Table 6.5: JobMA "View info" window weaknesses

How this could be improved?


Job information
No Service
12:42
File View Action Help
Start
Command Owner State Nodes
Job information
OK
Thanks to the scroll bar user can consult all Job info.
Useful scroll bar to consult all Job info.
Message to inform what is doing the user, consulting Job info in this case.

Job ID: 7d8c05f4-75cf-48c7-886e-b5aae58
Command: /usr/bin
Name:
Owner:
Status:
Sub. time: Tue Aug 18 12:37:5
Node:

You are consulting Job info

Figure 6.11: JobMA "View info" improved window on OpenMoko NeoFreeRunner

6.3.1.9 Running jobs with JobMA over OpenMoko

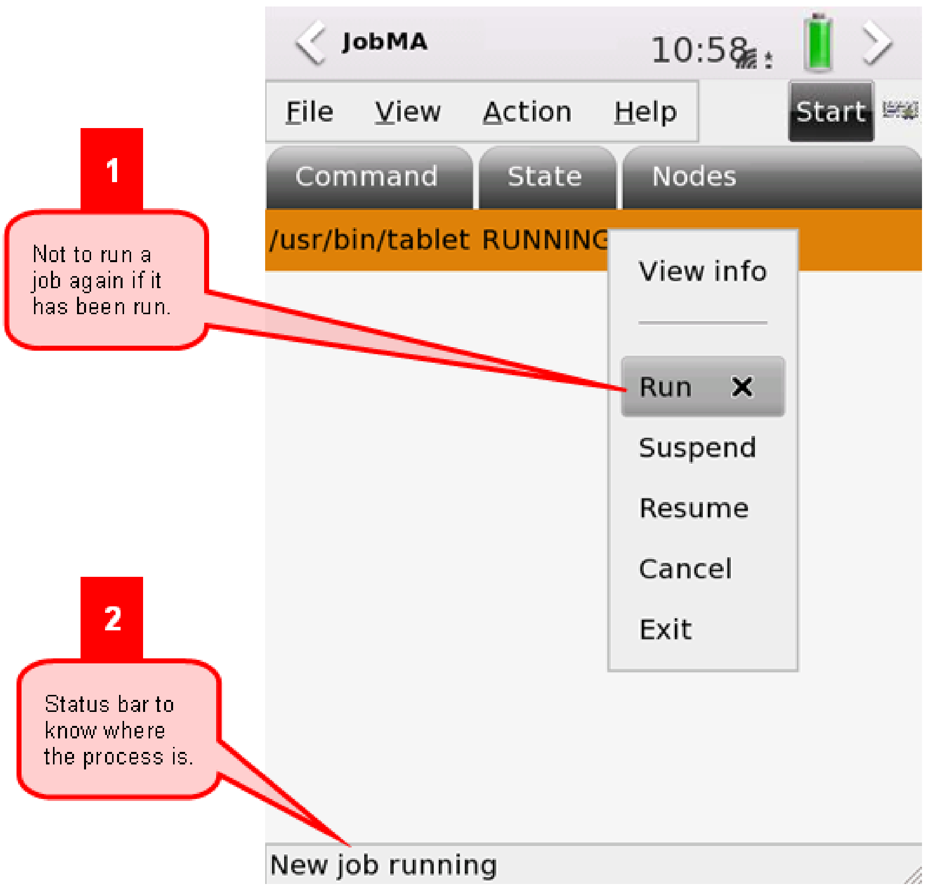


Figure 6.12: JobMA “Run a job” window on OpenMoko NeoFreeRunner

1	Run operation appears when double-clicking on a running job	Minor: Run operation should be disabled in order to not run again the job
2	When running a job, the user doesn’t know the next step to follow	Minor: A message at the bottom showing the different possibilities could help

Table 6.6: JobMA “Run a job” window weaknesses

How this could be improved?

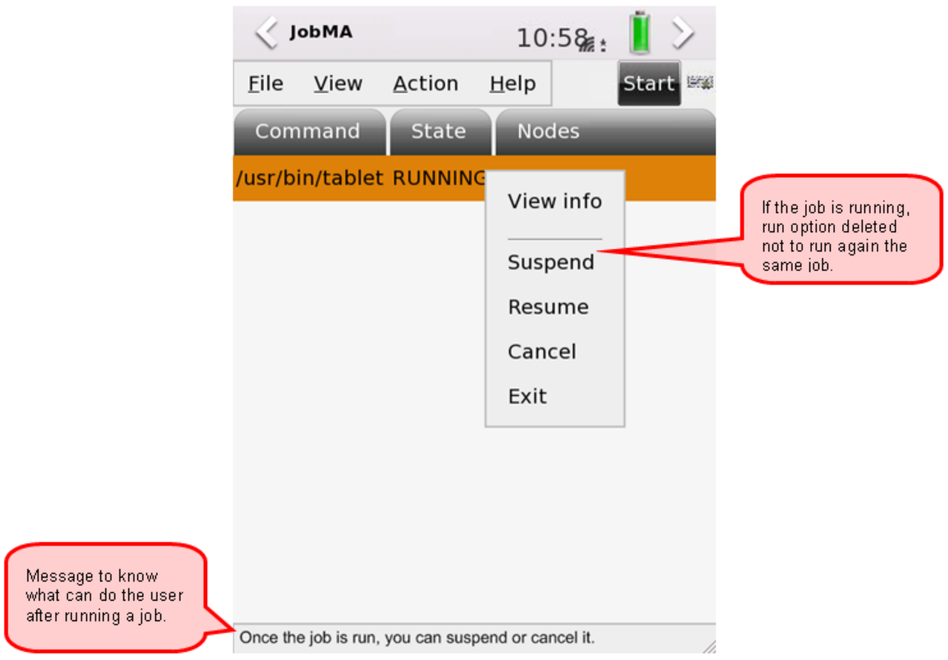
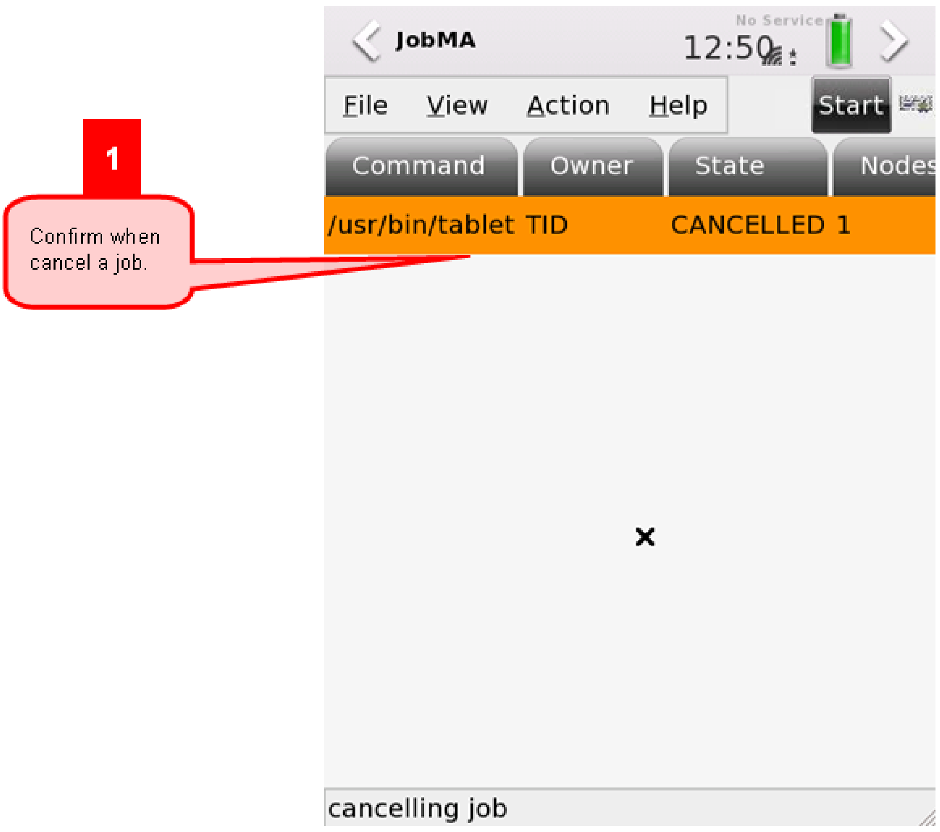

JobMA
10:58
File
View
Action
Help
Start
Command
State
Nodes
/usr/bin/tablet RUNNING
View info
Suspend
Resume
Cancel
Exit
If the job is running, run option deleted not to run again the same job.
Message to know what can do the user after running a job.
Once the job is run, you can suspend or cancel it.

Figure 6.13: JobMA “Run a job” improved window on OpenMoko NeoFreeRunner

6.3.1.10 Canceling jobs with JobMA over OpenMoko



/usr/bin/tablet TID CANCELLED 1

cancelling job

Figure 6.14: JobMA “Cancel job” window on OpenMoko NeoFreeRunner

1	After running a job, the user can cancel the execution by clicking on the Action -> Cancel option in the menu bar, but no confirmation is required, and the job will be automatically canceled	Medium: A new window requesting the confirmation could avoid some unintentional job cancellations

Table 6.7: JobMA “Cancel job” window weaknesses

How this could be improved?

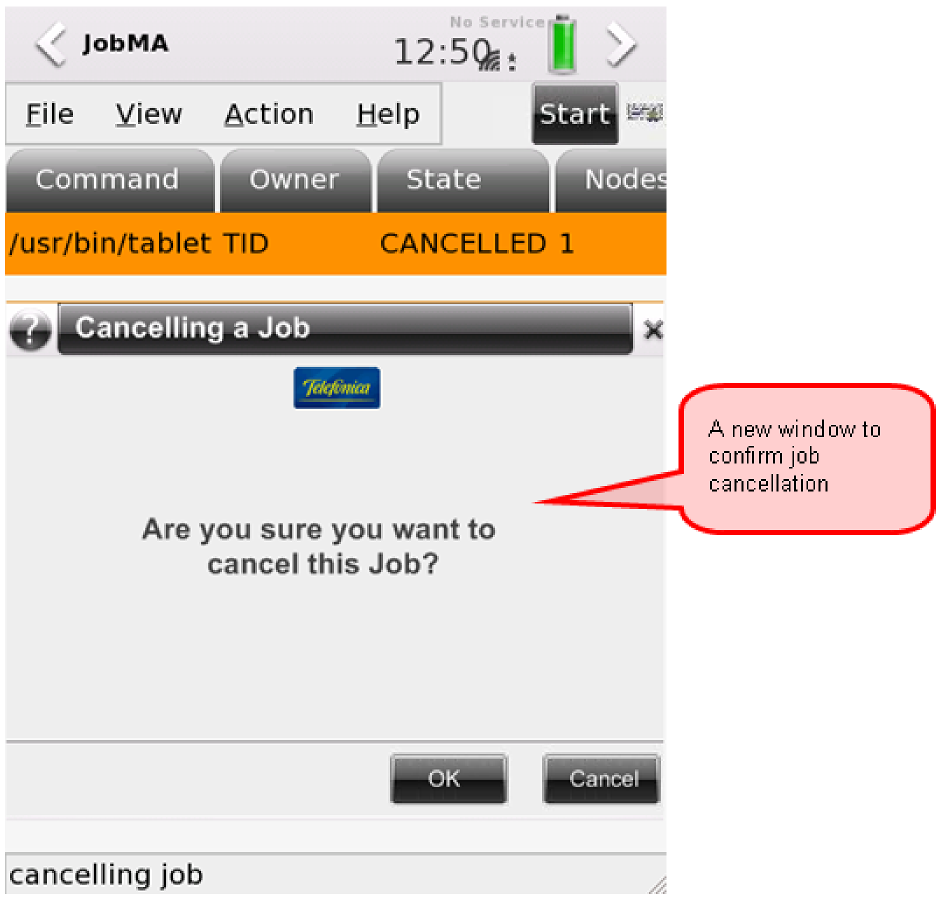

JobMA
No Service
12:50
File View Action Help
Start
Command Owner State Nodes
/usr/bin/tablet TID CANCELLED 1
Cancelling a Job
Telefónica
Are you sure you want to cancel this Job?
OK
Cancel
cancelling job
A new window to confirm job cancellation

Figure 6.15: JobMA "Cancel job" improved window on OpenMoko NeoFreeRunner

6.3.1.11 Final Assessment

There are some operations associated to JobMA application: load user certificate, open a JSDL, Define a Job, View Job info, run a Job and view this Job running. In the figure below, the ease of those operations have been punctuated:

- Load user certificate: when a user launches JobMA application, the user certificate is automatically loaded. This factor obtains a punctuation of 5 points out of 5 (this factor has been analyzed because in the first beta versions of JobMA applications the certificate needed to be manually loaded).

- Open A JSDL: open a JSDL to create a job is very easy; user only needs to click on the tab in the menu and find the file in the browser. This factor obtains a punctuation of 4 points out of 5

- Define a Job: when a user tries to create a job defining it, three fields (not very common for a end-user otherwise) should be filled. This factor obtains a punctuation of 3 points out of 5

- View Job info: view job info is very easy, user only has to click on the menu and a new window will appear. This factor obtains a punctuation of 4 points out of 5

- Run a Job: run a job is easy too, clicking on the option "Run" on the menu. This will run the job automatically, so this factor obtains a punctuation of 4 points out of 5

- Monitoring a running Job: user can consult info about the running jobs just by clicking on the option in the menu or double-clicking on the job. This factor obtains a punctuation of 4 points out of 5

Finally, we have concluded that managing JobMA application in a mobile device is very easy and it should not be a problem for the users.

6.3.2 XtreemOS-MD installer

6.3.2.1 XtreemOS-MD installer Strengths

- Very easy to find the installer
- Quick download thanks to the minimal installer size
- Very easy to install
- Quick installation, it is only necessary a few steps to install it

6.3.2.2 XtreemOS-MD installer Areas of improvement

- Not every device supported by this installer
- No graphical installer for every mobile devices

6.3.2.3 Summary of the evaluated scenarios

The following figure shows the level of ease in four different scenarios, depending on the action the user does during the installation of the XtreemOS-MD installer: getting the XtreemOS-MD installer, downloading it, trying to install it and finally following the instructions and steps needed to finish the installation. All these aspects are shown in the next figure with values between 1 and 5 points, depending on the ease of use in each case.

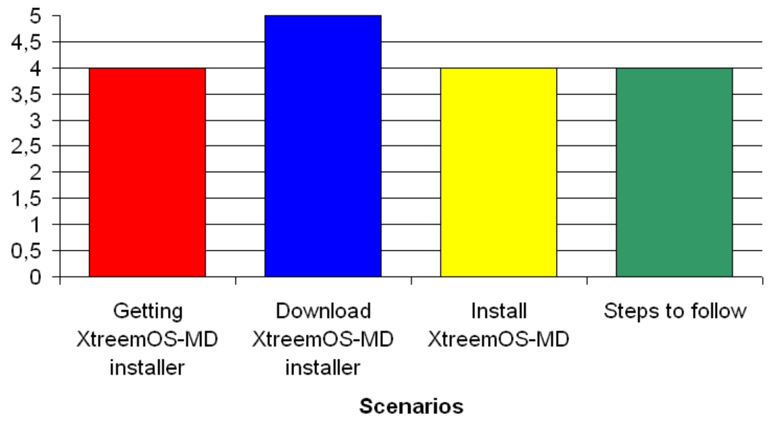
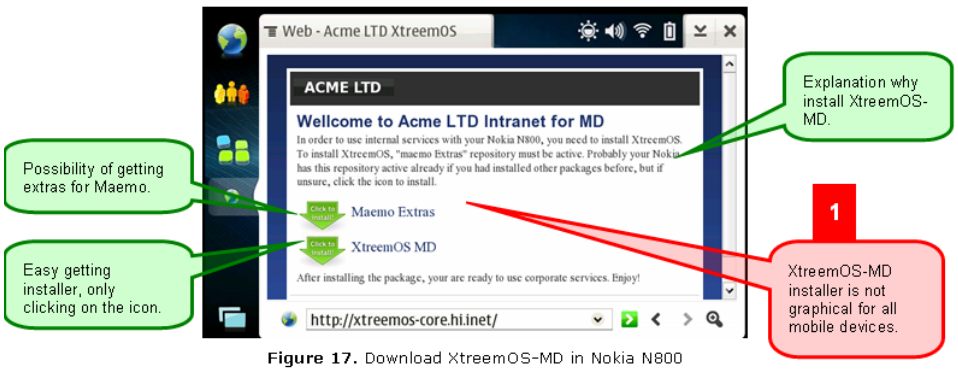


Figure 6.16: XtreemOS-MD installer evaluation result, scale 1-5 (1 very difficult and 5 very easy)

It's worth noting that only Nokia N800 with Maemo has a graphical interface to install XtreemOS-MD, so this will be the option we are going to analyze. However, Openmoko NeoFreerunner does not provide any graphical interface: the user should just execute a script, previously created by the developers, and provide user and password on the grid to fully install XtreemOS-MD.

6.3.2.4 Getting and downloading XtreemOS-MD installer on a Nokia N800


Web - Acme LTD XtreemOS
ACME LTD
Wellcome to Acme LTD Intranet for MD
Click to install!
Maemo Extras
Click to install!
XtreemOS MD
http://xtreemos-core.hi.inet/
Explanation why install XtreemOS-MD.
Possibility of getting extras for Maemo.
Easy getting installer, only clicking on the icon.
1
XtreemOS-MD installer is not graphical for all mobile devices.
Figure 17. Download XtreemOS-MD in Nokia N800

In order to use internal services with your Nokia N800, you need to install XtreemOS. To install XtreemOS, "maemo Extras" repository must be active. Probably your Nokia has this repository active already if you had installed other packages before, but if unsure, click the icon to install.

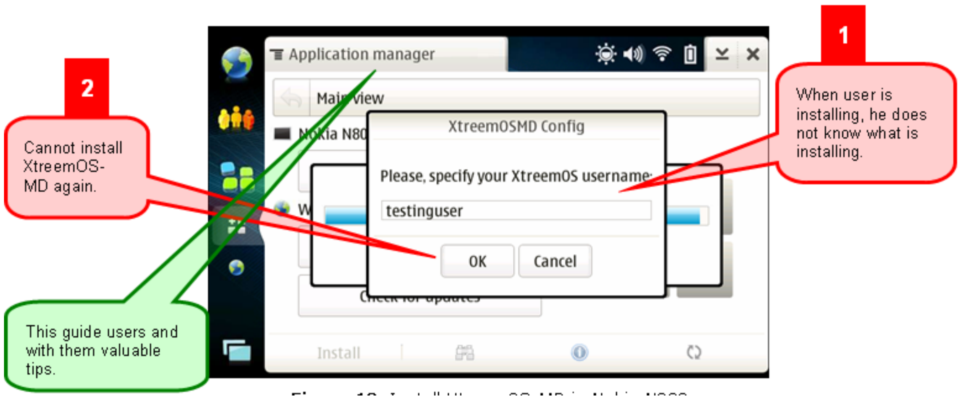
After installing the package, your are ready to use corporate services. Enjoy!

Figure 6.17: Downloading XtreemOS-MD on a Nokia N800

1	In some mobile devices, XtreemOS-MD cannot be installed graphically. In these cases, the user should execute a concrete installation script	Medium: it's easier installing from a graphical interface for a normal end-user

Table 6.8: Downloading XtreemOS-MD weaknesses

6.3.2.5 Installation of XtreemOS-MD on a Nokia N800


Application manager
Main view
Nokia N80
XtreemOSMD Config
Check for updates
Install
OK
Cancel
2
Cannot install XtreemOS-MD again.
1
When user is installing, he does not know what is installing.
This guide users and with them valuable tips.

Please, specify your XtreemOS username:

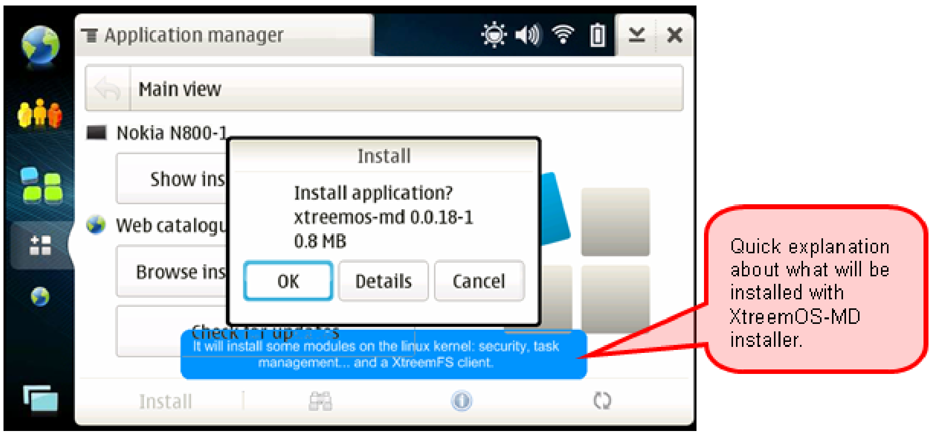
testinguser

Figure 6.18: XtreemOS-MD installer window on Nokia N800

1	When the user tries to install XtreemOS-MD, he doesn't know exactly what it's really being installed	Medium: Even if finally XtreemOS-MD works, the process is not much transparent for end user
2	A user cannot install XtreemOS-MD twice in the same mobile device without re-flashing it, as there are problems while uninstalling	Medium: Problematic if the user uninstall XtreemOD-MD unintentionally

Table 6.9: XtreemOS-MD installer window weaknesses

How this could be improved?



It will install some modules on the linux kernel: security, task management... and a XtreemFS client.

Quick explanation about what will be installed with XtreemOS-MD installer.

Figure 6.19: XtreemOS-MD installer improved window on Nokia N800

6.3.2.6 Final Assessment

The main operations related to XtreemOS-MD installation are: getting the installer, downloading it, installing it and following the steps to complete the installation. In the figure below, the ease of those four operations have been scored. Let's analyze each operation and the score obtained:

- Getting XtreemOS-MD installer: getting this installer is really easy for end user, who just needs to access to the web page for downloading the installer. The punctuation obtained is 4 points out of 5

- Downloading XtreemOS-MD installer: downloading XtreemOS-MD installer is very easy because, after the user has accessed to the previous page, he only has to click on the icon provided to download the installer. The punctuation obtained is 5 points out of 5

- Install XtreemOS-MD: installation is not very difficult and user can follow it easily. The punctuation obtained is 4 points out of 5
- Steps to complete the installation: steps to complete installation are very common and user only has to click on the install button. Moreover, he has only to provide user and password to authenticate in the grid. The punctuation obtained is 4 points out of 5

Finally, we have concluded that the installation of XtreemOS-MD in a mobile device is very easy and it should not be a problem for the potential users.

Chapter 7

Conclusion

The deliverable introduced the new set of WP4.2 and then put the main emphasis on the evaluation of XtreemOS. The evaluation was carried out in four different test categories: evaluation of installation and configuration, evaluation of XtreemOS components, comparison with other Grid solutions and usability evaluation of the XtreemOS MD flavor. In the following, the test summaries of these test categories shall be re-visited and directions for future work will be given.

The first category of evaluations conducted a long-term survey and gave insights into the experiences with two public and three intermediate internal releases from the end-user perspective. The survey examined the satisfaction with the installation, configuration and basic usage of the install CDs provided as well as with the accompanying documentation. In all four categories, one could detect a remarkable improvement with respect to end-user satisfaction. The ratings increased monotonously along all XtreemOS versions examined, most noticeable, however, is the leap made with the introduction of the public release of XtreemOS 2.0. Early major problems with lacking integration, instability, complicated manual setup, bugs and lacking synchronization between software development and documentation have been addressed to a far extend. Advancements with software integration have been reported and also the automatized installation and configurations tools have been added which render the adoption of the new OS much easier. One further major reason for improved satisfaction was the revised documentation which provides for more clarity, completeness and corrected many errors. Also the separation into a user and an admin guide is highly appreciated. Further ways for improvements have been proposed. This includes debugging of various packages which still fail to install. Also the number of manual steps and work-arounds should be reduced, e.g. by introducing configuration scripts and further self-explaining GUIs, e.g., for the xosautoconfig tool. Furthermore, it would be useful to introduce lower level automation for certificate setup. Basic usage would benefit from more detailed error outputs and from ensuring a proper startup of all services which would avoid manual restart. And finally, it is suggested to further work on the quality of the documentation, e.g. by adding more screenshots and by reducing the need to jump between sections. It is planned to extend the long-term survey to also cover forthcoming XtreemOS and to track the evolution of end-users ratings across the XtreemOS history.

The second category of tests consists of the evaluation of XtreemOS components including node-level VO support, checkpointing and restart, DIXI message bus, XtreemOS API, Distributed Servers, Virtual Nodes, application execution management, data management, security services and the mobile device flavor. In the following, the results per component shall be summarized.

Among others, the tests covered the evaluation of the web interface VOLifeCycle, which worked as expected and thus satisfied all the requirements. Some issues have been detected, e.g., the command-line interface does not use the XtreemOS user certificates nor has any provision to log in, resulting in local root being able to do anything and no other user being able to use it at all. The former cannot be avoided, as local root can also edit the database manually etc, thus VOlife must run on a trusted node. The latter, however, limits the use of the command-line interface to test environments only. This shows that the CLI is provided just as a test tool. A VO administrator in a large grid environment thus has no practical means to automate repetitive administrative tasks. Furthermore, the administrator will have to scroll through long tables when managing users, groups, and roles because no search method is provided. Only the list of VOs has the option of filtering. Further tests have been done to analyze the performance test of VOlife, which shows that its scalability over large databases is good but not excellent. However, VOlife cannot really be a bottleneck of a grid environment because it does not affect performance of other XtreemOS components. The account mapping performs as expected in normal usage, i.e. until the account pool is used up. Distinct grid users are mapped to distinct dynamically created local accounts. The same user acting as a member of two distinct VOs is also mapped to distinct accounts to prevent VO interference. Members of the same VO are mapped to the same local group. The only detected problem was that once the account pool is used up two different users can be mapped to the same local account, thus they are not isolated - e.g., one of them can control jobs created by the other. It can be can concluded that the components for node-level VO support in XtreemOS Release 2 are adequate, with all major functional requirements met but performance and usability leaving some room for improvements. All tests will also be repeated on later releases of XtreemOS to check for regressions.

The experimentation completed for checkpoint/restart functionality provided good coverage of the features provided by XtreemOS. All main functions relating to checkpointing worked adequately, despite a number of limitations: 1) kernel-level checkpointing is unable to checkpoint complex, multi-process applications, 2) container functionality was not available in the 2.0 release, 3) graphical applications are not checkpointable in any form. Despite these limitations, checkpointing has been shown to have good performance and scalability, as shown in the kernel-level testing, and the functionality provided by the checkpointing and migration of containers provides a useful alternative to larger virtualisation formats. This functionality holds significant benefit for a number of WP4.2 applications, particularly Rule-based System Management (RBSM) from SAP. As the RBSM application is able to make decisions based on the state of the systems under its control, checkpointing and migrating processes and containers between nodes allows for a powerful additional layer of control for the application. Later releases should use a similar approach to testing for comparisons, and also include a wider range of experimentation, including other test applications not used in these experiments.

The evaluation of the DIXI message bus and staging environment measured the performance in terms of the time it takes to serve a request. The tests involved various scenarios that we based on the features provided in the DIXI runtime. We performed benchmarking in set-ups representing each of these scenarios. As competitor middleware and messaging bus we used CORBA. The results show that in the current version of DIXI there is a penalty for the middleware's added functionality that manifests as increased time needed for each invocation to finish with a result. This is due to the fact that, during the development of DIXI, a lot of emphasis was put on enriching the framework with features, while the optimising of important aspects of the solution and speeding up the message exchange has been planned for later stages of the development. The speed-ups can be achieved, for instance, by replacing the generalised service message marshalling with a content-aware one. It is also planned to examine the possibilities to speed up the Apache Mina library used for the networking.

The XtreemOS API (XOSAGA) has been evaluated with the three programming languages supported: C++, Java, and Python. For each of these implementations, their performance overhead (compared to directly using the underlying service interfaces) has been investigated. With the micro benchmarks for each of the implementations, it was shown that the performance overhead caused by using XOSAGA is negligible for job submission and modest for namespace operations and file I/O. In the case of Python, file I/O was even faster than the native counterpart. Further evaluations with XOSAGA will be done with the ported versions of the WP4.2 applications COMPSs (from BSC) and openTurns (from EDF). At the time of writing this deliverable the application-centric experiments performed by WP4.2 failed because of bugs and missing documentation of XOSAGA. With the ported applications it shall be possible to evaluate the usage of the interfaces and the scalability and performance overhead from the application perspective. The prospected tests should also include large-scale performance tests on Grid5000 and interoperability tests with other Grid solutions.

Further tests were carried out with Distributed Servers which are a unique feature of XtreemOS, so there is no comparable other system against which it coudl have been evaluated. Instead two implementations of distributed servers have been compared: an old, non-portable kernel-based version, and the new portable user-level version. The user-level implementation is demonstrated to work well, although it is in certain cases significantly slower than the old version. However, no particular effort has been spent on optimizing it yet, so important performance gains are expected in future versions.

The experiments with Virtual Nodes have shown that this component is about one order of magnitude slower than Java RMI. Furthermore, it was presented that the runtime overhead of passive replication (as implemented so far), is slightly higher than for active replication. This gap might turn out to be more serious in the case that deterministic schedulers are used for active replication. Finally, the evaluation showed that an application's state size barely has an impact on service availability. Yet, for more replicas the availability increases drastically. Evaluation has given some hints on what to improve. First of all multiplexing of connections or at least re-using of open sockets would be beneficial to the overall system performance. Second, overload renders the system unusable or may even lead to the failure of individual replicas or the entire Virtual Node. This makes the system vulnerable for denial-of-service attacks. Thus, adding mechanisms that are able to handle overload and support graceful degradation in such situations are desirable extension. Third, it is not clear which set-up is the best alternative in case of node failure. Here, further investigation and evaluation is required.

The evaluatio with Application Execution Management (AEM) gauged the functional requirements, evaluated its performance and compared it to the performance of Globus toolkit. Tests were performed to assess the suitability of XtreemFS as a platform for running web server We also performed tests to evaluate the difference of performance between XtreemOS PC flavor and a grid running the Condor middleware. The robustness of the AEM scheduler was also tested. Our tests also included comparative perfromance analysis of job submission responce times of XtreemOS and Globus. We also evaluated power computational performance depending on client flavour. Following our tests we conclude that that AEM in XtreemOS Release 2 are adequate. We also have proven that XtreemOS is a suitable environment for hosting web services and applications. Our tests have shown that the perfromance of XtreemOS and Condor system is comparable, and for some tests like testing management times, XtreemOS performs better. With regard to the robustness of AEM, we discovered that only a part of the jobs submitted reaches the Done final Status, which shows that the robustness of AEM scheduler still leaves the room for improvement. The results of comparative performance analysis obtained in the job submission and execution at XtreemOS and Globus show a great improvement over Globus in this basic unit.

Test with XtreemFS have been carried out to examine the functional requirements, evaluate its performance and compare it to the performance of NFS. The functional requirements of XtreemFS were also tested in the simulated at real wan environment. Tests were also performed to assess functional requirements of Object Sharing Service provided by the XtreemOS. We also evaluated the performance of Object Sharing Service provided by XtreemOS. The perfromance of kDFS was evaluated using the Bonnie benchmark. It was found that POSIX complience tests passes 89.4% of the adequate tests. Non-parallel IO tests reveal that for sequential access pattern XtreemFS does not provide performance improvements as compared to NFS. Parallel IO tests reviel that the performance of XtreemFS scales well with the increase in the number of OSDs. XtreemFS also shows very good scalability under transactional load and the load generated by enterprise search application. It effectively caches big application IO work set, utilizing its de-facto distributed cache based on OSDs. Even without using SSD non-volatile memory, it enables to almost reach the throughput of the baseline filer technology that uses SSD non-voletile memory. We also found that applications using XtreemFS are capable to work under wan-simulated latency. It was found that Object Sharing Service may be used to share the application state among nodes in the wan conditions. With regard to kDFS perfromance evaluation we found that the read/write performance improves when the block size grows. kDFS successfuly passed the file creation and deletion tests of the Bonnie benchmark, which control system stability and POSIX compliance. It means that kDFS is a stable and POSIX-compliant file system. Tests were also performed to compare the perfromance of kDFS and NFS. The tests show that kDFS outperforms NFS almost by factor 2. Based on the tests we conclude that POSIX Compliance leaves the room for improvement. While currently the MRC is the bottleneck for writes, this limit will probably be overcome in the upcoming XtreemFS release, which is expected to cache file size updates. Write latency under the transactional load is still very low as compared to the baseline NetApp filer technology. And most probably this shortcoming may be overcome only by means of using SSD non-voletile memory at OSD nodes. Our experimental results under transactional load looks very promising and suggest that XtreemFS may support transactional load. However more experiments need to be performed to support this conclusion. We also experimentally assessed that XtreemFS does not yet include mechanism to counter packet corruption in the wan conditions. Based on the tests perfromed for kDFS we conclude that it is POSIX compliant. We also conclude that the global cooperative cache of kDFS enables it to cache the application work set and write the data locally to disk. As the result kDFS outperforms NFS filesystem at our tests. We can conclude that XtreemFS, Object Sharing Service in XtreemOS Release 2 and kDFS are adequate, with all major functional requirements met but performance and some functional requirements leaving some room for improvement. Further testing is required for features skipped in this iteration, most of which relate to comparative performance analysis of XtreemFS with kDFS and with other advanced research file systems such as CEPH. Another set of tests to be performed in the next iteration will target read/write replication and using it to overcome failures such as network partitioning and to improve performance of applications in wan conditions.

The goal of the evaluation of the security evaluation was to make sure that the security services do not become bottlenecks and unjustified overheads for the other features of XtreemOS. Moreover, it is critical to ensure that security services do not introduce security problems for the overall system. Firstly, the DTokens used for delegation proved to be more efficient than the standard proxy certificates used by more established Grid infrastructures. Secondly, the VOPS has linear performance with regards to the number of simultaneous policies that can be handled. Finally, the impact on CPU and memory of operating the CDA server was also tolerable, with a consumption between 3 and 8% during the servicing of 100 sequential requests. Further evaluation and ongoing improvement of the security services can only be accomplished through the use of rigorous, runtime vulnerability assessment. There is still some work to be done in the area of managing the processes of issuing and updating policies and certificates. In addition, there is more scenario-based testing to be done of the VOPS, where different cases like monitoring and usage control will be investigated. This work will be done in conjunction with the demonstrator activities.

The tests with the mobile device (MD) flavor passed without major problems. The functionalities promised by XtreemOS-MD, IMA and JobMA applications have been verified and from the performance tests we can conclude that the performance of XtreemOS-MD as a Grid client executed from a mobile device like a Nokia N800 is similar to the one achieved by the XtreemOS PC client. On the other hand, execution of jobs in the Grid and even access to the Grid file system are faster than the local execution or local file system access to the terminal, which exemplifies the benefits of using the Grid from a mobile device (specially when thinklng on scenarios like the video conversion one). The advanced version of XtreemOS-MD, including support for smartphones and some additional features where the context awareness and resource sharing are the most important ones, is currently under development. This new release will extend XtreemOS-MD not only in the number of devices supported, but also on the scope of the software, as the mobile device will become a *light* resource of the Grid and not just a mere client. New tests will be designed and executed to evaluate the new functionalities, specially concerning the "offline-mode" operation and the resource sharing from the mobile device.

The third category, the comparison between XtreemOS and other Grid approaches, was conducted in two ways: As theoretical comparison and experimental comparison. The theoretical comparison assessed the various appproaches first from a general perspective, then XtreemOS was contrasted to various Grid middleware solutions and related Grid operating systems. The comparison showed that XtreemOS unifies many Grid features in a operating system which avoids a complex stack of middleware layers which could affect the speed and manageability of the Grid system landscape. The experimental comparison summarized the results of the comparative experiments with 1) XtreemOS-AEM and Globus-GRAM and 2) XtreemOS-DTokens and Globus-GSI, where XtreemOS could achieve superior performance measure in both cases. Finally, the setup of currently executed application-centric large-scale tests (on Grid5000) with Galeb (from XLAB) executed on XtreemOS and Globus GT4.0 was described.

The forth and final evaluation category consisted of analyzing the usability of the mobile device application JobMA from WP4.2 and with the mobile device flavor installer. The following operations of JobMA application have been examined: load user certificate, open a JSDL, Define a Job, View Job info, run a Job and view this Job running. For all operations, the JobMA application received high usability ratings and it can be concluded that managing JobMA application in a mobile device is very easy and it should not be a problem for the users. The operations examined with XtreemOS-MD installation were: getting the installer, downloading it, installing it and following the steps to complete the installation. Also here, best usability ratings have been recorded and it was concluded that the installation of XtreemOS-MD in a mobile device is very easy and it should not be a problem for the potential users.

Forthcoming evaluations by WP4.2 will put further emphasis on evaluating the requirements beyond the technical assessment. The test results so far in the deliverable have been quantitative and aimed at evaluating the functional correctness and requirements satisfaction of the various XtreemOS features. However, there are various reasons for qualitative evaluation of systems, especially new systems, which cannot be obtained from pure quantitative study. As the project approaches its end and the software product becomes increasingly mature and complete, we plan to add a further business-oriented value-based assessment of the requirements.

The evaluation reports and in particular the results of the evaluation are being communicated through various channels, including wiki pages, mailing lists, bug trackers, phone calls and finally this deliverable. All intermediate and final test documents are made available on the internal SVN server. We hope that the evaluation provides useful feedback to the development work packages and we are looking forward to a successful continuation of the cooperation.

Chapter 8

Acknowledgments

We would like to thank the developers in SP2 and SP3 and in particular the respective work package leaders for their cooperation regarding the identification of interesting test scenarios and for their support during the XtreemOS installation. We are looking forward to a successful continuation of this cooperation.

Bibliography

[1] G. Allen, K. Davis, T. Goodale, A. Hutanu, H. Kaiser, T. Kielmann, A. Merzky, R. van Nieuwpoort, A. Reinefeld, F. Schintke, T. Schütt, E. Seidel, and B. Ullmer. The Grid Application Toolkit: Towards Generic and Easy Application Programming Interfaces for the Grid. *Proceedings of the IEEE*, 93(3):534–550, 2005.

[2] P. Asadzadeh, R. Buyya, C. L. Kei, D. Nayar, and S. Venugopal. Global grids and software toolkits: A study of four grid middleware technologies. *CoRR*, cs.DC/0407001, 2004.

[3] A. Barak and A. Shiloh. The MOSIX2 Management System for Linux Clusters and Multi-Cluster Organizational Grids. Technical report, Hebrew University of Jerusalem, March 2007.

[4] SPECweb2005 benchmark. Specweb2005 benchmark. Website, 2008. http://www.spec.org/web2005/.

[5] SPECweb2009 benchmark. Specweb2009 benchmark. Website, 2009. http://www.spec.org/web2009/.

[6] R. Berlich, M. Hardt, M. Kunze, M. Atkinson, and D. Fergusson. Egee: building a pan-european grid training organisation. In *ACSW Frontiers '06: Proceedings of the 2006 Australasian workshops on Grid computing and e-research*, pages 105–111, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[7] S. Burke, S. Campana, P. Méndez Lorenzo, C. Nater, R. Santinelli, and A. Sciabà. GLITE 3.2 USER GUIDE. EGEE, 2009. Document identifier: CERN-LCG-GDEIS-722398, https://edms.cern.ch/file/722398/1.2/gLite-3-UserGuide.pdf.

[8] XtreemOS consortium. Design and implementation of node-level vo support. XtreemOS deliverable D2.1.2, 2007.

[9] XtreemOS consortium. Evaluation report and revision of application requirements. XtreemOS deliverable D4.2.5, 2008.

[10] XtreemOS consortium. Evaluation report and revision of application requirements. XtreemOS deliverable D4.2.4, 2008.

[11] XtreemOS consortium. Reproducible Evaluation of a Virtual Node System. XtreemOS Deliverable D3.2.9, 2008.

[12] XtreemOS consortium. Extended Version of a Virtual Node System. XtreemOS Deliverable D3.2.14, 2009.

[13] T. Cortes, C. Franke, Y. Jégou, T. Kielmann, D. Laforenza, B. Matthews, C. Morin, L. P. Prieto, and A. Reinefeld. Xtreemos: a vision for a grid operating system. Technical report, XtreemOS Technical Report # 4, 2008.

[14] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6). RFC 2460, December 1998.

[15] N. Drost, E. Ogston, R. V. van Nieuwpoort, and H. E. Bal. ARRG: Real-World Gossiping. In *Proceedings of The 16th IEEE International Symposium on High-Performance Distributed Computing (HPDC)*, Monterey, CA, USA, June 2007.

[16] I. Foster. Globus toolkit version 4: Software for service-oriented systems. In *IFIP International Conference on Network and Parallel Computing*, LNCS 3779, pages 2–13. Springer-Verlag, 2006.

[17] Globus toolkit compute grid tutorial. Website, 2006. http://www.globusconsortium.org/tutorial/.

[18] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, A. Merzky, J. Shalf, and C. Smith. A Simple API for Grid Applications (SAGA). Grid Forum Document GFD.90, Open Grid Forum (OGF), 2007. Version 1.0 http://forge.ogf.org/short/saga-core-wg/saga-core-v1.

[19] A. S. Grimshaw, W. A. Wulf, and CORPORATE The Legion Team. The legion vision of a worldwide virtual computer. *Commun. ACM*, 40(1):39–45, 1997.

[20] IEEE. IEEE standard for software test documentation, ieee 829-1998. IEEE Computer Society, 1998.

[21] E. Jeanvoine, C. Morin, and D. Leprince. Vigne: Executing easily and efficiently a wide range of distributed applications in grids. In *Euro-Par*, pages 394–403, 2007.

[22] D. Johnson, C. Perkins, and J. Arkko. Mobility Support in IPv6. RFC 3775, June 2004.

[23] S. Kortas. Résolution haute précision des équations de navier-stokes sur machines parallèles à mémoire distribuée. Phd thesis, Univer-sit'e de Provence, Centre de Mathematiques et d'informatique.I, France, 1997. http://samuel.kortas.free.fr/DOCS/THESE/these_Samuel_Kortas.pdf.

[24] S. Kortas and P. Angot. Parallel preconditioners for a fourth-order discretization of the viscous Bürgers equation. In P. E. Bjørstad, M. Espedal, and D. Keyes, editors, *Proceedings of the 9th international conference on domain decomposition method*, pages 387–405, 1998. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.23.4449&rep=rep1&type=pdf.

[25] Amazon Web Services LLC. Amazon elastic compute cloud (amazon ec2). Website, 2009. http://aws.amazon.com/ec2.

[26] Amazon Web Services LLC. Amazon simple storage service (amazon s3). Website, 2009. http://aws.amazon.com/s3.

[27] J. Mehnert-Spahn, T. Ropars, M. Schoettner, and C. Morin. The architecture of the xtreemos grid checkpointing service. In *Euro-Par '09: Proceedings of the 15th International Euro-Par Conference on Parallel Processing*, pages 429–441, Berlin, Heidelberg, 2009. Springer-Verlag.

[28] E. Miloševi, M. Novak, M. Pihlar, and G. Pipan. Grid-based solution for financial modeling. In *MIPRO 2006. [Vol. 1], Microelectronics, Electronics and Electronic Technologies/MEET. Hypermedia and Grid Systems/HGS*, pages 253–256, Rijeka, Croatia, 2006.

[29] A. Mirkin, A. Kuznetsov, and K. Kolyshkin. Containers checkpointing and live migration. In *Ottawa Linux Symposium*, 2008.

[30] A. Mirtchovski, R. Simmonds, and R. Minnich. Plan 9 – an integrated approach to grid computing. In *International Parallel and Distributed Processing Symposium (IPDPS)*, April 2004.

[31] R. Nou, J. Giralt, J. Corbalan, E. Tejedor, J. O. Fito, J. M. Perez, and T. Cortes. Xtreemos application execution management: A scalable approach. In *Submitted to CCGRID'10*, 2010.

[32] R. Nou, S. Kounev, F. Juliá, and J. Torres. Autonomic QoS control in enterprise grid environments using online simulation. *J. Syst. Softw.*, 82(3):486–502, 2009.

[33] OpenVZ. Openvz. Website, 2009. http://wiki.openvz.org/.

[34] P. Padala and J. N. Wilson. Gridos: Operating system services for grid architectures. In Timothy Mark Pinkston and Viktor K. Prasanna, editors,

HiPC, volume 2913 of *Lecture Notes in Computer Science*, pages 353–362. Springer, 2003.

[35] M. Pfeil. Optimising and Self-adaptive Strategy Selection in a Replication Framework. master thesis VS-D07-2009, Institute of Distributed Systems, Ulm University, Germany, 2009.

[36] R. Pike, D. Presotto, S. Dorward, B. Flandrena, K. Thompson, H. Trickey, and P. Winterbottom. Plan 9 from Bell Labs. *Computing Systems*, 8(3):221–254, 1995.

[37] M. V. Reddy, A. V. Srinivas, T. Gopinath, and D. Janakiram. Vishwa: A reconfigurable p2p middleware for grid computations. In *ICPP '06: Proceedings of the 2006 International Conference on Parallel Processing*, pages 381–390, Washington, DC, USA, 2006. IEEE Computer Society.

[38] B. Sotomayor. The globus toolkit 4 programmer's tutorial. Website, 2005. http://gdp.globus.org/gt4-tutorial/multiplehtml/index.html.

[39] M. Szymaniak, G. Pierre, M. Simons-Nikolova, and M. van Steen. Enabling service adaptability with versatile anycast. *Concurrency and Computation: Practice and Experience*, 19(13):1837–1863, September 2007. http://www.globule.org/publi/ESAVA_ccpe2007.html.

[40] Apache Tomcat. Apache tomcat. Website, 2008. http://tomcat.apache.org.

[41] S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson. Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile. RFC 3820, June 2004.

[42] A. Vahdat, T. Anderson, M. Dahlin, E. Belani, D. Culler, P. E., and C. Yoshikawa. WebOS: Operating system services for wide area applications. In *Proceedings of the Seventh Symposium on High Performance Distributed Computing*, 1998.

[43] M. van Steen, P. Homburg, and A. S. Tanenbaum. The architectural design of Globe: A wide-area distributed system. Technical Report IR-422, Vrije Universiteit, Amsterdam, Netherlands, 1997.

[44] VMWare. Vmware. Website, 2009. http://www.vmware.com.

[45] J. P. Walters, V. Chaudhary, M. Cha, S. Guercio Jr, and S. Gallo. A Comparison of Virtualization Technologies for HPC. In *Proceedings of the 22nd International Conference on Advanced Information Networking and Applications*, pages 861–868. IEEE Computer Society, 2008.

[46] Xen. Xen. Website, 2009. http://www.xensource.com.

[47] XtreemOS. First prototype version of ad hoc distributed servers. Deliverable D3.2.2, November 2007.

[48] XtreemOS. Reproducible evaluation of distributed servers. Deliverable D3.2.6, December 2008.

[49] XtreemOS. Extended version of the distributed servers platform. Deliverable D3.2.11, December 2009.

[50] XtreemOS Consortium. First Draft Specification of Programming Interfaces. Deliverable D3.1.1, November 2006.

[51] XtreemOS Consortium. Second Draft Specification of Programming Interfaces. Deliverable D3.1.2, November 2007.

[52] XtreemOS Consortium. Third draft specification of programming interfaces. Deliverable D3.1.5, November 2008.

[53] XtreemOS Consortium. Third Prototype of XtreemOS Runtime Engine. Deliverable D3.1.8, May 2009.

[54] XtreemOS Consortium. Third Prototype of XtreemOS Runtime Engine. Deliverable D3.1.8, May 2009.

[55] E. Y. Yang and B. M. Matthews. DTokens: A Lightweigth and Traceable Delegation Architecture for Distributed Systems. In *Proceedings of the 28th IEEE International Symposium on Reliable Distributed Systems*, pages 107–116, 2009.