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
D4.2.7

Due date of deliverable: August 31st, 2010
Actual submission date: October 15th, 2010

Start date of project: June 1st 2006

Type: Deliverable
WP number: WP2.2, WP2.3, WP3.1, WP3.3, WP3.4, WP3.5, WP3.6, WP4.2
Task number: T2.2.13, T2.3.8, T3.1.5, 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 7.0 / Last edited by Bernd Scheuermann / 15/10/2010

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	31/05/2010	Bernd Scheuermann	SAP	Initial version
2.0	17/09/2010	Maik Jorra, Barry McLarnon, Philip Robinson, Bernd Scheuermann	SAP, ZIB	Finished insertion of evaluation of XtreemOS for Cloud Computing
3.0	24/09/2010	Bernd Scheuermann	SAP	Finished documenation of installation and configuration
4.0	29/09/2010	Marjan Sterk, Guillaume Pierre, Ales Cernivec, Florian Müller, Primoz Hadalin, Enric Tejedor, Santiagi Prietro, Alvaro Arenas, Alvaro Reol, Louis Rilling, Matthieu Fertré, Oriol Fito, John Mehnert-Spahn, Michael Sonnenfroh, Roman Talyansky, Matej Artac, Ronald Fowler, Ian Johnson, Guillaume Alleon, Lokendra Singh, Michael Schöttner, Yvon Jégou, Peter Iszak, Zhouyi Zhou	BSC, EADS, ICT, INRIA, KER, SAP, STFC, TID, UDUS, VUA, XLAB	Finished insertion of test documentation for components evaluation
5.0	29/09/2010	Massimo Coppola, Barry McLarnon, Roman Talyansky, Bernd Scheuermann, Ronald Fowler, Samuel Kortas, Marjan Sterk, Santiago Prietro, Maik Jorra, Guillaume Alleon, Oriol Fito, Enric Tejedor, Michael Sonnenfroh, Yvon Jégou	BSC, CNR, EADS, EDF, INRIA, SAP, STFC, TID, UDUS, XLAB, ZIB	Finished insertion of application descriptions
6.0	29/09/2010	Bernd Scheuermann	SAP	Included introductory text and Executive Summary. Submitted to internal review.
7.0	15/10/2010	all authors	all teams	Internal review, proof reading, corrections.

Reviewers:
Jan Stender (ZIB)

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*
T3.3.14	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 evaluation carried out by WP4.2 has guided the development of the XtreemOS software, has driven bug fixes and performance improvements and has also given recommendations for short- and longterm product decisions. As in the previous deliverable, the evaluation is supplemented by the experiments performed by developers in SP2 and SP3, in order to have a common evaluation document from the entire consortium. The work done by WP4.2 focuses on the application-oriented evaluation from the end-user perspective, whereas the contributions from SP2 and SP3 put emphasis on measuring the performance on a lower technical level.

The deliverable at hand introduces the extended set of applications which were used for experiments with XtreemOS. We also report how these applications can contribute to the dissemination and exploitation of XtreemOS. Subsequently, the evaluation results are presented sub-divided into three categories: evaluation of installation and configuration, of XtreemOS components and of XtreemOS as foundation for Cloud Computing. The first category continues the longterm survey of user experience with XtreemOS packaging in the form of XtreemOS install CDS starting from the very first release (v1.0) up to the latest release (v3.0 beta 2) at the date of conducting the last survey. Thereafter the evaluation of XtreemOS components comprises in-depth experiments with the following technologies: Node-level VO support, checkpointing and restart, Linux SSI, DIXI message bus, XtreemOS API, Resource Selection Service, Application Execution Management, Data Management, Security Services and the Mobile evice Flavor. In the third category, we evaluate possible ways of using XtreemOS as a foundation for Cloud Computing in two scenarios: XtreemOS supporting the management of cloud systems and XtreemOS as the basis for an exemplary implementation of an online photo archive in the Cloud.

The documentation of all tests include detailed specifications which shall deliver comprehensive and reproducible test setups. The results obtained are analyzed and corresponding feedback is provided to XtreemOS developers. Finally, the deliverable concludes with a summary of the test results.

Contents

Chapter 1

Introduction

This deliverable reports on the results of the XtreemOS evaluation which has provided feedback to developers, packaging providers and project management. The evaluation is structured into three different categories:

1. evaluation of installation and configuration,
2. evaluation of XtreemOS components,
3. evaluation of XtreemOS as foundation for Cloud Computing.

The first category extends and continues the evaluation of XtreemOS packaging. This evaluation was executed in the form of a long-term end-user survey gathering their experience with the various XtreemOS install CDs. It started with the very first public XtreemOS version 1.0, several intermediate releases and the public release 2.0. In this deliverable, the evaluation is extended including version 2.1.2 and release 3.0 beta 2. The latter version was the latest release at the date of executing the last survey. The strategy applied allow for monitoring the change of results as the software evolves during project execution. Furthermore, the survey collects 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 evaluation of XtreemOS components is devoted to extensive testing of XtreemOS technology including:

- Node-level VO support
- Checkpointing and restart
- Linux SSI
- DIXI message bus
- XtreemOS API

- Resource Selection Service
- Application Execution Management
- Data Management
- Security Services
- Mobile Device Flavor

The emphasis of these experiments is set on evaluating the performance, scalability, stability and correctness of the respective developments. In addition to WP4.2, development work packages in SP2 and SP3 contributed to planning, specification, execution and documentation of the experiments. Each development work package introduced a dedicated task (cf. DoW) and devoted manpower to carry out the performance evaluation. In each test unit, we indicate the partners and work packages responsible. As a general rule, WP4.2 mainly conducts application-centric tests from the end-user's view whereas SP2 and SP3 focus on lower-level performance benchmarking.

In the final category, the evaluation focuses on the benefits of using XtreemOS technology as foundation for Cloud Computing. The last deliverable D4.2.6 [10] addressed XtreemOS as a solution for Grid Computing comparing XtreemOS with competitive Grid Middleware technology. In the deliverable at hand, we give an assessment of XtreemOS used for enabling (or supporting) the management of cloud environments and the development of cloud services.

The remainder of this deliverable is structured as follows: Chapter 2 gives an updated overview of the applications used for end-user oriented experimental evaluations during the project execution period. It also introduces new applications that have been added by various partners during the project extension phase. We give short descriptions per application, report on application development and porting work and outline how the applications can contribute to the exploitation and dissemination of XtreemOS. The test setup and results of the long term survey of the evaluation of XtreemOS installation and configuration are provided in Chapter 3. Chapter 4 gives the test plans, specifications and results of the evaluation of the various XtreemOS components. The evaluation of XtreemOS as basis for Cloud Computing is presented in Chapter 5. Finally, the evaluations are re-visited and summarized in Chapter 6.

Chapter 2

Application Descriptions

A wide range of applications from different domains have been used for end-user oriented experimental evaluations during the project execution period. Further applications have been added by various partners for the project extension phase. The following section gives an overview of these applications. Afterwards, we give a short description per application, address application development and porting aspects and finally outline how the application can contribute to the exploitation and dissemination of XtreemOS.

2.1 Overview

Table 2.1 provides an overview of the current set of applications in WP4.2 . Further applications have been added by various partners for experiments during the project extension phase. These applications are marked by (*). The total set of 22 applications covers a wide spectrum of fields with practical relevance. Hence, the perspectives of end-users from different industrial and academic domains can be respected during evaluation.

2.2 Hmmpfam on COMP Superscalar (BSC)

BSC contributes to XtreemOS with a COMP Superscalar (COMPSs) enabled version of the hmmpfam application. Both COMPSs and hmmpfam are explained in the next subsections.

2.2.1 Application overview

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.

Table 2.1: Applications in WP4.2. Applications marked by (*) have been added for experimentation during the project extension period

Partner	**Application Name**	**Short Name**	**Application Area**
BSC	COMP Superscalar	COMPSS	Bio-informatics
BSC	SpecWeb	SPECWEB	Enterprise solutions
CNR	XOS-GATE Tomographic Applications (*)	XOSGATE	Numerical simulations
EADS	Amibe/JCAE	JCAE	Computer aided engineering
EADS	openFOAM (*)	OPENFOAM	Computational Fluid Dynamics
EADS	Paraview (*)	PARAVIEW	3D data analysis and visualization
EADS	Elfipole	ELFIPOLE	Electromagnetics
EDF	Moderato/Maestro	MODERATO	Particle physics
EDF	OpenTURNS	OPENTURNS	Sensitivity and reliability analysis
EDF	Zephyr	ZEPHYR	Fluid mechanics
EDF/INRIA	Salomé (*)	SALOME	Numerical simulations
INRIA	Environmental Application (*)	ENVIRON	Environmental simulations
SAP	SAP NetWeaver Search and Classification	TREX	Enterprise solutions
SAP	SAP MaxDB replayer	MaxDB	Enterprise solutions
SAP	Rule-based System Management	RBSM	Enterprise solutions
STFC	BioLinux Application (*)	BIOLINUX	Bio-Informatics
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

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.

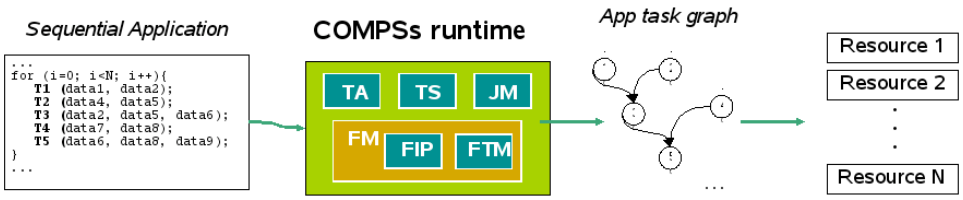


Figure 2.1: Execution of a COMPSs application.

Figure 2.1 depicts the COMPSs application execution scenario.

2.2.2 Application Development and Porting

The original version of the COMPSs was built on top of the JavaGAT API. JavaGAT is the Java version of the Grid Application Toolkit [1], which is a generic and flexible API for accessing Grid services from application codes, portals and data management systems. The calls to the GAT API are redirected to specific adaptors which contact the Grid services, thus offering a uniform interface to numerous types of Grid middleware. COMPSs invokes the JavaGAT API to request the submission of jobs and the transfer of files. In several series of tests conducted in the MareNostrum supercomputer [6], the hmmpfam application was proven to run properly on top of COMPSs.

For COMPSs-hmmpfam to run on XtreemOS, the COMPSs runtime had to be ported to the XtreemOS APIs. Figure 2.2 shows the architecture of COMPSs-hmmpfam on top of these APIs.

First, COMPSs was ported to the AEM XATI API, from which it exploited the job management and resource management features. While the former was present in JavaGAT, the latter was not supported and so it was a new feature added to the COMPSs runtime. Second, the COMPSs runtime was ported to XOSAGA, also exploiting the job and resource management capabilities.

At the end of the project, the hmmpfam application has been successfully tested on the Grid5000 environment and runs on top of COMPSs, in both the porting to AEM and SAGA.

2.2.3 How the Application can Contribute to the Exploitation of XtreemOS

The COMPSs-hmmpfam application, ported to the AEM and SAGA APIs, benefits from the job management and resource management features offered by XtreemOS. In addition, the resource management was not implemented before in the COMPSs runtime, and therefore it is a new feature provided by XtreemOS.

Hmmpfam is a widely used bioinformatics application, and as a result of the COMPSs porting on top of AEM and SAGA, it can now potentially run in any cluster where COMPSs and XtreemOS are installed. However, the use of COMPSs

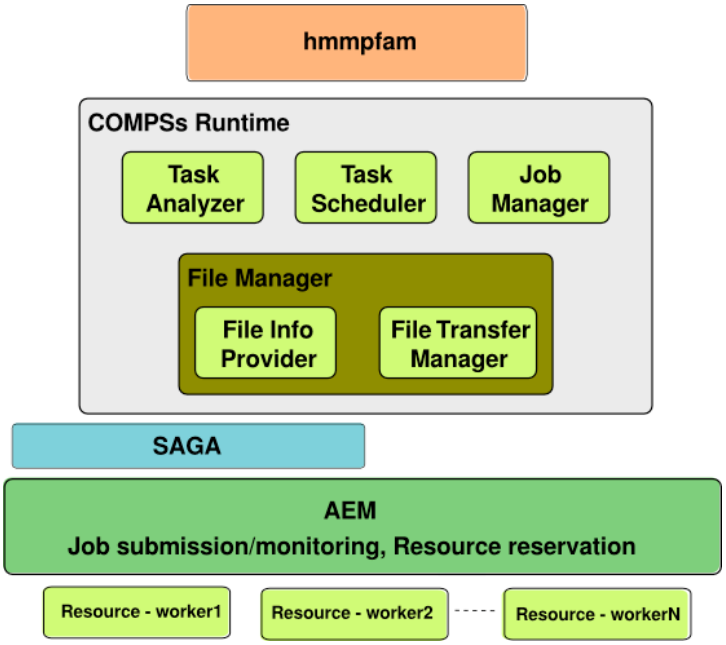


Figure 2.2: Architecture of COMPSs-hmmpfam on top of XtreemOS.

in XtreemOS is not limited to hmmpfam; a Java application which is programmed following the guidelines of the COMPSs would be suitable to run in XtreemOS.

2.2.4 How the Application can Contribute to the Dissemination of XtreemOS

COMPSs provides a simple programming model for applications to be executed on top of XtreemOS, exploiting its job management and resource management features. This exploitation, however, is hidden to the programmer.

Concerning the dissemination of XtreemOS, the COMPSs-hmmpfam application appears in a paper about the Application Execution Management of XtreemOS, which was submitted to the 11th ACM/IEEE International Conference on Grid Computing (Grid 2010):

- XtreemOS Application Execution Management: A Scalable Approach. R. Nou, J. Giralt, J. Corbalan, E. Tejedor, J.O. Fito, J.M. Perez and T. Cortes.

2.3 SPECweb2005 (BSC)

2.3.1 Application overview

SPECweb [3] [4] is a benchmark for evaluating the performance of World Wide Web Servers. It is recognized by the community as a representative benchmark

for measuring the system's ability to behave as a web server. Nowadays, web applications in general are changing a lot and, for this reason, the SPEC corporation decided to include many sophisticated and state-of-the-art enhancements to cope with the modern demand of Web users of today and tomorrow: user think time between requests (i.e. the time between user interactions), and conditional GET requests to simulate browser caching effects, among others.

It simulates a web server using dynamic web pages and a backend database. In fact, it provides three new workloads based on analysis of real web server logs. This is due to both the high variability in security demands and the differences in dynamic content in various web server workloads. As a result, SPECweb2005 incorporates three different workloads:

1. *Banking*, based on Internet personal banking. All requests in this workload are based on SSL.

2. *E-commerce*, which represents the workload characteristics of an e-commerce site. This workload includes SSL and non-SSL based requests.

3. *Support*, based on the characteristics seen in sites used to download patches for computer support. Its aim is to test the ability of a web server to download large files. No SSL is used in this third workload.

Moreover, the benchmark architecture has four major logical components (illustrated in Figure 2.3):

- *Client*. The clients run the application program, which sends requests to and receives responses from this one.

- *Prime client*. This is the component that initializes and controls the behaviour of the whole testing process. Therefore, its main actions are to initialize both web server and back-end simulator, and to collect and store the results of the benchmark tests.

- *Web server or System Under Test (SUT)*. It is the collection of hardware and software that handles the requests issued by the clients. Note that we use Apache Tomcat [41] as the server to be evaluated.

- *Back-End Simulator (BeSIM)*, is the logical component that is aimed to emulate a back-end application server that the web server must communicate with in order to get back dynamic content needed to complete HTTP responses.

2.3.2 Application Development and Porting

The SPEC corporation released the SPECweb benchmark (version 2005 [3]) before the starting of the XtreemOS project. Nowadays, an updated version (i.e.

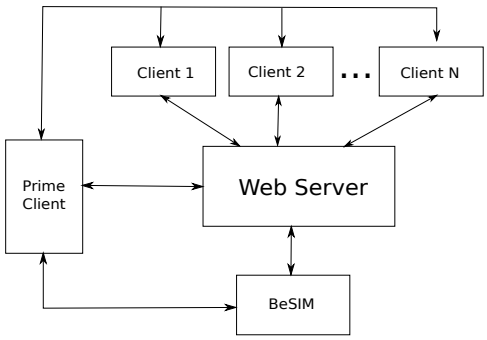


Figure 2.3: SPECweb2005 benchmark architecture.

2009) was released [4]. The only change among these versions is the inclusion of a power workload, which actually is based on the E-commerce workload, and a performance/power metric. However, this new workload is not significant for us because our work in XtreemOS does not include any energy/power issue. As a result, we use the SPECweb2005 version, but actually all the evaluation performed would be the same if we were using the new version.

Moreover, the application in question has not needed any new development in order to be used in the XtreemOS system. In this regard, all the benchmark's components are based on Java code and, therefore, its portability to XtreemOS is immediate. Nonetheless, we have developed a set of scripts with the aim to perform the tests desired.

2.3.3 How the Application can Contribute to the Exploitation of XtreemOS

As a matter of fact, and taking into account the complex execution environment needed by the benchmark, XtreemOS opens a wide range of opportunities. Hence, we consider this promising Grid system as both an evaluation and hosting platform. On the one hand, its distributed nature is very useful to run tests using any number of nodes (thus demonstrating the scalability of XtreemOS), and the resource discovery for acquiring those resources needed allows us to reduce the time of configuring whichever execution of any magnitude. On the other hand, XtreemOS can be used as a hosting platform in which we can scale the amount of resources that the web server under test has. In this way, the server will be able to scale its performance according to the scalability of the underlying resources.

Concretely, SPECweb2005 takes benefit from the AEM component of XtreemOS by means of:

- *Resource management*. We can reserve as many resources as a test of any dimension need. Additionally, we are able to scale the total amount of resources needed by a single component of the application, i.e. the server to be evaluated.

- *Job management*. All the components of the benchmark are executed and monitored by means of the job submission and monitoring mechanisms of AEM, respectively.

As a result, this benchmark on XtreemOS can be used after the project as an evaluation and a hosting platform.

In addition, we aim to evaluate the container-based checkpointing / restore mechanism of XtreemOS (i.e. OpenVZ). In this regard, our goal is to test the suitability of this mechanism for checkpointing and restoring the web server component of the benchmark, namely System Under Test (SUT), that runs inside a Java Virtual Machine (JVM).

2.3.4 How the Application can Contribute to the Dissemination of XtreemOS

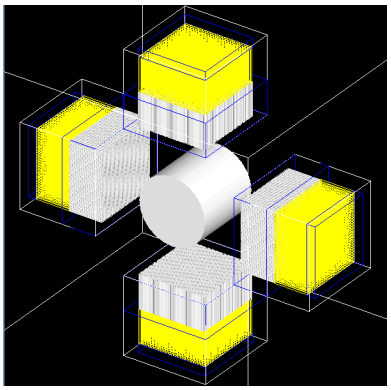
The XtreemOS features that can be demonstrated by SPECweb2005 are the aforesaid AEM component and the container-based checkpointing mechanism. Regarding dissemination actions, in the General Technical Meeting (March 2009), we presented a demonstration of how the SPECweb2005 benchmark uses AEM in order to perform web servers testing. A similar demonstration was recorded and the resulting video was uploaded on the official project's webpage. In addition, we contributed to a paper [29] with some of the results obtained from the experimentation carried out.

Obviously, all these results obtained were included in proper WP4.2 deliverables of XtreemOS.

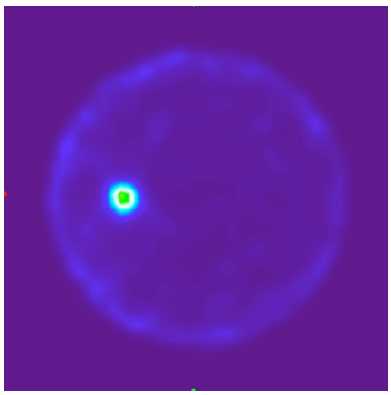
2.4 XOS-GATE Tomographic Application (CNR)

Single photon emission computational tomography (SPECT) is an imaging technique for the in vivo volumetric analysis of the distributions of radio-pharmaceuticals. The analysis employs radionuclides which emit single or multiple gamma rays during their diffusion in the living tissues. The distribution of gamma photons is detected at various positions by using one or more rotating gamma cameras (see Fig. 2.4(a)), and from this information a 3D image (see Fig. 2.4(b)) is then reconstructed by software [25].

In the last decade, SPECT imaging has made considerable progress, driven by the demands from medical and biological research. Simulations of SPECT acquisitions played a fundamental role, since they allow to design and simulate cameras in a virtual environment and to avoid the cost of constructing and testing physical devices.



(a) Simulation environment with four sensor, with a cylindrical sample phantom.



(b) Section of the 3D model resulting from a SPECT simulation of the phantom with a point-wise gamma source.

Figure 2.4: SPECT experimental settings and an example of final results

2.4.1 Application overview

Our application is in essence an XOS-aware parallelization of the GATE [18] software, which is currently widespread in the physics community for single photon computed emission tomography (SPECT). Our target was to provide enough computational power for (1) a realistic simulation of the whole process, from particle emission to final 3D reconstruction, and more important (2) a fully computational-based analysis of Tomographic devices, allowing to compute their system matrix and to develop and test new analytical methods to derive the system matrix from simulation data. The latter application contributes to lower the development cost of commercial SPECT devices, and to the investigation of open problems in applied physics research.

At a very high level, SPECT analysis is composed by two distinct processes: acquisition and reconstruction. For both phases there are consolidated computational techniques, which can be computational demanding depending on the experimental parameters. We focused on the computational simulation of the acquisition phase, where particle emission, deflection and absorption is reproduced by means of mathematical models.

The aim of simulating a SPECT analysis is to acquire the *system matrix* (or system response matrix) of a device, which depends on the whole device structure and is essential in order to reconstruct 3D images from the gamma sensor readings. The system matrix correlates the detection probability of the photons from each voxel in the image space to their projection bin in the projection space (e.g. the response measured from a specific pixel of the gamma sensor in a specific position). Figure 2.5 shows the process of converting the response to a set of point-wise sources, the point spread function, to a system matrix

The Applied Physics group of the University of Pisa which we collaborated

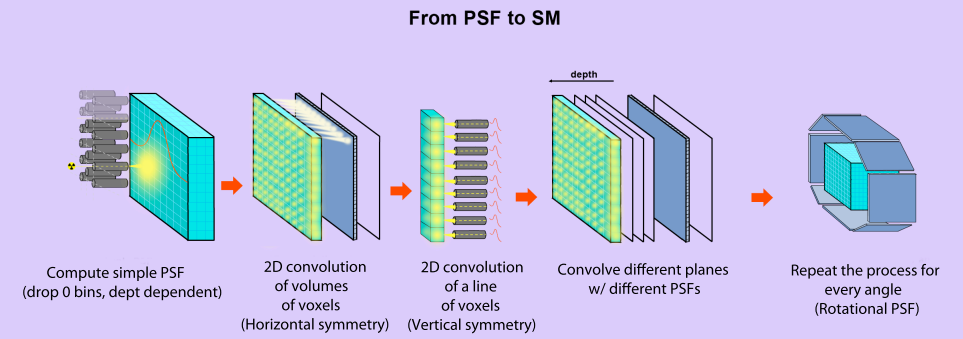


Figure 2.5: From point spread function to System Matrix

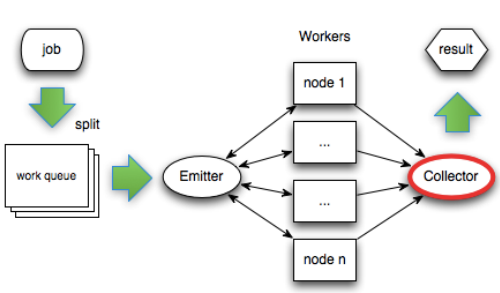
with in order to develop the GATE porting on XtreemOS is specifically focusing its experiment on system matrices of SPECT systems using *parallel hole collimators*. Collimators are metallic devices which are placed in front of the gamma cameras to enhance their directionality at the expense of their sensitivity.

GATE, the Geant4 Application [18], is an advanced open source Monte Carlo toolkit developed by the international OpenGATE collaboration [14], and it is dedicated to numerical simulation in medical imaging. GATE allows modelling of user-specific applications realistically by an easy and powerful macro language. It has been designed as to be flexible and precise, but still simulations remain computation-intensive, the presence of collimators usually increasing the load by more that two orders of magnitude.

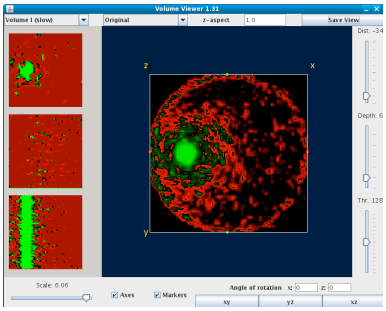
Architectural Overview Monte Carlo simulations of SPECT are CPU-intensive tasks, most of the work being due to the numerical calculations which simulate particle transport (ray-tracing) and physical interaction of the high energy photons with atoms of the subject and of the imaging device. Simulations often take up to several days or months to complete even with state-of-the-art single-CPU computers (e.g. in [36], a realistic computation is reported to take up to 25000 machine hours for producing statistically meaningful results). However, it is also generally true that long-running Monte Carlo simulations are excellent candidate for parallelization, given a computing platform that is big enough and fault-tolerant.

GATE already sports a parallel version by means of the *root* tool for splitting a gate macro into partial simulations. The existing tools only target HPC clusters, as it is a single-shot spawn of same-size subcomputations without any support for load balancing, heterogeneous machines, or failure tolerance. A shared file system and same administrative domain is, of course, also assumed.

We developed a different approach, where job splitting is performed to achieve a much finer grain, and the resulting set of tasks is dynamically scheduled on top of a large set of possibly heterogeneous XtreemOS nodes. The mechanisms built, shown in Fig. 2.6(a), is one implementation of the classical *task farm* paralleliza-



(a) XOS-GATE application architecture (simulation phase).



(b) 3D reconstruction from an XtreemOS/G5K run.

Figure 2.6: SPECT architecture and user interface.

tion pattern, often used for so-called *embarrassingly parallel* problems. The implementation exploits XtreemOS features and can deal with the specific details of the GATE subcomputations. The high-level application architecture is made up by an emitter node, a set of worker nodes and a collector node. The emitter allocates worker nodes from the XtreemOS platform, and acts also as the collector in our implementation. Several control threads in the emitter dynamically distribute subcomputations to the worker nodes in an on-demand fashion, and collect the related output. The on-demand work distribution evens up performance differences among the computing nodes, which is not unexpected in an XtreemOS platform, and balances the simulation load among the resources in use whenever network issues or other computations happen to slow down some of them.

The emitter node, which also acts as a result collector, can deal with missing results and malfunctioning worker nodes, thus accomplishing a fault tolerant behaviour. Lost subcomputations due to suddenly unavailable nodes are automatically reissued, and none of the pending tasks is lost even in case of a whole platform stop. As the intermediate data can reside on the XtreemFS file system, we exploit its transparent replication and fault tolerance features to keep safe consolidated results.

The overall fault tolerant behaviour of the XOS-GATE application is also used to run very long simulations in batches of a few hours, the parallel computation automatically resuming from its previous state. More computing nodes can be recruited, if necessary, by leveraging the XtreemOS AEM functionalities.

2.4.2 Application Development and Porting

The existing GATE application and the related tools are available as open source. While commonly used in the field, the GATE application is used and maintained by a community relying on powerful symmetric multiprocessor machines or dedicated clusters. As a consequence, not all LINUX distributions are equally supported in

all versions, with many cross-dependences among GATE and other supporting software libraries changing from version to version of the application, and interacting with those supported by the specific LINUX version. Simply recompiling the set of tools from scratch can thus be an issue, not to mention the fact that heterogeneous machine configurations are not managed by the program at all.

Porting activities As a first start, the GATE tool have been compiled and used on a local Ubuntu cluster. Concurrently, GATE porting on top of the XtreemOS Mandriva LINUX begun.

In order to set up a large enough XtreemOS platform, we resorted to deploying a large set of XtreemOS virtual machines on top of the Grid5000 (G5K) computing infrastructure. Since XtreemOS and its deployment on G5K were still evolving, several sub-steps were needed, porting XOS-GATE first to Debian LINUX on G5K, then to Mandriva and finally to XtreemOS/G5K. During this activity, several quirks and bugs of the GATE software were discovered and coped with.

The run-time has been gradually built implementing the emitter node functionalities, introducing the task farm behaviour and its load balancing and FT features. It was experimented on the different XOS-GATE prototypes (cluster and G5K based) as soon as the different platforms were ready, with small-size tests (a few machine hours) and then with large tests (thousands of machine hours).

Application status The resulting application is compatible with the XtreemOS distribution. Some system libraries needed have been pushed into XtreemOS 3.0, so that the application can be directly packaged with it and used as an XtreemOS demo. XOS-GATE has been used so far for several thousand hours reliably exploiting an XOS platform on top of G5K. The performance scalability is good, and results produced (see Fig. 2.6(b)) are being used by a Ph.D. student in Applied Physics as the base of her current research.

XOS-GATE has also been successfully run for relatively small tests on the XtreemOS permanent testbeds (development and open). A full test of XOS-GATE, not constrained to a single HPC platform like G5K (a large heterogeneous one, or at least a couple of different HPC platforms) needs to be performed for a full assessment.

2.4.3 How the Application can Contribute to the Exploitation of XtreemOS

Porting the application onto XtreemOS allowed us to relieve the aforementioned constraints of the pre-existing parallel version. Instead of building/renting dedicated HPC clusters for long-running simulations, the user can run them on a large dynamic platform, within a VO administrative domain, and exploiting XtreemFS to access data. The load balancing and fault tolerance techniques adopted makes many more hardware configurations available to run experiments that were formerly constrained to high-end HPC sites.

CNR is going to further develop the parallel management part of the simulation to extend it to other computational simulation applications which are only available for HPC clusters. CNR will exploit the work done to port more applications on top of the XtreemOS platform, and contribute to the development of XtreemOS as a shared HPC platform. The load balancing technique applied can be further integrated into the XtreemOS architecture, exploiting also the AEM functionalities for dynamic job spawning.

The XOS-Gate application just crafted is a tool to design new physical SPECT devices, that can push XtreemOS adoption among companies who work in the field, this including a spin-off based in Pisa and designing small size, high resolution SPECT scanners. In addition to this, the application is already being used on top of the G5k platform, the data being collected to study new, more efficient analytical methods to derive the system matrix of devices from computational simulations.

Current plans are to deploy an extension of the XtreemOS platform on top of the resources of the new IT center of the University of Pisa [30] and exploit those additional resources to continue the application development and use.

2.4.4 How the Application can Contribute to the Dissemination of XtreemOS

As it comes out from previous presentation, the XOS-GATE application can demonstrate several XtreemOS features, mainly:

- VO resource sharing on large platforms, and VO authentication and authorization mechanisms
- resource selection and management through the AEM and SRDS subsystems
- large-scale, transparent and fault tolerant file system access
- large-scale, high-throughput computing

The XOS-Gate application has been applied to produce data for an applied physics study, which was presented at the XtreemOS symposium co-located with Europar 2010. A first scientific paper is already in preparation, and more are planned on the system implementation, experimental evaluation and data analysis topics. We plan to submit works at a couple of upcoming Parallel computing conferences, but in general the scientific contribution coming from the current application will cover both the Computer Science and Applied Physics areas. The application will be ported and showcased in the IT center of the University of Pisa [30].

2.5 Amibe (EADS)

2.5.1 Application overview

AMIBE is an algorithm that converts geometry data into 3D mesh which is used in a variety of aerospace simulations. The originial implementation of AMIBE is obtained from jCAE. The algorithm is designed to create very large meshes and is able to mesh faulty geometries.

Since, the algorithm is both compute and data intensive, the algorithm is being ported as a grid-based application on XtreemOS to take advantage of distributed resources. The application makes use of XOSAGA API, in order to submit, run, control and monitor the meshing jobs on XtreemOS. The experiments and implementation were carried out on Grid5000 testbed of XtreemOS-2.1, France.

The algorithm reads OpenCASCADE geometries (BRep, STEP and IGES) and produces 3D finite element surface meshes.

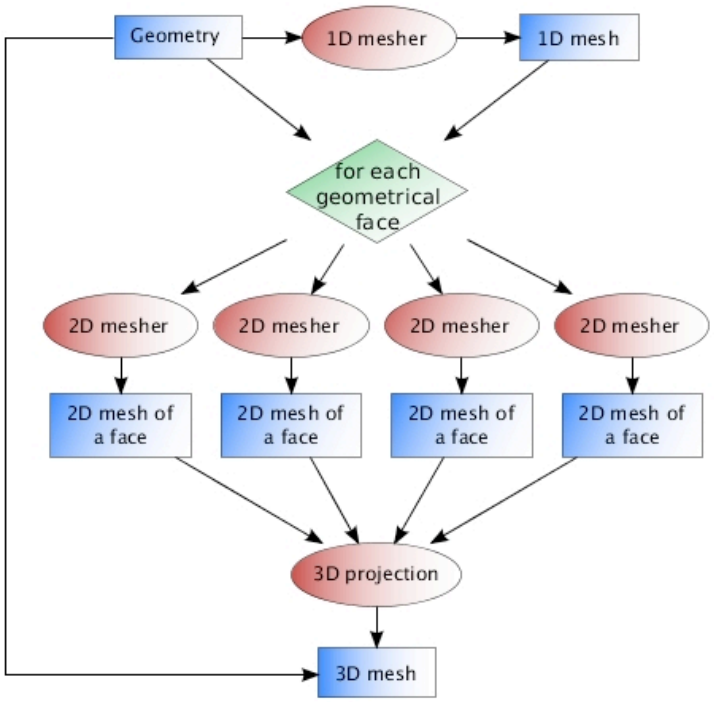


Figure 2.7: Amibe Algorithm

The flow diagram above explains in detail, how AMIBE creates the 3D meshes in 3 steps:

1. A descretization of geometrical edges. These are 1D linear elements (beams).

2. 2D discretization geometry. These are 2D triangles. Since, this is most costly step. it is being designed to run in parallel. In order to help in parallel computations, tessellation of different geometry patches are written into seperate files. Each face can be meshed without interaction from other faces.

3. A 3D discretization geometry: When finished, each mesh is written into seperate files, and merged to produce final mesh.

2.5.2 Application Development and Porting

The original implementation of Amibe algorithm was actually used to be serial implementation of the meshing steps, as explained above. In our design, we make use of parallel descretization of 2D geometry. That is, meshing of every face is created as a seperate job and submitted in parallel, to the XtreemOS.

For every step of meshing, we have seperate scripts (written in Groovy – A JVM based scripting language) let's say 'mesh1D.groovy', 'mesh2D.groovy' and 'mesh3D.groovy'. For merging the output of 1D and 2D meshing, a script called 'merge.groovy' is used.

A wrapper shell script to execute the above groovy scripts on Grid. Whenever, any groovy script is to be executed on Grid, a call to this wrapper script is made with the name of groovy script as argument, and this wrapper script runs that particular groovy script.The main reason for using a wrapper-shell script is to setup environment variables and initial configuration like setting up of groovy installation. (Although, theoretically such configurations can be done by using XOSAGA methods and setting up of attributes like JobDescription.ENVIRONMENT, we faced some problems with functioning of these attributes).

In addition, to above scripts, there is one driver groovy script which uses XOSAGA API to execute the meshing scripts on the grid (by making calls to the wrapper shell script), monitor the meshing, and retrieving the output back.

All three meshing scripts (mesh1/2/3D.groovy), the wrapper script and the merging script (merge.groovy) are stored onto the shared filesystem of XtreemOS, called as XtreemFS. This filesystem is shared by the participating nodes and hence the scripts which are to be executed on Grid are stored in this shared volume. The driver script is stored on the submission machine and is responsible for transferring the meshing, merging, wrapper scripts onto the XtreemFS volume. It then executing the scripts and finally retrieving the output back on the submission machine.

In addition to the meshing, merging, wrapper and driver scripts, we have geometry files (BREP, IGES and STEP format). These Geometry files are also stored onto XtreemFS, so that the meshing scripts can access the Geometry files during execution.

Steps of execution:

1. Execution starts with running the driver groovy script on the submission machine
2. The driver script transfers the other scripts (meshing, merging, CAD files) onto the XtreemFS volume.

3. ‘mesh1D.groovy’ is executed on the remote machine by making a call to wrapper shell script

```
 //Create and Run the Mesh1D
String jobname = "mesh1D";
Job job = jobService.createJob(jobDesc);
job.run();
monitorJob (job);
```

4. Once, 1D meshing completes, the result will be stored on the XtreemFS volume. In the next phase, parallel processing of 2D meshes will take place. The algorithm counts the total number of faces from mesh1D result data. It will then submit a job for each face.

```
 //Create the 2D Mesh for all faces
int iFaces = countFaces();
i = 1;
def jobList = [];
while ( i <= iFaces ) {
jobname = "mesh2D"+i;
Job job = jobService.createJob(jobDesc);
job.run();
jobList.add(job);
}
monitorJobs (jobList)
```

5. Once, the mesh2D operations have been completed, the results must be merged. The merging is done by executing the ‘merge.groovy’ on grid.
6. And finally, mesh3D operation takes place. ‘mesh3D.groovy’ is executed for 3D meshing
7. The driver script finally, retrieves the output back to the submission machine.

However, we also discovered that when submitting a large number of jobs in parallel, simultaneously, on a grid with large number of nodes, the failure rate of jobs was high. In our scripts, we had a mechanism to re-run a job if it fails, hence we were able to re-run the failed jobs. So, it is an overhead on both efficiency and the way in which a software is coded.

2.5.3 How the Application can Contribute to the Exploitation of XtreemOS

The application had a parallel processing component in 2D meshing. Submitting the meshing of every face as a seperate job in parallel makes use of the distributed resources and hence improve the performance.

Amibe meshing is a complex algorithm wih respect to computation and memory requirements. Using distributed resources helps delegating the compute and data intensive part of algorithm to distributed resources

2.5.4 How the Application can Contribute to the Dissemination of XtreemOS

The application makes use of XOSAGA API implemented in Java, for submission/monitoring of jobs, transferring files using Namespace package of SAGA. In addition, the application utilises the XtreemFS, the high speed shared file system of XtreemOS, for sharing data between different nodes.

A demo of application on XtreemOS is planned at the Slovene Chamber of Commerce with a workflow including other applications as well.

2.6 OpenFOAM (EADS)

2.6.1 Application overview

The OpenFOAM (Open Field Operation and Manipulation) CFD Toolbox is a free, open source CFD software package produced by a commercial company, OpenCFD Ltd. OpenFOAM has an extensive range of features to solve anything from complex fluid flows involving chemical reactions, turbulence and heat transfer, to solid dynamics and electromagnetics.

The core technology of OpenFOAM is a flexible set of efficient C++ modules. These are used to build a wealth of: solvers, to simulate specific problems in engineering mechanics; utilities, to perform pre- and post-processing tasks ranging from simple data manipulations to visualisation and mesh processing; libraries, to create toolboxes that are accessible to the solvers/utilities, such as libraries of physical models.

OpenFOAM is supplied with numerous pre-configured solvers, utilities and libraries and so can be used like any typical simulation package. OpenFOAM uses finite volume numerics to solve systems of partial differential equations ascribed on any 3D unstructured mesh of polyhedral cells. The fluid flow solvers are developed within a robust, implicit, pressure-velocity, iterative solution framework, although alternative techniques are applied to other continuum mechanics solvers.

Domain decomposition parallelism is fundamental to the design of OpenFOAM and integrated at a low level so that solvers can generally be developed without the need for any parallel-specific coding.

2.6.2 Application Development and Porting

We used Grid5000 as testbed for XtreemOS-based porting of OpenFOAM. During porting of OpenFOAM on Grid000, we discovered that there were some issues with the UserID's assigned to the grid-user as they were interfering with the

OpenFOAM working. Actually, the userId's assigned by XtreemOS deployment on Grid5000, use some illegal characters like hyphen ('-') and ('='), and OpenFOAM happens to perform a strict typechecking on the userId. The reason for such typechecking being, OpenFOAM at the core needs to know some user-home and user details, so it can more easily know where to tread in a multi-core environment. Hence, we had to patch the OpenFOAM (being an open-source tool), to ignore the specific typechecking on UserID and making it bit XtreemOS specific.

Following are the steps used to deploy OpenFOAM on XtreemOS

1. A definite number of nodes are reserved on Grid5000 and XtreemOS is deployed.
2. The OpenFOAM was installed on each of the participating nodes in the Grid.
3. The OpenFOAM testcases were stored on the grid-user's home directory on XtreemFS (The shared file-system of XtreemOS).
4. Along with the test cases, a wrapper shell script is also stored in the grid-user's home directory which actually runs the OpenFOAM test cases.
5. The wrapper shell script is submitted and run as job using Groovy script (a JVM based scripting language) using XOSAGA API. The wrapper script is actually responsible for running the OpenFOAM test-cases.

The tests for this application were made on the Grid flavor nodes of XtreemOS on Grid5000, on which it was successfully being ported. Since, there is no parallel component in the application, we believe that the user running the OpenFOAM application on a SSI cluster would benefit the most. However, we had issues with deployement of the SSI flavor of XOS image on Grid5000, so we could not test the application on a SSI cluster.

2.6.3 How the Application can Contribute to the Exploitation of XtreemOS

OpenFOAM is an application which typically uses high-memory and high-processing requirements. SSI capabilties of XtreemOS allows us to distribute and delegate the computation requirements over multiple nodes. The deployment of OpenFOAM on XtreemOS has minimal requirements, and it just uses existing Grid infrastructure to run OpenFOAM on XtreemOS. Hence, the application is easily deployable on any XtreemOS grid.

2.6.4 How the Application can Contribute to the Dissemination of XtreemOS

OpenFOAM porting on XtreemOS discovers the SSI capabilities of XtreemOS, hence applications similar to OpenFOAM, where parallel processing is inherent in

the application or users don't need to configure a software specifically for parallel processing, can also make use of distributed resources on XtreemOS using its SSI flavor.

2.7 Paraview (EADS)

2.7.1 Application overview

ParaView is an open-source, multi-platform data analysis and visualization application. ParaView users can quickly build visualizations to analyze their data using qualitative and quantitative techniques.

The data exploration can be done interactively in 3D or programmatically using ParaView's batch processing capabilities. It has an open, flexible, and intuitive user interface. Furthermore, ParaView is built on an extensible architecture based on open standards.

ParaView was developed to analyze extremely large datasets using distributed memory computing resources. It can be run on supercomputers to analyze datasets of terascale as well as on laptops for smaller data. The capability of Paraview to be capable of being run on a parallel platform is exploited in its deploymnet on XtreemOS.

ParaView is designed to work well in client/server mode. For parallel computations and parallel processing, Paraview uses MPI libraries. Hence, Paraview has to be built against MPI implementation (MPICH, OpenMPI etc.). The Paraview server is a parallel MPI program that must be launched as a parallel job, while the paraview client is a serial application.

2.7.2 Application Development and Porting

XtreemOS enables an user to use an XtreemOS grid as an MPI cluster and allows users to run MPI applications on it. XtreemOS provides some scripts which act as wrappers to 'mpirun'. The wrapper script is XOS_mpirun and also the scripts, mpirun.xtreemosmpi and mpirun.xtreemosmpipg, are provided which are subsequently called and executed. In addition, XtreemOS also provides a tool, 'xsubMPI', as submitting tool on cluster and to be used instead of rsh/ssh.

To run an MPI application on XtreemOS grid, user first needs to reserve required number of nodes and then use 'xsubMPI' and 'XOS_mpirun' (provided by XtreemOS) to submit and run the MPI executable across the nodes.

Hence, to exploit parallel processing capability of Paraview, it is first built against MPICH. Once, we have MPI enabled Paraview built, the Paraview server, which is a parallel MPI program, is launched as a parallel job using 'XOS_mpirun' (wrapper to mpirun). Once, the server has been started, Paraview client, which is a serial application, can be used to establish connection with the Paraview server and makes use of Paraview parallel processing.

During the implementation, we discovered that XtreemOS has some issues with running MPI applications as well as there were issues related with parallel submission and running of Jobs which will be fixed in version 2.1.2 of XtreemOS. Hence, with 2.1.2 or higher version of XOS, we will be able to make more concrete tests. In our deployment, we used Paraview will be used to visualize results of OpenFOAM computations.

2.7.3 How the Application can Contribute to the Exploitation of XtreemOS

The Paraview application inherently uses MPI libraries to parallelize its computation and XtreemOS allows the deployed grid to be used as MPI cluster with no additional setup or configurartion. Hence, the MPI applications make use of the existing infrastructure, user credentials like certificates, and configurations to run MPI application. This provides another advantage of an XtreemOS grid to be used as MPI cluster.

The deployment of Paraview on XtreemOS has very minimum requirements. Its just required to be built against MPICH and using the XtreemOS wrapper scripts, one can easily run parallel server on a XtreemOS grid.

2.7.4 How the Application can Contribute to the Dissemination of XtreemOS

The deployment of Paraview on XtreemOS can be used to promote the porting of many MPI applications, which need a specifically configured MPI cluster and configuration. Instead, the user can use the existing XtreemOS infrastructure and directly make use of parallelizations without much additional efforts on setup a cluster.

2.8 Elfipole (EADS)

2.8.1 Application overview

The ELFIPOLE software is designed to solve Maxwell equations in an unbounded domain of the space by the Boundary Equations method. The resolution is performed with direct or iterative solvers; in the case of an iterative solver we can possibly use the Fast Multipole Method (FMM) in order to speed-up the matrix-vector products.

Elfipole allows the solving of an electromagnetic problem with three kinds of possible sources:

- Incident plane waves
- Generators
- Spherical waves (the input is an antenna diagram)

2.8.2 Application Development and Porting

Running Elfipole in a distributed environment is fairly easy, as the Elfipole executables are compiled from static libraries and hence, have very minimal dependencies.

Elfipole is an MPI-based application which benefits from an existing MPI infrastructure. Hence, to benefit from the MPI based parallel component of Elfipole, we tried running Elfipole on XtreemOS Grid using the XtreemOS specific MPI wrapper scripts. But, we faced several issues with running it as MPI application on XtreemOS.

Finally, we decided to benefit from the kerrighed module of XtreemOS. We setup a two node SSI cluster, integrated with our XOS Grid.

The Elfipole jobs were submitted using the XOSAGA API to the head node of SSI cluster which by the benefit of kerrighed, made use of all the resources in the cluster.

2.8.3 How the Application can Contribute to the Exploitation of XtreemOS

The application could be a very good demonstration of running MPI tasks on the XtreemOS Grid, and using the grid as an MPI cluster, if the issues related with MPI are fixed in later versions of XtreemOS.

Nevertheless, our implementation of the application exploits the Kerrighed module of XtreemOS to use a set of nodes as Single System Cluster as well as it benefits from the SAGA API to submit jobs. Hence, it is a good demonstration of the SSI module of XtreemOS.

2.8.4 How the Application can Contribute to the Dissemination of XtreemOS

The application mainly benefits from the SSI flavor of XtreemOS and as well as makes use of the features like XOSAGA and XtreemFS.

The application is part of a CAE workflow and will be demonstrated at Slovene Chamber of Commerce.

2.9 Maestro (EDF)

2.9.1 Application overview

Maestro was designed to reproduce the footprint of Moderato, a radiographic modeling code – used internally at EDF – that combines a Monte-Carlo and a straight-line attenuation model. Moderato can simulate the behaviour of each photon by tracing their route from the light source up to the impact reached on the inspected object or on the argentic film. The combined result of thousands of these impacts gives a global image as would be obtained on a photographic film during actual radiographic inspection.

Maestro demonstrate how such simulations behave when launched and monitored using the job manager of XtreemOS.

Maestro is a framework entirely written in python. Through it, the user can sweep a parametric domain running elementary job for each combination of parameters possible. Maestro provides a consolidated view to start, stop or monitor all the jobs as a unique global study.

Maestro was already interfaced with job scheduler like Torque/Maui or LSF. On a recent workstation, it could also exploit all its cores by automatically detecting them and running the same number of parallel job. Since the end of 2009, we have been linking Maestro with the AEM modules.

While the template parametric application is placed in the home directory of a VO, Maestro create as much jobs as single parametric combination and submit them to XtreemOS via xsub. Results are stored into the VO home directory shared by all available resources thanks to XtreemFS.

2.9.2 Application Development and Porting

At this time, Maestro has only been tested with the 2.1 XtreemOS version. We observed that only a part of the jobs submitted reaches the *Done* final status, others stays either in a *Running* or *LocalSubmitted* status. This should be corrected in the last version of XtreemOS that we will tested as soon as we have access to a well-installed test bed.

2.9.3 How the Application can Contribute to the Dissemination of XtreemOS

Quite a large number of developers like to use python to test new features or concepts in quickly written mockups.

Maestro will give them a good example of job submission and monitoring from a python program.

2.10 OpenTurns (EDF)

OpenTURNS is an Open source initiative to treat uncertainties and risk statistics in a structured industrial approach.

Since the beginning of 2005, a partnership of three companies has been working on building together a tool designed to perform uncertainty treatment and reliability analysis. Sources are under GNU Lesser General Public License where as all documentation are under GNU Free Documentation License

OpenTURNS is a Unix/Linux software with three main components :

- a scientific C++ library including an internal data model and algorithms dedicated to the treatment of uncertainties. The main function of that library is to give an application all the functionalities needed to treat uncertainties in

studies. Targeted users are all engineers who want to introduce the probabilistic dimension in their so far deterministic studies.

- an independent application with a graphical user interface. This application was built to become the work environment for the specialist of the treatment of uncertainties. Targeted users are once again the industrial practitioners: those who identify the treatment of uncertainties as a full task, which can be spread to multiple engineering domains.

- a python module with high level operators in the probabilistic and statistical field. The interest of this language is to be both a powerful scientific language (capable of using C++ libraries), and a user friendly interpreted language like Matlab's one. This python module was designed to make the development of prototypes easier (in order to test new algorithms and methods for example), to become an easy-to-use support for educational works. This module intends to become a natural environment capable of integrating new developments within the field of uncertainty and sensitivity analysis. The targeted users are here research centers and the academic community.

OpenTurns is composed of several modules. Our work is to interface the distribution module with XtreemOS.

2.10.1 Application Development and Porting

Since the 2.0 XtreemOS release, OpenTurns is available with the same features it has on a regular Linux box. Interested users can already try it to test their study, running separate execution one after another on the current XtreemOS node they are logged on.

We are now working on interfacing the distribution module of OpenTurns with XtreemOS through the XOSAGA interface or by scripting the submission of independent jobs directly to AEM.

2.10.2 How the Application can Contribute to the Exploitation of XtreemOS

Once interfaced, OpenTurns will transparently submit jobs to the XtreemOS grid and the XtreemOS scheduler should naturally exploit all resources available executing several runs in parallel.

2.10.3 How the Application can Contribute to the Dissemination of XtreemOS

XtreemOS provides a nice distributed environment to handle uncertainty studies driven by OpenTurns. Once an XtreemOS is correctly set-up, OpenTurns should install very easily and at once take advantage of all resources available.

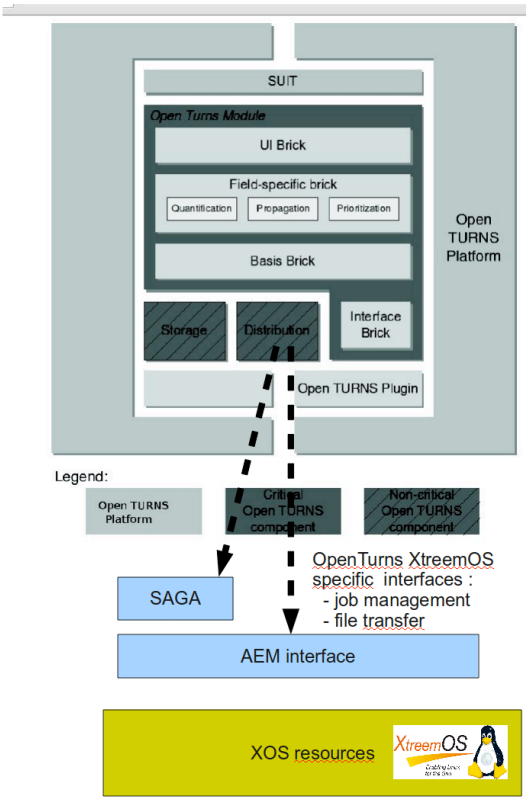


Figure 2.8: Interfacing OpenTurns with XtreemOS

Some users interested in uncertainties studies have no experience or skills to put the unused workstations into a coherent distributed test bed that OpenTurns will address. If the installation of XtreemOS becomes sufficiently easy and automatic, it might be a clever and affordable mean to install OpenTurns on these machines and run these studies as fast as possible.

2.11 Zephyr (EDF)

2.11.1 Application overview

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 and the MPI version of Zephyr.

Zephyr is packaged as a single program - sequential or MPI - taking a list of

parameters describing the problem to solve from an input text file. Parameters sets the precision of the discretization (number of point in X and Z), the precision of the numerical scheme (2nd or 4th order), the problem solved, the solver used (Conjuguate Gradient with different preconditioners with or without multigrid algorithms), and the frequency of result savings.

Zephyr produces result data both on the standard output stream and on demand in binary result files. Written in Fortran 90, it requires the presence of at least the "gfortran" compiler, and its wrapping into MPI.

To visualize the obtained result, another process drives a gnuplot session that regularly reads the current output result and displays it on on screen.

More information on the calculation schemes implemented in Zephyr can be found in [20], and on their parallel efficiency in [19].

2.11.2 Application Development and Porting

Zephyr was easily ported to numerous HPC machines. The only pre-requisite was the availability of MPI and a decent access to the filesystem seen by all the nodes running Zephyr.

Still, it had not yet been ported non-heterogeneous platform or to grid of computer due to the lack of common performant filesystems. XtreemOS architecture gave us this opportunity taking advantage of the shared filesystems XtreemFS.

During the project, Zephyr correctly compiled under sequential and MPI version. At this date, Zephyr has run fine in sequential.

On this code, the Checkpoint/restart XtreemOS feature has been tested and worked well (see Deliverable 4.2.6). We demonstrated that this mechanism was easy and natural to use and that its overhead in memory footprint and in time of calculation was low and grew linearly with the size of the problem solved.

Concerning the MPI version, it has been compiled successfully but we are waiting to access a platform to start testing it extensively.

This should be done by the final defense of the XtreemOS project.

If some testing time is left, we will check if the performances measured on XtreemOS are comparable with regular MPI use.

2.11.3 How the Application can Contribute to the Exploitation of XtreemOS

XtreemOS makes the execution of Zephyr possible on an heterogeneous grid of computer. Zephyr also benefits from the Checkpoint/restart feature with no modification of the code.

Zephyr is only an application used to benchmark hardware. Still its footprint matches the one of bigger code like Code_Saturne widely used at EDF R&D to perform Fluid Dynamics calculation. So any proof of concept made with Zephyr is easely transfered to Code_Saturne or any MPI intensive application.

After project end, eventual further experiments will be held internally on these targeted codes.

2.11.4 How the Application can Contribute to the Dissemination of XtreemOS

With this case, the Fortran 90 programming language availability and MPI environment under XtreemOS will be proven to be OK.

The porting of Zephyr on XtreemOS demonstrated that Checkpoint/Restarting of an existing sequential application was possible at virtually no effort.

If XtreemOS gains in influence and disseminates, EDF R&D could envisage to port Code_Saturne on this platform. That way, Code_Saturne users could transparently access the aggregated power of single workstations gathered together thanks to one single Grid Operating System.

2.12 SALOME (EDF, INRIA)

2.12.1 Application overview

SALOME (see Figure 2.9) is an open-source software application that provides a generic platform for Pre- and Post-Processing for numerical simulations. It is based on an open and flexible architecture made of reusable components. SALOME can be used as a standalone application for generation of CAD models, their preparation for numerical calculations and post-processing of the calculation results. SALOME can also be used as a platform for integration of external third-party numerical codes to produce a new application for the full life-cycle management of CAD models. SALOME allows users to:

- create, modify, import, export, repair, and clean CAD models;
- mesh CAD models, edit mesh, check mesh quality, import and export mesh;
- handle physical properties and quantities attached to geometrical items;
- perform computation using one or more external solvers (code-coupling);
- display computation results (scalar and vectorial);
- manage studies (create, save, reload).

The SALOME platform architecture (see Figure 2.10) is based on the model of distributed components built on CORBA as a distributed objects architecture. Two main levels can be distinguished:

Lower layer: embeds core functionalities of the kernel (communication between distributed modules), graphical user interface and management of the studies. These services are handled by the KERNEL and GUI components.

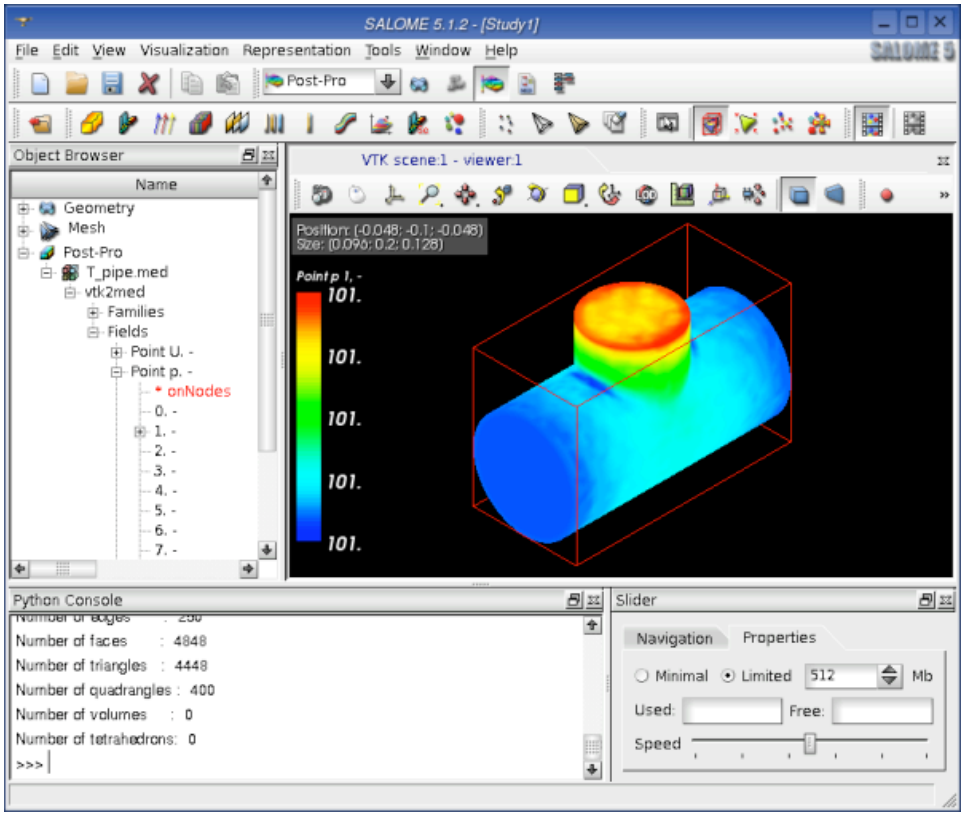


Figure 2.9: Screenshot of the graphical interface of SALOME (fluid mechanics and thermal study of a T pipe)

Modules layer: higher level components built on the services provided by the lower layer. Modules perform dedicated services that are needed to reach the general objective of SALOME. The main modules involved in this layer include GEOM, MESH, MED, SUPERV, POST-PRO and YACS. Furthermore, this layer can be enhanced by a number of dedicated end-user modules that can, for example, encapsulate a numerical solver behavior, simplify input of initial condition for a computation, etc.

2.12.2 Application Development and Porting

At the end of the XtreemOS project, porting SALOME on XtreemOS is still an on-going work.

SALOME should not be too complicated to port on XtreemOS thanks to its modular architecture. The KERNEL component contains the resource manager which will be specialized to use XtreemOS as a backend (SALOME resource manager interface will be implemented with XATI). SALOME programs, called

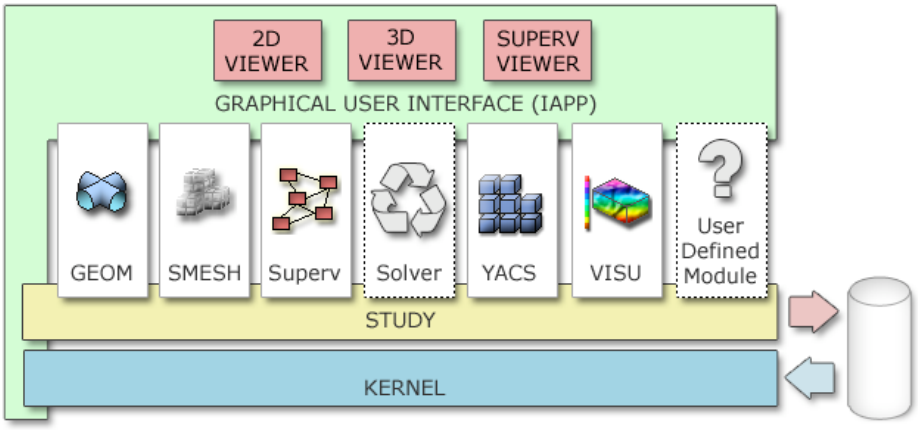


Figure 2.10: General architecture of the SALOME platform

schemas, are similar to workflows or dataflows of components. The execution supervisor wraps these components into containers which run on each resource. SALOME's containers will be implemented by XtreemOS jobs.

2.12.3 How the Application can Contribute to the Exploitation of XtreemOS

If most of the popular scientific applications, like SALOME, are ported to XtreemOS, engineers will more likely switch to XtreemOS to benefit from large scale grids.

Currently, SALOME's resource manager cannot deal with failures and is not scalable. People will choose to run SALOME on XtreemOS because, with XtreemOS, SALOME will be fault-tolerant, scalable, etc.

SALOME scientific applications are also quite resource demanding and complex software, so it will be a good test for XtreemOS.

2.12.4 How the Application can Contribute to the Dissemination of XtreemOS

SALOME is an open-source project and the XtreemOS backend will be included in the SALOME platform as open-source.

2.13 SAP NetWeaver Search and Classification (SAP)

2.13.1 Application overview

SAP NetWeaver Search and Classification provides search and classification capabilities using functionality of TREX application. TREX application runs over

distribute landscape. Figure 2.12 shows the architecture of storage solution of TREX. The search engine has two main stages it its activities: index construction and search for documents. At index construction a large collection of documents is read from a file system and index is written and read from the file system in several phases. The file system is either NFS or distribute file system allowing an access to the files from all the nodes of the distributed landscape. Thus the indexing stage imposes heavy load to the underlying file system. At the search mode the index is partitioned into several chunks and loaded into the main memory of the distributed landscape and the search queries are answered from the main memory, almost without disk access.

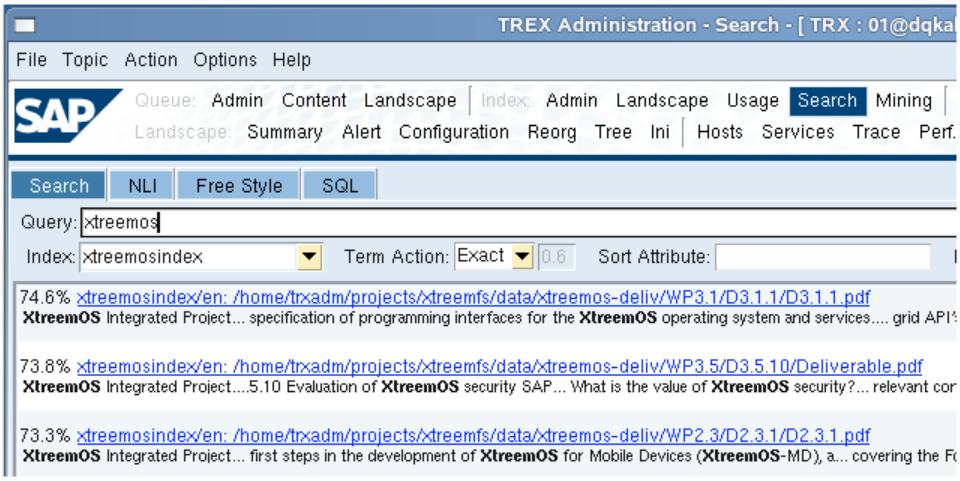


Figure 2.11: TREX - administration window

2.13.2 Application Development and Porting

At the beginning of the project TREX - SAP NetWeaver Search and Classification engine - was a product generally available to SAP customers. Since the beginning of the project TREX underwent development, and new features were added, i.e. enabling managing delta index in the main memory. However those changes do not significantly change the IO workload at the index construction stage. For this reason working with the newer version of TREX would not significantly change TREX setup over XtreemOS.

Our major focus of working with TREX at XtreemOS was testing functional and performance features of XtreemFS - distributed file system developed under XtreemOS project. Since TREX assumes POSIX semantics at the underlying file system and XtreemFS provides this semantics, no porting or additional development was needed to run TREX over XtreemFS. However few scripts were developed mainly to set up XtreemFS at our local SAP cluster.

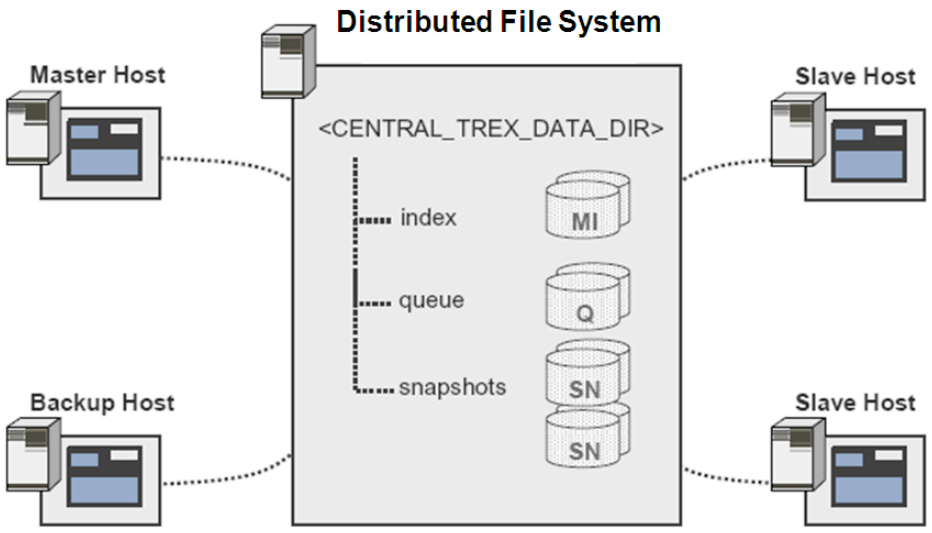


Figure 2.12: TREX - Enterprise Search Engine

2.13.3 How the Application can Contribute to the Exploitation of XtreemOS

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 summarize, striping capabilities of XtreemFS open the opportunity of scaling out, as opposed to scaling up with most of the commercial filer solutions.

TREX team looks for ways to exploit the benefits of a commodity-based distributed file system to provide the users with the performance that is currently available under commercial file systems. Under such a setting the TCO of NetWeaver Search and Classification product can be lowered without jeopardizing the performance guarantees for its users. XtreemFS is one of the candidate file systems that can be used to support TREX IO requirements.

2.13.4 How the Application can Contribute to the Dissemination of XtreemOS

TREX benefits from the XtreemFS capability to read and write data to the file system in parallel due to the striping and replication capabilities of XtreemFS. The

replication capability of XtreemFS can also be used for high availability of TREX solutions.

2.14 SAP MaxDB Replayer (SAP)

2.14.1 Application overview

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 2.13. 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).

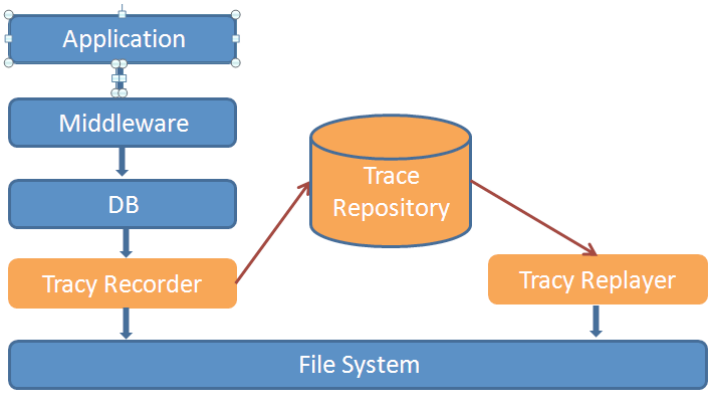


Figure 2.13: Enterprise Scenario - MaxDB Replay

2.14.2 Application Development and Porting

2.14.3 How the Application can Contribute to the Exploitation of XtreemOS

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. The file system 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. The replication of XtreemFS may 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.

MaxDB team looks for ways to exploit the benefits of a commodity-based distributed file system to provide the users with the performance that is currently achievable under commercial file systems. Under such a setting the TCO of applications that use MaxDB can be lowered without jeopardizing the performance guarantees for its users. XtreemFS is one of the candidate file systems that can be used to support MaxDB IO requirements.

2.14.4 How the Application can Contribute to the Dissemination of XtreemOS

MaxDB benefits from the XtreemFS capability to read and write data to the file system in parallel due to the striping and replication capabilities of XtreemFS. This feature of XtreemFS can help to achieve high throughput of XtreemFS without sacrificing the latency. The replication capability of XtreemFS can also be used for high availability of TREX solutions.

2.15 Rule-based System Management (SAP)

2.15.1 Application overview

Rule-Based System Management (RBSM) is an Automated System Management tool that provides a powerful administrator interface for rapid application and infrastructure deployment and control. Using a set of administrator-defined management rules and deployment templates, it enables administrators to automate many of the routine tasks they perform, making them more predictable and less prone to error . The main interface can be seen in Figure 2.14. Information on the set of resources that are under the control of the administrator is shown to the left of the interface window, allowing the administrator to monitor the current state of the system landscape. Operations such as software or virtual machine deployment, node reconfiguration and system migration can be accessed from a secondary tab, and the administrator provides only the information required, allowing the system to perform the task itself.

The application consists of a number of discrete components to form a control loop, which can be seen in Figure 2.15 . In order to gather information on the current landscape state, a series of probes are deployed on each node, collecting data on CPU load, memory usage, and other important metrics. These probes communicate with a monitoring component, which gathers and aggregates data from

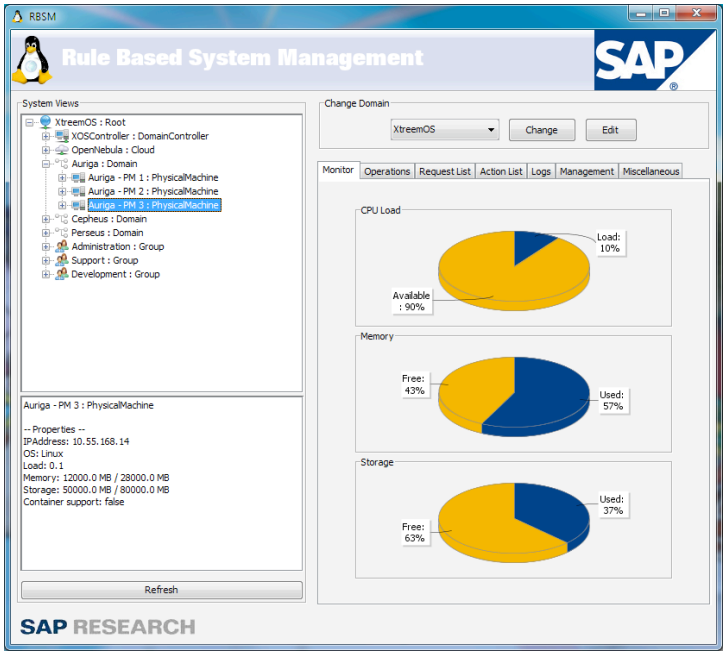


Figure 2.14: RBSM Main Administrator interface

across the landscape, creating a live system model, which is used to inform the administrator and provide input for the rule-engine. Through the main interface, the administrator can define a request, such as installing a piece of software. The rule-engine combines the request, with the rules for deployment, which involves selecting the most suitable resource from a range as defined by the adminstrator. The output of the rules is a set of Actions, which are interpreted by the deployment component and used to create a script from a library of templates. A connection is created between this component and a constantly running daemon on the target node and the control actions are transmitted and performed on the node, thus completing the cycle.

2.15.2 Application Development and Porting

Development of the RBSM application began in the 2nd year of the XtreemOS project and was geared towards taking advantage of the management features of XtreemOS, as well as enabling rapid deployment of an XtreemOS grid into a cloud computing environment. As it was built in Java and geared towards portability, there was no specific work required to enable it to run natively on an XtreemOS system.

The main components of the application have been completed, and further development will focus on expanding the range of control actions available in the management system.

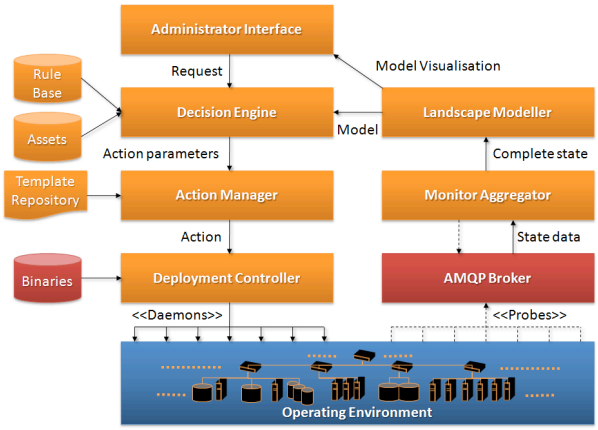


Figure 2.15: RBSM Architecture

2.15.3 How the Application can Contribute to the Exploitation of XtreemOS

RBSM takes advantage of a number of featuers of XtreemOS that aid in system management and control of resource deployment. The main features that are used are the built-in application checkpoint and migration functionality, as well as the OpenVZ platform for virtualised containers.

Internally, the application is used to deploy the XtreemOS small testbed environment for the purposes of testing and evaluation of the use of XtreemOS as part of a cloud computing infrastructure. Externally, it has been used as the basis of a number of peer-reviewed papers and in demonstration sessions, including the Euro-Par 2009 conference. The application will allow XtreemOS to continue to be used as a testing platform for grid applications on our internal cloud computing infrastructure beyond the end of the project.

2.15.4 How the Application can Contribute to the Dissemination of XtreemOS

The RBSM application has so far been shown at the following events:

- Internal presentations
- XtreemOS Summer School 2009
- Euro-Par 2009
- XtreemOS Summer School 2010

Work related to the RBSM application was described and presented in a paper accepted at the DEPEND 2010 Conference called “Classification and Impact

Analysis of Faults in Automated System Management". The paper acknowledged the input gained from the XtreemOS project.

2.16 BioLinux Application mpiBLAST (STFC)

2.16.1 Application overview

The NERC Bio-Linux distribution is a collection of over 500 Open Source packages for the analysis of biological data built on top of Ubuntu Linux. The latest release of the software, version 6.0, is based on the current release of the Ubuntu operating system. See http://nebc.nerc.ac.uk/tools/bio-linux for more details. The applications include:

- Comparison of sequences of proteins and nucleotides searching for statistically significant matches, e.g. BLAST and Blastx.
- Statistical analysis of many biological processes and data.
- Simulation of biological systems, e.g. Gromacs.
- Work flow management using tools such as Taverna

It is not possible to implement all these packages within XtreemOS since it would take too long. Debian packages are available for most of the tools and if a Debian port of XtreemOS is made then most Biolinux tools could be included directly.

Of most interest to XtreemOS are high performance applications requiring multiple processors and a distributed file system. To this end we have focused on the Blast series of programs which are used to search for alignments between biological structures. The Blast site,
http://blast.ncbi.nlm.nih.gov/, describes the purpose of these tools. Databases are available with increasing size and complexity as new research provides more data. These are freely available online via web interfaces and as web services which can be programmatically accessed, e.g. via a perl script.

Scientific task

Following discussions with Dawn Field from the Molecular Evolution and Bioinformatics Group in Wallingford, it was decided to define a typical analysis task for Blast, which would be of interest to scientists. The computational problem is of modest size to allow for testing on the XtreemOS permanent test bed.

The task selected is the Blast analysis of all the fully sequenced Baculovirus genomes in terms of their protein lists. Baculoviruses are a class of viruses that attack certain insects. They have many potential uses including as biopesticides and in production of human vaccines.

To date 47 complete baculovirus genomes have been determined and included in the NCBI genome database. These can be downloaded to give the protein sequences in the standard FASTA format. Blast or mpiBlast can then be used for analysis of similarities in the structures. A database of baculovirus proteins was built using mpiformatdb. This was then used in an all-against-all Blast run. This compares every protein in the database against each other and produces a list of those with the greatest similarity.

Architectural overview

For efficient parallel execution mpiBLAST was selected. The package makes uses of special techniques including:

- Database segmentation. Large databases can exceed the memory of a single processor and it is useful to split these across a number of processors, so that all the data can reside in memory.
- Dynamic search partitioning. Typically a Blast search will contain many different search terms which can be processed independently. mpiBLAST efficiently partitions the available workload.
- Parallel I/O for output. Blast searches often yield a large amount of output and parallel I/O can be used to optimize this.

A Blast run returns a list of matches found between sequences. A simple visualization of the matching sections of two proteins is shown in Figure 2.16, highlighting areas of alignment.

2.16.2 Application Development and Porting

mpiBLAST is a well established tool for parallel Blast investigations. It has been tested on a wide range of Linux systems and shows good scalability. Building the package under XtreemOS requires a C++ compiler and an MPI library.

The MPI library that has been integrated into XtreemOS is based on the ANL distribution of mpich-1.2.7. This package has some advantages over other MPI implementations, e.g. it is open source and has good support for mixed architecture grids, which are possible within XtreemOS.

Running tests with mpich-1.2.7 on the XtreemOS 2.1 test bed showed some problems. In particular there was a conflict with running programs due to a back ported fix from XtreemOS 3.0 that prevented MPI jobs running in the test bed. This bug has been fixed, but hindered evaluation of this MPI library.

As a result of these difficulties, MPICH-2 from ANL was also installed on the 2.1 test bed. mpiBLAST has been successfully built against both versions of MPI. In order to run mpiBLAST linked against MPICH2 on XtreemOS, a simple shell wrapper was developed. It sets up the necessary daemons before the job starts and closes them down after it is finished. This is not as flexible as the integrated

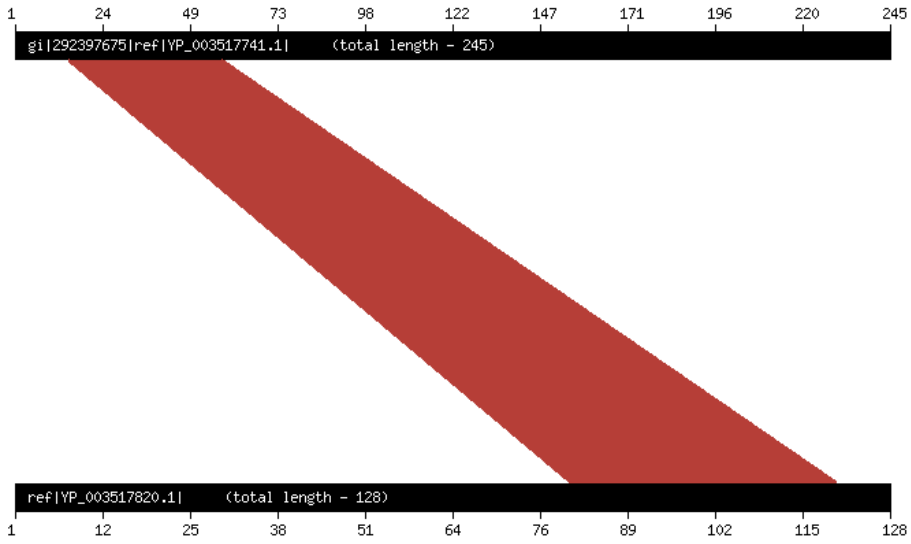


Figure 2.16: A result from mpiBlast on XtreemOS showing areas of two proteins that are structurally similar

XtreemOS MPI implementation in that it can only run on a single cluster of processors, rather than across all nodes of the Grid.

2.16.3 How the Application can Contribute to the Exploitation of XtreemOS

Within computational biology the BLAST packages are very widely used. The amount of genome data available is rapidly increasing and fast sequence searching is vital to exploit this resource. An XtreemOS VO can easily be constructed from idle processing resources within an institution, or across several institutions, which can then run mpiBLAST jobs to efficiently search through large databases.

Having shown that mpiBLAST can easily be run on XtreemOS will make the grid operating system much more attractive to computational biologists who need access to large databases, but do not have dedicated HPC resources. Users at the Molecular Evolution and Bioinformatics Group at Wallingford are interested in future collaborations that could use XtreemOS and mpiBLAST.

2.16.4 How the Application can Contribute to the Dissemination of XtreemOS

Collaboration with the Molecular Evolution and Bioinformatics Group on analysis of genome data should lead to a scientific publication of this work and the use of XtreemOS. The appropriate forum for this has yet to be determined.

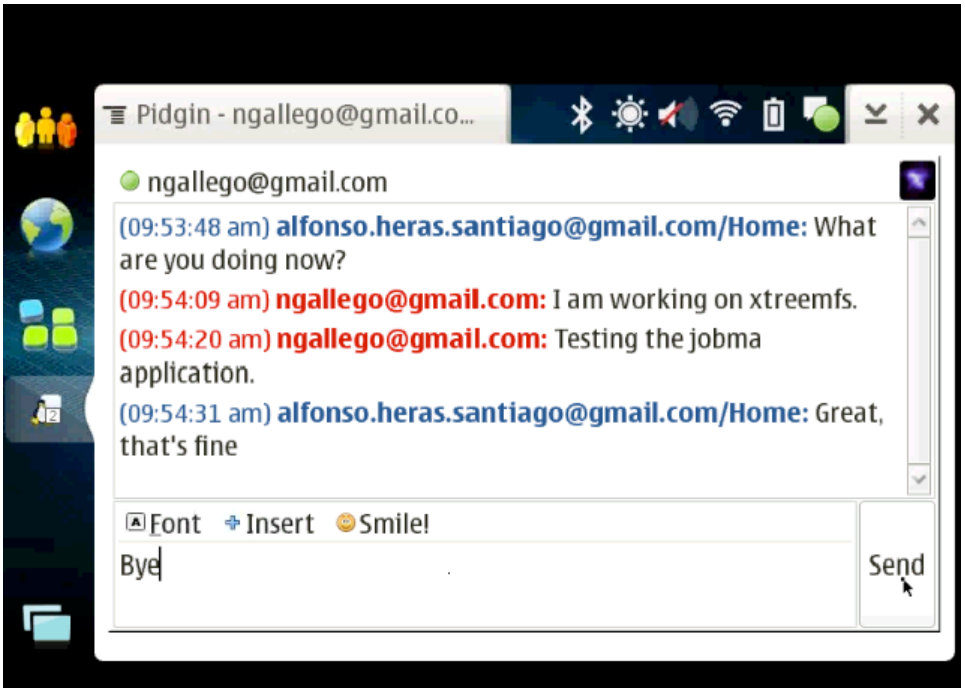


Figure 2.17: IMA application on a N800

2.17 Instant Messaging Application (TID)

2.17.1 Application overview

IMA is an instant messaging application oriented to facilitate the typical user tasks related to XtreemOS such as: registration in a VO, storing the configuration files and logs of the conversations in the Grid (XtreemFS). Advanced features like file sharing and the possibility of including every member of the VO as contact are also possible. It is available for mobile devices since XtreemOS 2.1 version.

IMA relies on the full architecture of XtreemOS-MD. IMA is a direct porting of the well-known Pidgin application [34] (a universal chat client) to XtremOS-MD. No special modifications have been done to the messaging protocols and/or user interfaces. However, some small modifications were done in order to integrate VO, security and XtreemFS services within the application functionality for mobile devices.

Figure 2.17 shows the IMA application on a N800 where a typical instant messaging conversation is shown. Figure 2.18 shows how the conversations logged and available at XtreemFS under the user's volume.

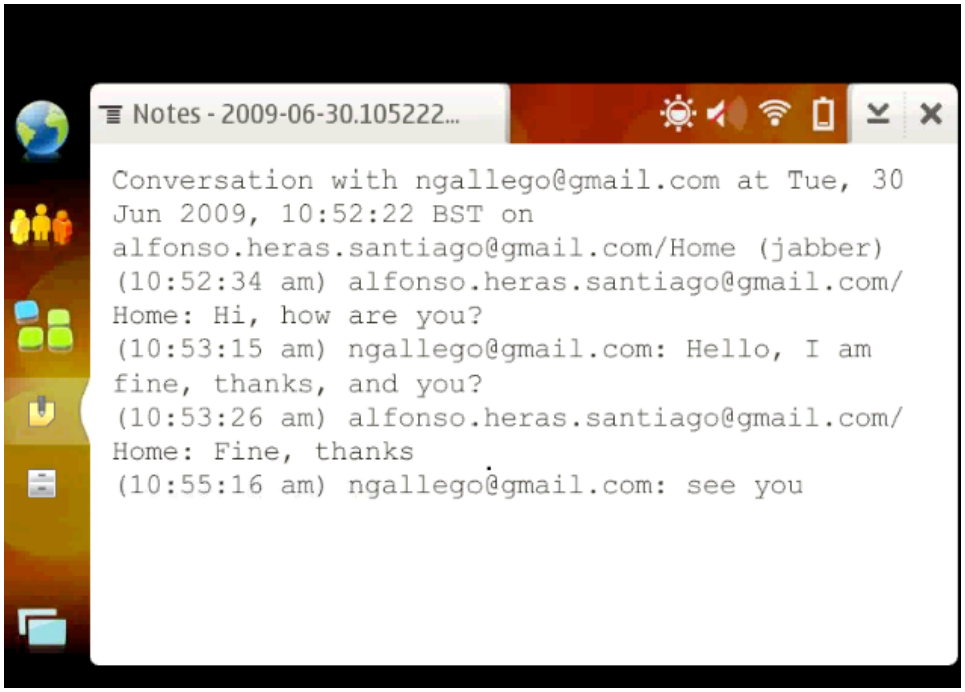


Figure 2.18: IMA logs user converstations into XtreemFS mobile device user's volume

2.17.2 Application Development and Porting

IMA application is fully based on the well-know Linux Pidgin app. At the beginning of the project Pidgin was only available for Desktop computers but not for mobile devices. Later on, community ported it to Nokia N800 and recently to Maemo platforms. Since the first porting to Nokia, we started working on modifying the code so as to make it compatible with the XtreemOS-MD API's. The final IMA version is available for Nokia N900 and Ubuntu Netbooks (direct porting of XtreemOS-MD to Ubuntu).

2.17.3 How the Application can Contribute to the Exploitation of XtreemOS

IMA is a good example of how an existing application is directly improved by XtreemOS features, in this case, XtreemFS, VO and security. This application is useful for XtreemOS Grid developers interested on learning how an existing application can be extended with XtreemOS Grid features from a Mobile Device, that being the main exploitation target. The application is distributed as part of XtreemOS, hence it is possible to be used by any user who has installed and deployed XtreemOS core elements plus some supported mobile devices. IMA and XtreemOS-MD software is licenced as GPL and porting to other mobile device

platforms by the community should be encouraged and promoted.

2.17.4 How the Application can Contribute to the Dissemination of XtreemOS

IMA shows how an existing Linux-based messaging application like Pidgin may be easily ported to Linux-based mobile devices under XtreemOS. In particular, it shows the integration with security and XtreemFS. The application is available in the XtreemOS ISOs.

During 2010, several demonstrations were done at Eurosys 2010. In addition the paper: - "XtreemOS-MD: Grid Computing from Mobile Devices", Alvaro Martínez, Santiago Prieto, Noé Gallego, Ramon Nou, Jacobo Giralt, Toni Cortes, Telefonica I+D and Barcelona Supercomputing Center - was published and presented at Mobilware 2010, Chicago.

2.18 Job Management Application (TID)

2.18.1 Application overview

JobMA is a graphical job manager application, specifically designed to be used from mobile devices, where the usual lack of a physical keyboard and, moreover, the different philosophy of the usual applications, makes it recommendable not to use only the `xsub` command-line tool for job management.

JobMA relies on the full architecture of XtreemOS-MD. The user interface is based on the GTK library and provides an intuitive interface to use the main features offered by the Application Execution Manager (AEM), such as job creation and execution, stop/resume/cancel of running jobs, job monitoring, etc. Once the job is created (or loaded from a JSDL file), the user can run it in the Grid. To do this, the user will select the job clicking on it and then will click on the "Run" option under the "Action" tab in the top menu.

As an example of operation, JobMA offers two different ways of creating jobs. The first method is performed by defining a job from scratch with basic parameters as shown in Figure 2.19.

The second method is based on loading a more specific JSDL file with a full set of parameters. This method is shown in Figure 2.20

In both cases, when the job is loaded, it may be run to the Grid from JobMa as shown in Figure 2.21. A list of jobs is available and from this window any job may be suspended, canceled or restarted. The status of each Job is shown in the main window list and more details are provided by clicking on the job. For more information please visit the XtreemOS User Guide.

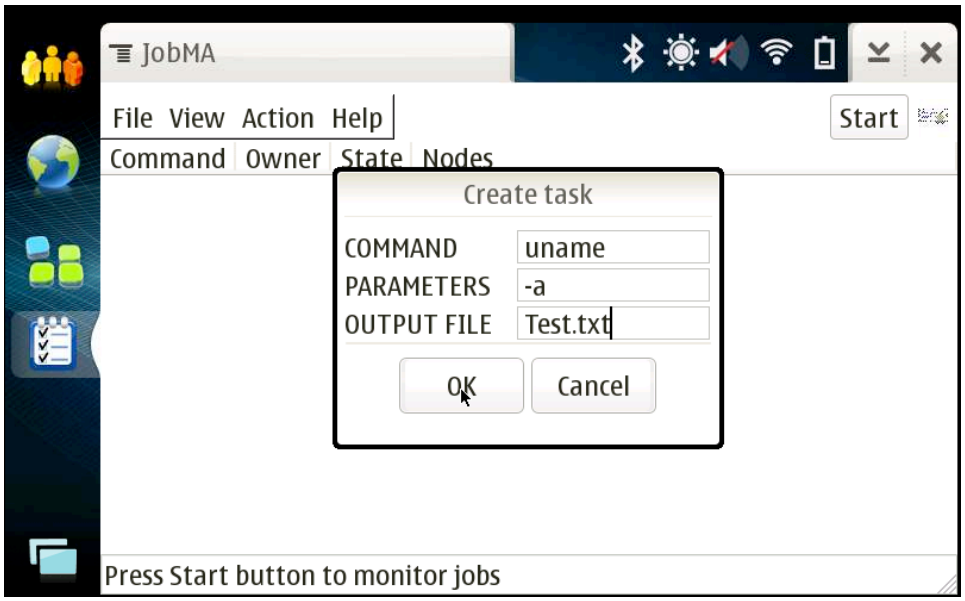


Figure 2.19: Defining a new job with Jobma in a Nokia N800 terminal

Figure 2.20: Loading a JSDL to run a job in Jobma with a Nokia N900 terminal

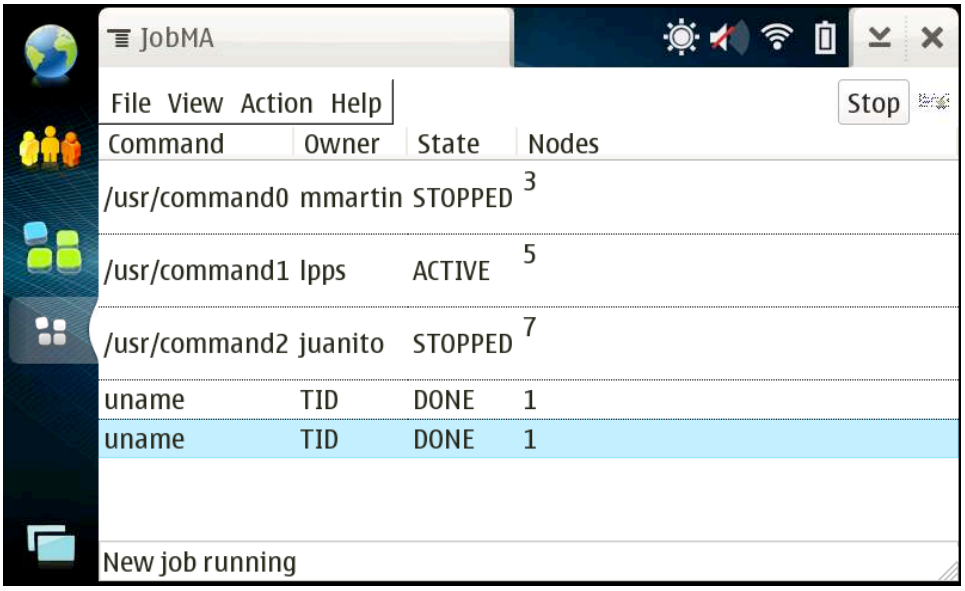


Figure 2.21: Running jobs with JobMa

2.18.2 Application Development and Porting

The JobMa reference application has been developed from scratch specially for mobile devices. It relies on the libraries and API's of XtreemOS-MD stack. The application has been developed initially for Nokia N800 and later ported to Nokia N900, OpenMoko and Ubuntu for Netbooks. The only modifications needed on each platform are done in the user interface as there are some differences in user interface development framework's on each platform.

2.18.3 How the Application can Contribute to the Exploitation of XtreemOS

JobMa application is a good example of application for mobile devices that shows the main functionalities of AEM in XtreemOS from the user's side. This application is useful for XtreemOS Grid users interested on managing jobs from the simplificity of mobile device interface, as such, that is the main exploitation target. The application is distributed as part of XtreemOS, hence it is possible to be used by any user who has installed and deployed XtreemOS core elements plus some supported mobile devices. JobMa and XtreemOS-MD software is licence as GPL and porting to other mobile device platforms by the community should be encouraged and promoted.

2.18.4 How the Application can Contribute to the Dissemination of XtreemOS

As explained before, JobMa shows the main functionalities of Application Execution and Management of XtreemOS from the user's side. The application is available in the XtreemOS ISOs.

During 2010, several demonstrations were done at Eurosys 2010. In addition the paper: - "XtreemOS-MD: Grid Computing from Mobile Devices", Alvaro Martínez, Santiago Prieto, Noé Gallego, Ramon Nou, Jacobo Giralt, Toni Cortes, Telefonica I+D and Barcelona Supercomputing Center - was published and presented at Mobilware 2010, Chicago.

2.19 Wissenheim (UDUS)

2.19.1 Application overview

Wissenheim is a distributed interactive 3D virtual world for edutainment and entertainment. In contrast to the present massively multiplayer virtual environments (MMVE) which are using a message passing approach Wissenheim employs a data centric approach by sharing the dynamic game state via an in-memory data grid. Every participating node is accessing the game state directly, synchronized by a variety of different consistency models, like e.g. transactional consistency. In contrast to the dynamic game state, media data like meshes, textures or sounds are loaded from files located in either a local or distributed file system.

Figure 2.22: Wissenheim screenshot.

2.19.2 Application Development and Porting

Prior to the XtreemOS project the Wissenheim application was available for the experimental distributed operating system Plurix, developed at the University of Ulm. Within the XtreemOS project Wissenheim was ported to XtreemOS/Linux which involved not only the replacement of the previously used data-centric features provided by Plurix with Object Sharing Service of XtreemOS, but also porting of the graphics and threading modules to the X-Window and POSIXruntime. Additionally new content and demonstration scenes for XtreemOS have been added for demonstration and experimentation purposes. While under Plurix only local area network had been used, extra efforts had been made to handle the wide area nature of XtreemOS by introducing e.g. latency hiding mechanism within the application. In addition to the Object Sharing Service, Wissenheim uses XtreemFS to share its persistent media data in an efficient and consistent way among the participating nodes.

2.19.3 How the Application can Contribute to the Exploitation of XtreemOS

As mentioned in Section 2.19.2 Wissenheim mainly benefits from the grid data management services provided by XtreemOS, namely XtreemFS and OSS. The Wissenheim/XtreemOS demo video and live demos are used to promote XtreemOS inside and outside the Heinrich-Heine University Duesseldorf.

2.19.4 How the Application can Contribute to the Dissemination of XtreemOS

Wissenheim has been used for various demonstrations like e.g. at the ICS'09 in Hamburg, various review meetings and the XtreemOS summer school in Oxford 2009. Additionally a couple of video captures have been integrated into the XtreemOS website. The main features demonstrated by Wissenheim are the Object Sharing Service and the XtreemFS distributed file system.

2.20 Cloud Computing (VUA, ZIB)

2.20.1 Application overview

Zmile is an on-line photo archive, where users are be able to upload and categorize their photos, as well as browsing and searching images which are available to the public. It can be found under `www.zmile.eu`

Zmile consists of modern state of the art web-development software:

- The server-side is written in Java.
- It runs inside a Jetty application sever.

Figure 2.23: The Zmile web-site

- It uses Spring as servlet-framework.
- Hibernate is used to store user data in an PostgreSQL database.
- The website itself uses AJAX and is programmed with the Google-web toolkit.

Figure 2.24 shows the architecture of Zmile. As one can see it's nothing to fancy, rather a standard architecture of a state of the art web application.

2.20.2 Application Development and Porting

Zmile was developed from scratch for the XtreemOS project to demonstrate the possibility of the OS as a host for modern web-applications.

The main features used from XtreemOS, are the file-system and the XOSAGA-API to submit jobs. Thanks to the POSIX conformity of XtreemFS the development is transparent, as if you would store files in normal directories. Additionally Since XtreemFS scales so well it is not necessary to implement some nifty storage pooling solutions on your own.

The current status of Zmile can be described as working, it uses all features of XtreemOS as planned, but on the GUI side there is still some work to do, it is almost stable but has some glitches which might confuse the users left. Overall, the application could be rated as beta version.

2.20.3 How the Application can Contribute to the Exploitation of XtreemOS

The main benefit from using XtreemOS was the scalable storage solution, if the storage for the pictures runs short, one can add a new XtreemOS machine which

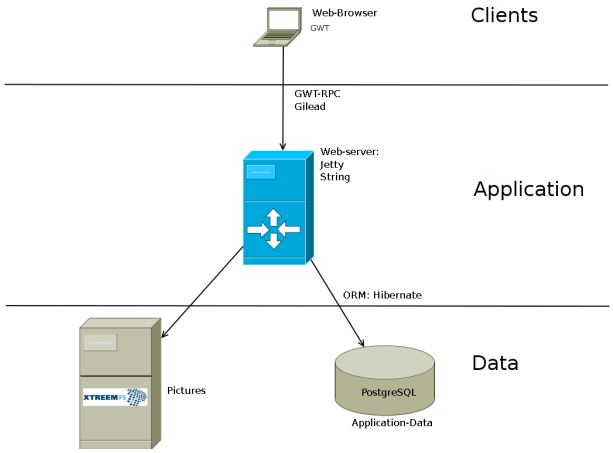


Figure 2.24: The Zmile architecture.

provides an OSD and the storage will transparently be extended.

The application currently runs on servers inside ZIB but is accessible like any website, everybody can register and use Zmile to manage pictures.

2.20.4 How the Application can Contribute to the Dissemination of XtreemOS

The purpose of Zmile was to show the usability of XtreemOS to the public, the web-site is available and freely usable, it also has a section explaining the interested user how XtreemOS was used to create the web-site and which benefit one get hosting the own application on an XtreemOS server. So we hope that it will attract some developers attention from beyond the project group.

As for now, it is planed that Zmile will serve as demonstrator on the final project review.

2.21 Galeb (XLAB)

2.21.1 Application overview

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.

The command-line version of Galeb is the one that has been ported to and ships with XtreemOS. The user supplies input data in the text file and, optionally, genetic

[1] http://lancet.mit.edu/ga/, the GAlib genetic algorithm package, written by Matthew Wall at the Massachusetts Institute of Technology.

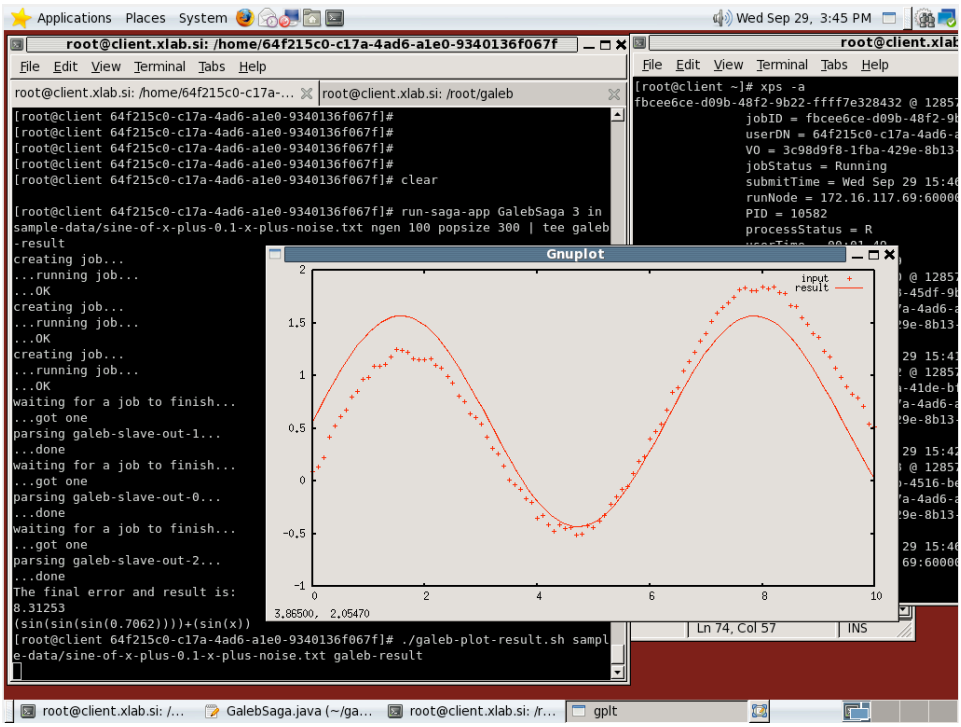


Figure 2.25: A successful run of Galeb on XtreemOS and the result in gnuplot

algorithm parameters such as number of generations, mutation probability etc. The output is a well fitting analytical function and its mean squared error. The result can be visualized with gnuplot or any other visualization tool. The screenshot of a successful run of Galeb on XtreemOS and the result in gnuplot are shown in Figure 2.25.

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.

2.21.2 Application Development and Porting

Galeb had been ported to the Globus Toolkit before the start of the project. Then, withing WP4.2, it was ported multiple times to XtreemOS.

1. The earliest port used `xos-ssh` because it was the only possible way to submit jobs at the time. XtreemFS was not yet functional at the time so `scp` was used instead. Therefore, this cannot be considered a full port.

2. The next port was a parallelization for multi-CPU machines using multiple processes communicating through System V IPC message queues. Although this port uses no XtreemOS-specific capabilities it is necessary to take advantage of LinuxSSI clusters.

3. Finally, the availability of XOSAGA allowed the full port of Galeb, in which the application lets XtreemOS handle resource selection, job submission etc. This version could also be easily ported to any other SAGA-enabled middleware. The only necessary modification would be to add explicit file staging because the XOSAGA version relies on XtreemOS for providing the capability to access input and output files without copying to a local filesystem.

2.21.3 How the Application can Contribute to the Exploitation of XtreemOS

Galeb had been developed as part of financial modelling research that XLAB did for an external client. The latter terminated the research multiple years ago and XLAB decided not to continue developing Galeb. It therefore cannot contribute to exploitation of XtreemOS.

2.21.4 How the Application can Contribute to the Dissemination of XtreemOS

Even though, as said above, the development of Galeb stopped, it is in its current version an ideal application to demonstrate certain technical features of XtreemOS. Furthermore, it is one of the applications in XtreemOS that have been previously adapted to run on Globus Toolkit 4.0 and have been ported to XtreemOS in WP4.2 [26]. As such it has been used for comparison of these two Grid paradigms from the application developer's point of view [10].

Each run of Galeb uses the XtreemOS grid as an elastic computing infrastructure, i.e. it lets XtreemOS select resources, run Galeb slave jobs on it and waits for them to finish. Each run also uses XtreemFS to access input and output files on resource nodes. In fact it was the first WP4.2 application that successfully submitted jobs in XtreemOS, as demonstrated at the general technical meeting in Pisa in June 2007.

Furthermore, the relative simplicity of the application makes it ideal to test and demonstrate checkpointing and restart as well as migration. In particular the tests with Galeb have revealed multiple migration and checkpoint/restart bugs in LinuxSSI.

Chapter 3

Evaluation of Installation and Configuration

Since the first XtreemOS release, WP4.2 carried out a long-term survey to record the end-user satisfaction with the project's software releases. The goal is to monitor and to track the evolution of software and packaging developments throughout project execution. The degree of end-user satisfaction with the various XtreemOS install CDs along with comments and recommendations have been gathered using online questionnaires. The results of the statistical evaluation have been reported to project management in order to provide support to software development and project planning. In the following, the survey setup and the statistical results are presented.

3.1 Survey Setup

The survey setup is identical with the one applied in deliverable D4.2.6, except for the two additional releases added to the survey. 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: The second public release of XtreemOS.
- v2.1.2: Last improved public release before v3.0
- v3.0 beta 2: Latest internal beta release at the time of submitting this deliverable

3.2 Survey Results

A total of 20 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 in-house 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

As an average over all categories, Figure 3.1 shows the end-user satisfaction for all XtreemOS releases examined. The results increase monotonously from a level of 2.78 for XtreemOS 1.0 to 4.3 for XtreemOS 3.0 beta 2. The improvement from v1.0 to v2.0 can mainly be attributed to facilitated installation and configuration tools, automatized setup, enhanced user interfaces (more graphical user interfaces) and advancement in the documentation. For the two latest releases, the increase can mainly be explained by further improvements regarding automated installation process and a more user-friendly documentation.

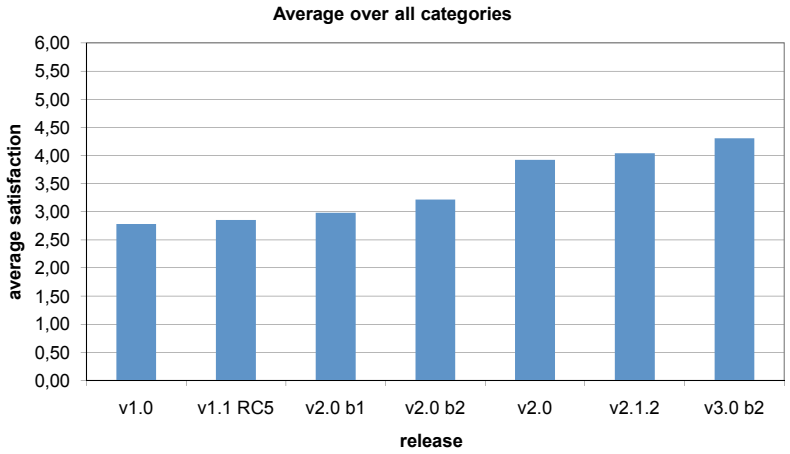


Figure 3.1: Overview of user satisfaction for all test items averaged across all categories.

Averaging the satisfaction across groups of test items, Figure 3.2 presents the results for the different survey categories (installation, configuration, basic usage and documentation). The set of test items has been divided into three groups: all releases (from v1.0 to v3.0 beta 2), the releases discussed in D4.2.6 (v1.0 to v2.0), and the two latest releases added to the deliverable at hand (v2.1.2 & v3.0 beta 2). In all three groups, category installation receives the highest ratings (4.34, 4.06,

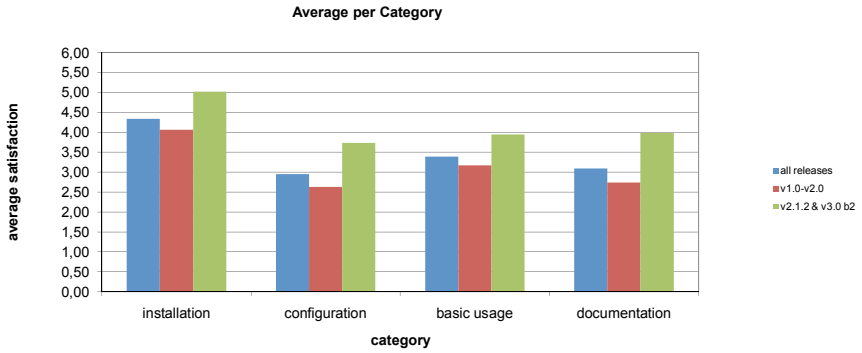


Figure 3.2: Overview of user satisfaction for all categories averaged across all test items.

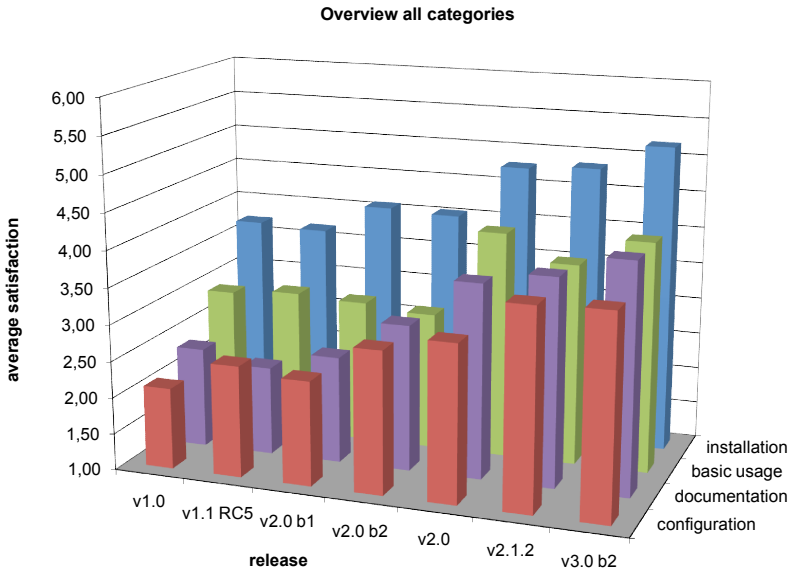


Figure 3.3: Overview of average user satisfaction for all categories and test items.

5.02) as people encountered least problems with the download and matured installation tools provided for the underlying Linux distribution. However, the configuration procedure was rated with the worst average values (2.95, 2.63, 3.74). For

the early releases 1.x, this effect can be accounted to many bugs and manual workarounds needed in particular. Recently introduced automated configuration scripts improved the results for v2.1.2 and v3.0 beta 2. 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.

An overview of the average satisfaction level for the different categories and test items is given in Figure 3.3. From version 1.0, all categories show clearly increasing trends. The most noticeable leap was made with the introduction of v2.0, which may originate from improvements in stability, ease of usage and also supported by a revised documentation.

3.2.2 Installation

The results for the installation procedure, broken per question, are depicted in Figure 3.4.

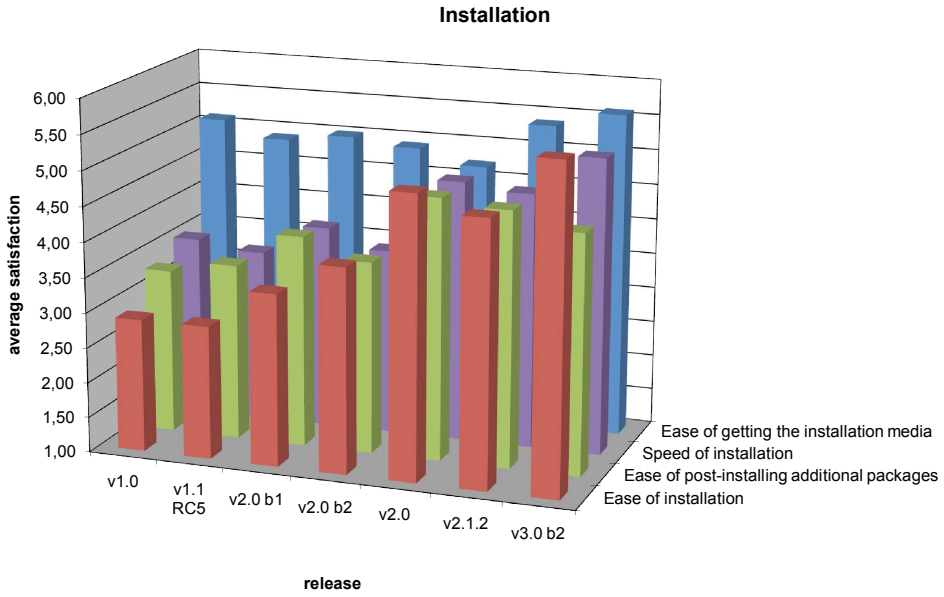


Figure 3.4: User satisfaction for category installation.

Consistently, the ease of getting the install media, namely CD-ISO, has received the highest average results. Download speed is very fast and the download servers work very reliably. Previously recommended improvements (see D4.2.6) are expected to be implemented with the new XtreemOS homepage such as facilitating the access to the install media on the XtreemOS home page. On the present homepage still too many mouse clicks are required to get to the download link. Ease and speed of installation have improved tremendously with the intro-

duction of the two major releases v2.0 and v3.0 beta 2. With respect to v2.0 the main advantage were the pre-configured node types. Meta-packages for different node type made the installation process much easier. In the latest releases, installation was commented to be as easy as any other standard Linux distribution except for minor problems with country-dependent keyboard selections and the need to re-install Erlang. Future improvements could involve reducing the number of installation options for inexperienced users. 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 and being further addressed approaching v3.0. Also post-installation of packages is easier, packages have been made available from the XtreemOS repositories. Previously it was reported that some packages still fail to install. Upon feedback to the developers, this problem seems to be resolved now.

3.2.3 Configuration

In Figure 3.5 the improvements in the configuration procedure are similar to those in the installation procedure.

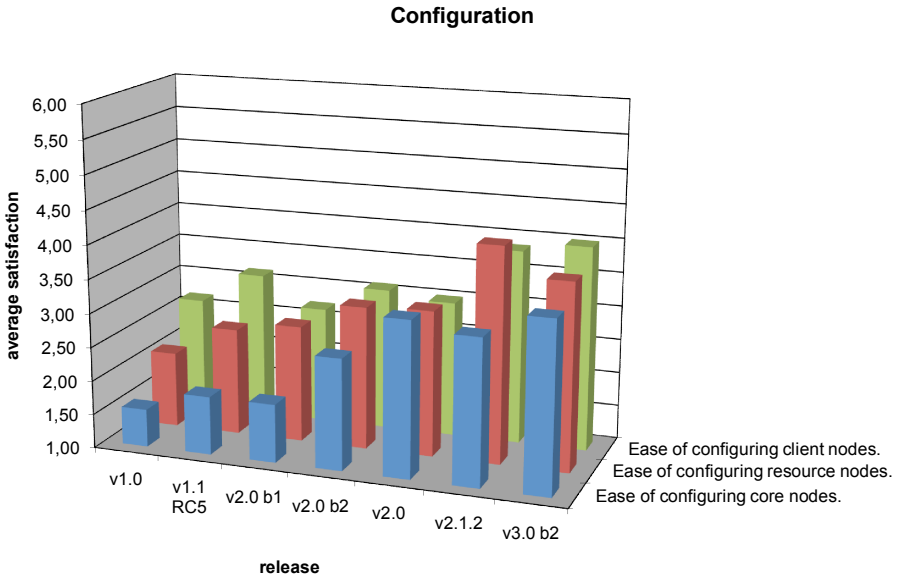


Figure 3.5: User satisfaction for category configuration.

Whereas from v1.0 to v2.0 the configuration of core nodes gained most, version 2.1.2 and v3.0 beta 2 gained most for the configuration of client and resource nodes. For core nodes and resource nodes, the configuration used to be a relatively tedious task in the early XtreemOS versions, whereas the configuration of a client is simpler and caused less problems reflected by the better rating. Thanks to the improvements in the *xosautoconfig* tool the configuration was reported to

work well and to make the job easy. Probably, the fact that the configuration of the core node is a bit harder cannot be avoided. Though the difference in user ratings regarding v3.0 beta 2 is rather marginal for the three node types: 3.50 (core), 3.75 (resource) and 4.00 (client). In the earlier evaluations, it was recommended to further reduce the number of manual steps and work-arounds, e.g. by introducing configuration scripts (either with default setup or with interactive parameter entry) and further self-explaining GUIs for the *xosautoconfig* tool. These recommendations have been addressed reflected by the better results for v3.0 beta 2. Still there is space for further improvements. For instance, it would be agreeable to the user could be with a step-by-step checklist for modifications to be made in various configuration files for core, resource and client nodes. Presumably, such a checklist would not be big because most of the default configurations need not be changed. Secondly, it would be beneficial if the configuration of a "pure and real" client could be facilitated which avoids job to be submitted to clients accidentally. Finally, the larger-scale configuration could profit from being able to automatically configure the IP addresses of a given set of nodes granted to a test bed. So a sort of automatized distributed installation could be envisaged.

3.2.4 Basic Usage

Figure 3.6 shows a remarkable increase in the rating of basic usage actions with the introduction of release 2.0.

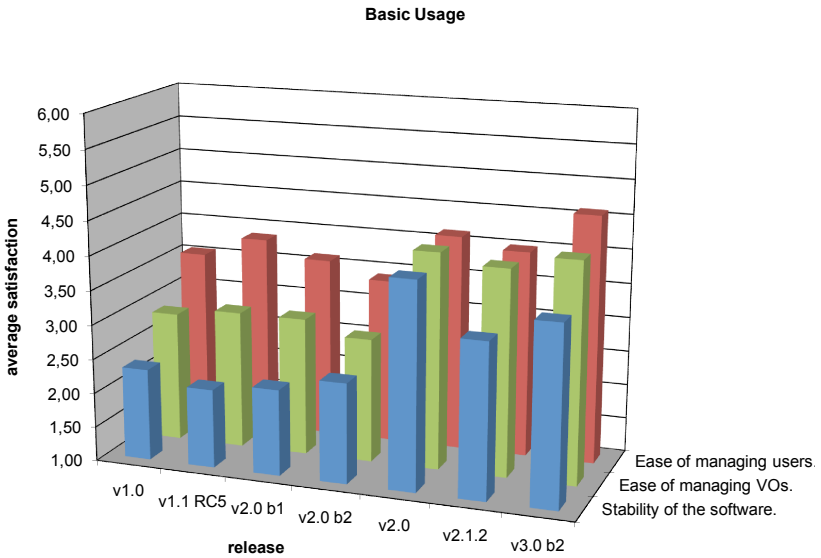


Figure 3.6: User satisfaction for category basic usage.

In particular, the stability of the software gained most. With the two most recent versions, the stability was reported to be a bit worse than before, random

XtreemFS and XOSD crashes may still occur. Issues are known to the developers and expected to be considered during the preparations for the major public release 3.0. Future XtreemOS releases might benefit from some sort of self diagnosing tool. It is assumed that the availability of VOLife has contributed to a facilitated VO management. In early versions, management of users and VOs were reported to require way too much manual work. Still user management could be improved. So it should not be required to download the private key to the server. Furthermore, downloading keys and certificates should be done in consistent ways, and the need to manually move them to .xos should be avoided.

3.2.5 Documentation

The clear upward trend in user satisfaction also applies to software documentation as visualized in Figure 3.7.

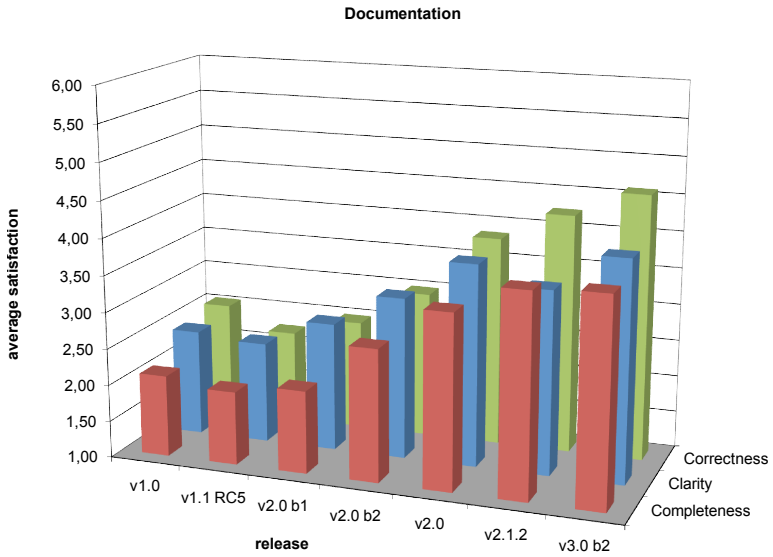


Figure 3.7: User satisfaction for category documentation.

Probably, issues with the documentation of early XtreemOS versions were one factor 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. Further progress in documentation quality was made with the introduction of versions 2.1.2 and v3.0 beta 2. The improved ratings might stem from the fact that now all documented steps match what is visible on the screen. The latest recommendations include the wish for a kind of quick install guide how to setup a Grid with just a few clicks. Also it would help to have a checklist of the steps to setup a testbed. The majority of suggestions for improvements target the documentation of XOSAGA. The documentation should show how XOSAGA works. In particular, it should elaborate on FileTransfer, WorkingDirectory and Cleanup. The XOSAGA documentation should clarify the abstraction of parallelism and also it would be helpful to visualize the delegation of jobs to various nodes. Apart from these changes, the documentation is reported to be fine and to make progress.

3.3 Summary

This survey gathered the experience of end users with seven XtreemOS releases from XtreemOS 1.0 (released end of 2008) until XtreemOS v3.0 beta 2 (tested September 2010). This allows for tracing the evolution of user satisfaction and for transferring feedback to developers in a continuous manner. The satisfaction was assessed with respect to the categories 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. The ratings improved almost 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.

Recommendations given in previous surveys were given to developers which tried to deal with them to a far extent. Also the comments and recommendations collected for the latest two releases covered by the survey were communicated. Minor problems with installations could be fixed like issue with country-dependent keyboard selections and the need to re-install Erlang. For inexperienced users it would also be nice to reduce the number of installation options. XtreemOS configuration cold be further facilitated by a step-by-step checklist for modifications to be made in various configuration files for core, resource and client nodes. Some assistance for configuring a “pure and real” client would also be appreciated. Also some support for larger-scale Grid setups could be offered to automatically configure the

IP addresses of a set of nodes, i.e. a sort of automatized distributed installation. Basic usage of XtreemOS could be improved by facilitating and further automating the management of keys and certificates. Finally, suggestions for improving the documentation were made like a quick installation guide, an installation checklist and elaborating on the documentation of XOSAGA.

After four years of research, XtreemOS end-users acknowledged the progress made in software installation, configuration, basic usage and documentation quality. This was expressed by ever increasing ratings in all categories. It is highly appreciated that so many recommendations and comments have been addressed which leveraged user experience.

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)
- LinuxSSI (Section 4.4)
- DIXI message bus (Section 4.5)
- XtreemOS API (Section 4.6)
- Resource Selection Service (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)

In this category, the experiments put emphasis on evaluating the performance, scalability, stability and correctness of the respective XtreemOS developments. As in deliverable D4.2.6 [10], this chapter collects the experimentation results from

the consortium presented in a common place. Apart from WP4.2, further work packages 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.

The following sections present the test documentation. The structure of the test documentation template is derived from "IEEE Standard for Software Test Documentation", IEEE 829-1998 [16]. 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 activities.

4.2 Evaluation of Node-level VO Support

This section covers the software for node-level VO support. The purpose, architecture and use cases are described in the XtreemOS deliverable D2.1.2 [9].

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 the latest release of XtreemOS available at the time of testing, updated with all the patches available.

4.2.1.3 Features to be Tested

This test plan includes testing the mapping of global user identities to local user identities and group identities.

4.2.1.4 Features not to be Tested

The VOlife VO administration tool will not be tested because its functionality has not changed since the last tests and no major bugs were found during the last test suite.

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 D4.2.6.

4.2.2 Test Unit 01: Correctness of account mapping

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

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 authorized user manages to log into the node, but is mapped incorrectly, e.g. as root,
- an authorized user is denied access.

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 a resource to both VO1 and VO2.

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

1. User1 logs into the resource node with `ssh-xos`.
2. User1 submits 'id', acting as member of VO1.
3. User1 runs a SAGA application that submits 2 jobs on his behalf.
4. User1 runs a SAGA application that submits 20 jobs on his behalf.

Wrap Up No wrap up is required.

4.2.2.3 Test Results

The test was executed by Marjan Šterk, XLAB, on 2010-07-04.

Execution Description The procedure was run on 2010-07-04 following the above Test Unit desciption.

Results When acting as a local non-privileged user on an XtreemOS client node posessing an XtreemOS user certificate, the user always successfully logged into the resource node and submitted into all jobs. He was never mapped incorrectly. However, there is a bug in the XtreemOS bug tracker: "ssh-xos can map you to a random user account", ID 0000247. We have not been able to reproduce this error.

When acting as local root on an XtreemOS client node posessing an XtreemOS user certificate, though, ssh-xos reverts to ssh in certain cases, preventing the use of XtreemOS credentials. We will investigate this further.

Anomalous Events As mentioned above, when logged in as root into a client node, ssh-xos reverts to ssh instead of using the XtreemOS credentials.

4.2.3 Test Summary Report

4.2.3.1 Summary of Tests and Results

The account mapping performs as expected in normal usage, except in cases when logged into the client node as root.

4.2.3.2 Conclusion and Directions for Future Work

We can conclude that the components for node-level VO support in XtreemOS Release 2.1 are adequate, with all major functional requirements met but certain bugs still to be fixed.

4.3 Evaluation of Checkpointing and Restart

4.3.1 Test Plan

4.3.1.1 Responsibilities

Partners and workpackages involved include BSC and XLAB as contributors from WP4.2, and UDUS as contributor from WP2.1/3.5.

4.3.1.2 Test Items

Test items include:

- Container-based checkpointing and restore mechanism of XtreemOS PC flavor
- XtreemOS grid checkpointing service (GCS)
- LinuxSSI checkpointing and restart

4.3.1.3 Features to be Tested

- With respect to Container-based checkpointing and restore, the tested features include the migration of virtual environments
- For XtreemOS GCS, the tested features include the CGS overhead, distributed checkpoint/restart and channel flushing
- For Linux SSI, checkpointing and restart the processes of a master-slave application was tested

4.3.1.4 Overall Approach

The three subsequent test units will cover the entire spectrum of checkpointing and restart stack of XtreemOS from kernel up to grid checkpointing of virtual environments and native applications considering the PC as well as the cluster flavor. The goal is to validate that the components work as expected and to measure the performance as well as the performance overhead. More details are given in the repective test specifications below.

4.3.2 Test Unit 01: Container-based checkpointing / restore mechanism with SPECweb

4.3.2.1 Responsibilities

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

4.3.2.2 Test Specification

Test Items

In the past deliverable D4.2.6 [10] we tested the container-based checkpointing/restore mechanism as stand-alone feature in a conventional linux environment. Now, this feature has already been integrated into XtreemOS and we aim to both test this integration and perform some performance measurements of it.

Features to be Tested

The main goal here is the testing of the XtreemOS feature "checkpointing and restore". Specifically, we want to evaluate the OpenVZ container-based checkpointing / restore feature that has been incorporated into the XtreemOS system.

OpenVZ [32] 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 given container can be rebooted independently and it has root access, users, IP addresses, memory, processes and applications, among others.

Additionally, 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

Concerning the application used, SPECweb2005 bechmark [3], we decided to use the Apache Tomcat (v5.5) [41] as the web server to be checkpointed and restored. Actually, this is the *System Under Test (SUT)* component of the aforesaid benchmark, and we have to checkpoint the Java Virtual Machine (JVM) in which this web server is encapsulated. In addition, it should be noted that this server has to be previously submitted, as an XtreemOS job, to an OpenVZ container.

We will consider the success of the feature if, at least, we are able to checkpoint and restart a container in which the Tomcat server has been submitted, i.e. it is deployed and running. In addition, we will use SPECweb2005 benchmark to validate that the checkpointing/restore operation is successfully performed. This means to check that the state of the web server after a restore operation is the same than before the checkpointing action. This fact is easy verifiable 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. 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 node during all the execution time of a given test. Nevertheless, this migration operation is based on the checkpointing and restore procedures tested here, so it is tested implicitly.

In a nutshell, we will present the time taken to submit the SUT component (web server) of SPECweb benchmark into an OpenVZ container, as well as the time needed to checkpoint and restore it.

Configuration The configuration needed to perform this evaluation is composed mainly by two components: the OpenVZ mechanism and the SPECweb2005 benchmark. On one hand, we have the XtreemOS support for different virtualization technologies, such as KVM and OpenVZ. It is based on libvirt [21] in order to handle this multiple types of virtual machines. On the other hand, we have configured the needed environment for running the benchmark. In particular, it is composed by the System Under Test (SUT), a Back-end database SIMulator (BeSIM), the clients required to perform the input load to the web server under test, and an special client (namely prime client) that controls the benchmark execution.

Set Up First of all, and due to contraints in the current release (3.0) of XtreemOS, we want to remark that we use the 2.1.1 version with the code needed for the integration of OpenVZ on XtreemOS. Once this simple integration has been made, the set up process must follows the following steps for configuring the OpenVZ support needed:

- Make sure that the "OpenVZ" modules for checkpoint and restore are in the kernel:
  - *modprobe vzcpt*
  - *modprobe vzrst*
- Properly configure the /etc/sysctl.conf file with the following lines:
  - net.ipv4.ip_forward=1
  - kernel.sysrq = 1
  - net.ipv4.icmp_echo_ignore_broadcasts=1

  - net.ipv4.conf.all.send_redirects = 0
  - net.ipv4.conf.default.rp_filter=1
  - net.ipv4.conf.all.rp_filter=1
  - net.ipv4.conf.default.proxy_arp=0
  - net.ipv4.conf.eth0.proxy_arp = 1
  - net.ipv4.conf.default.forwarding=1
  - net.ipv4.conf.default.send_redirects = 1
- Install a template (e.g. Debian) for OpenVZ containers, so jobs inside containers will run in the default Debian environment
  - Specify in the */etc/vz/vz.conf* file this aforesaid template as the one to be used when creating OpenVZ containers

Afterward, the tests are performed as described below:

- Specify that we want to submit the XtreemOS job to an OpenVZ container
  - Add <*Checkpointer*> *OpenVZ* <*/Checkpointer*> to the "JobDescription" field of the JSDL
  - The AEM reads this JSDL tag and performs the following actions:
    * Creates a new OpenVZ container
    * Sets up the network IP address of this container, as well as its hostname, DNS server address and root password
    * Start the container
    * Spawns the job
- Checkpoint (*vz chkpnt* <*container_id*>) and restore (*vz restore* <*container_id*>) the container
- Check that the web server is still accessible at the same container and that its state is the same

Start The actions necessary to begin the test are:

- Check that OpenVZ kernel, the application-level utilities (i.e. vzctl, vzprocps and vzquota), and the OpenVZ kernel modules (vzcpt and vzrst) have been successfully installed
- Configure the SPECweb2005 benchmark environment
- Prepare the appropriate JSDL to allow the job to be submitted into an OpenVZ container

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

Contingencies We have no estimation of any anomalous situation.

4.3.2.3 Test Results

This evaluation complements the one presented in D4.2.6 [10], where we showed the evaluation of the OpenVZ scalability and performance overhead. In fact, those results, obtained in a native linux environment, are the same as if using the OpenVZ in a XtreemOS system. Due to this fact, XtreemOS doesn't introduce any performance loss when performing OpenVZ operations.

In this section we present the analysis and evaluation of the OpenVZ capability of checkpointing / restoring a container in which a Tomcat (v5.5) web server has been deployed. Note that the tests were performed using a machine with the following hardware:

- Intel Core 2 CPU @ 2.33GHz
- 4GB of RAM memory

OpenVZ performance measurements In Table 4.1 we present the results regarding the time needed to: (1) submit a job to be executed natively, (2) submit a job into an OpenVZ container, and (3) to checkpoint/restore such a job. We repeated these operations ten times and we present the 95% confidence interval on the mean. Note that this job is a Tomcat web server encapsulated into a Java virtual machine with 1GB of memory.

Operation	Time (seconds)	95% interval confidence
XtreemOS native Job submission	23.29	$\pm$ 2.98
OpenVZ Job submission	129.41	$\pm$ 3.54
Container checkpointing / restore	6.45	$\pm$ 0.28

Table 4.1: OpenVZ performance measurements.

As you can see, there is a significant difference between the time taken for a native job submission and for the OpenVZ one. This is mainly due to, in the second case, the AEM, which is responsible for performing XtreemOS job submissions and has to create a new OpenVZ container where the job in question will be executed. Moreover, the time taken to checkpoint and restore a container with 1GB of memory is quite promising.

Conclusions The experimentation conducted shows us the success of the OpenVZ container-based checkpointing mechanism of XtreemOS. In fact, this is confirmed by the performance results obtained. Although the time taken to submit a job to an OpenVZ container is 5.5x greater than the one needed for native submissions,

the chance to execute them into containers seems to be very profitable for jobs that need isolation, for instance.

Implicitly to experimentation is the fact that we were able to submit a Tomcat web server in an OpenVZ container, and checkpoint/restore such a job. In this sense, XtreemOS provides transparency for submitting a job to an OpenVZ container, instead of executing it natively, which is very promising. In addition, through executions of SPECweb2005 benchmark we checked the success of these mechanisms.

Finally, these results presented herein complement the OpenVZ scalability tests and loss of performance presented in D4.2.6 [10]. According to all of them, we hope that its final integration will be a reality in the third release of XtreemOS.

4.3.3 Test Unit 02: Grid Checkpointing and Restart

4.3.3.1 Responsibilities

WP2.1, WP3.3, John Mehnert-Spahn (UDUS)

4.3.3.2 Test Specification

Test Items

Features to be Tested

Approach Refinements

4.3.3.3 Test Results

GCA overhead The diagrams in figure 4.1 show the times for a checkpoint operation with native BLCR, MTCP and SSI. The diagrams in figure 4.2 show the corresponding times of GCS-based checkpointing with BLCR, MTCP and SSI. As expected the checkpoint duration increases with the memory footprint of the test application. Native BLCR, MTCP and SSI outperform GCS-based checkpointing towards A10, A50, A100 and A500. However, the GCS overhead is within milliseconds. In A1000 there are differences in the range of seconds caused by dynamic network traffic behavior and non-linear disk access for these large checkpoint file images.
For the checkpoint operation the overhead introduced by GCS is negligible. The GCS-based checkpointing operation is longer because the native checkpointing process needs to be split into several phases: prepare, stop, checkpoint and resume. The latter is required to provide consistent checkpoints for distributed applications in the context of coordinated checkpointing.

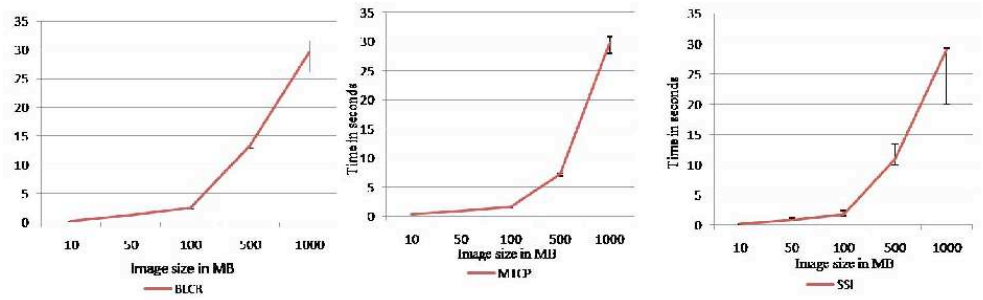


Figure 4.1: Checkpointing with native BLCR (l), MTCP (m) and SSI (r)

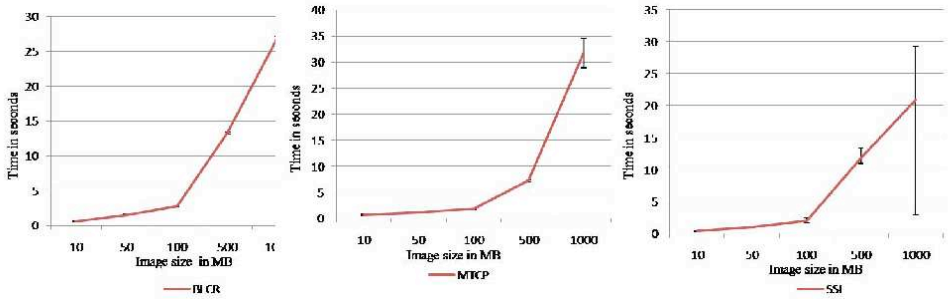


Figure 4.2: GCA Checkpointing with modified BLCR (l), MTCP (m) and SSI (r)

Finally, the diagrams in figure 4.3 and 4.4 show native and GCS initiated restart times. The difference between native and GCS-based restart is also within milliseconds for A10, A50 and A100. For A500 and A1000 however within multiple seconds. These differences are again caused by the underlying file system and seek times of the disk to read this large scattered checkpoint-images. Furthermore, the GCS-based restart also requires the phases rebuild and resume whereas the native restart not.
Obviously, checkpoint image input and output performance depends on the underlying network and disk infrastructure. Having a network bandwidth of 134 MB/s and NFS server disk with a write speed of 300 MB/s (asynchronous mode) and read speed of 120 MB/s explain the measured numbers.
Summarized, the overhead introduced by GCS is negligible both for the checkpointing and restart operation.

Distributed checkpointing/restart Figures 4.5 and 4.6 show measurements of coordinated checkpointing of a distributed application - a job consisting of nine and fifteen job-units. Nine job-units are used in the SSI-MAX configuration featuring seven SSI checkpointers (seven two-node clusters) and one BLCR and one MTCP. The x-axis shows the accumulated checkpointing time of all nine job-unit checkpoints (9 units x A10 = 90 MB, 9 units x A50 = 450 MB, ...). Fifteen job-

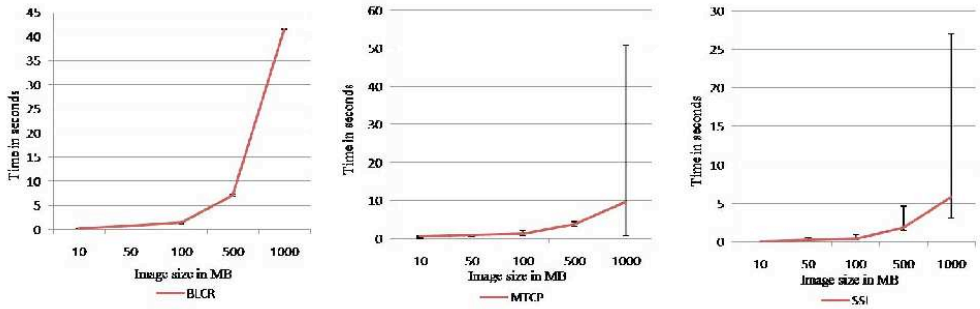


Figure 4.3: Restart with native BLCR (l), MTCP (M) and SSI (r)

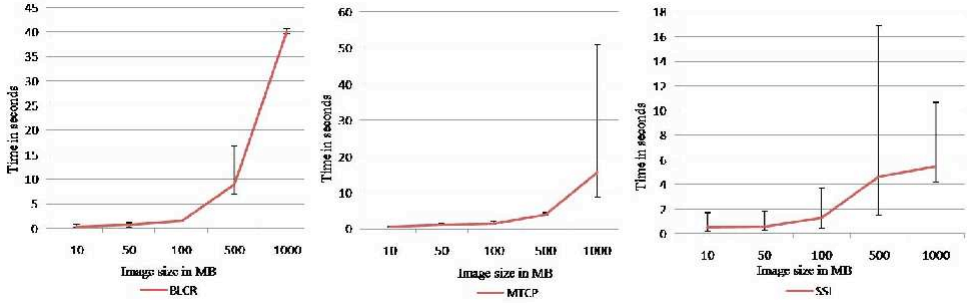


Figure 4.4: GCA restart using BLCR (l), MTCP (M) and SSI (r)

units are used in the SSI-MIN configuration featuring one SSI checkpointer and seven BLCR and seven MTCP. No channel flushing is included in these tests.
The checkpointing time in SSI-MIN is larger then the one for SSI-MAX because of SSI-MIN includes more nodes resulting in a larger checkpointing image size. While SSI-MIN requires 60 sec to save 15 GB does SSI-MAX take 30 s for 9 GB. Despite a node relation of 9:16 does SSI-MIN take twice as long as SSI-MAX. Thus, it is obvious that the native SSI checkpointer is more efficient than MTCP and BLCR. Additionally, the increased NFS transfer and storage overhead of fifteen job-units impacts the measurements, too.
Figures 4.7 and 4.8 show the restart times of a job consisting of nine and fifteen job-units. The long restart times for 4.5 GB and 9 GB memory footprint at SSI-MAX and 7.5 GB and 15 GB at SSI-MIN are strange. While 630 seconds are required to restart a 9 GB image with SSI-MAX, SSI-MIN takes 2350 seconds for a 15 GB image. Obviously, handling fifteen restart requests (SSI-MIN) takes longer than nine (SSI-MAX). These values can be explained by the disk and network traffic relation. While each NFS client may take advantage of a network bandwidth of 134 MB/s, the NFS server in turn must handle fifteen / nine incoming and outgoing data streams. Thus, the disk read and write bandwidth is the bottleneck and responsible for the tremendous overhead associated with SSI-MIN.

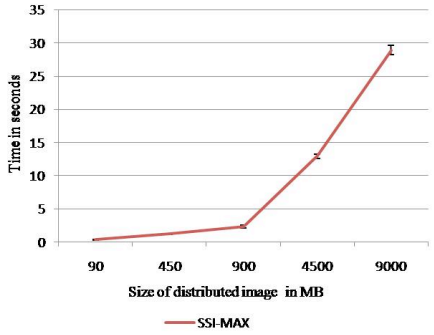


Figure 4.5: Checkpointing with SSI-MAX configuration

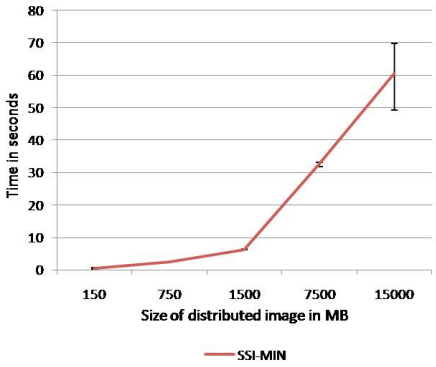


Figure 4.6: Checkpointing with SSI-MIN configuration

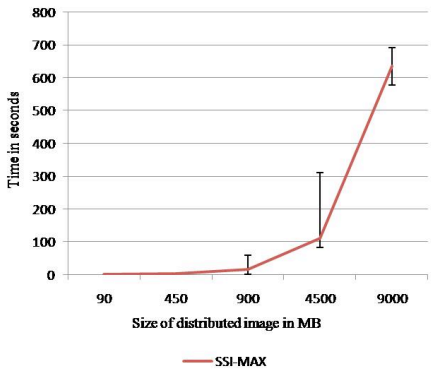


Figure 4.7: Restart with SSI-MAX configuration

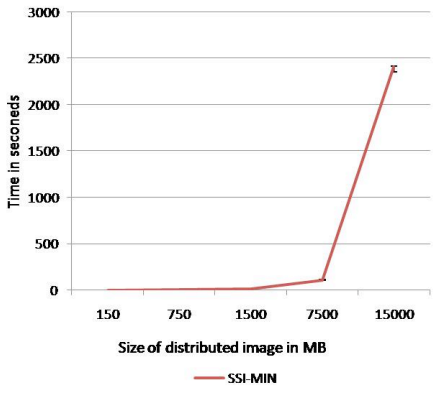


Figure 4.8: Restart with SSI-MIN configuration

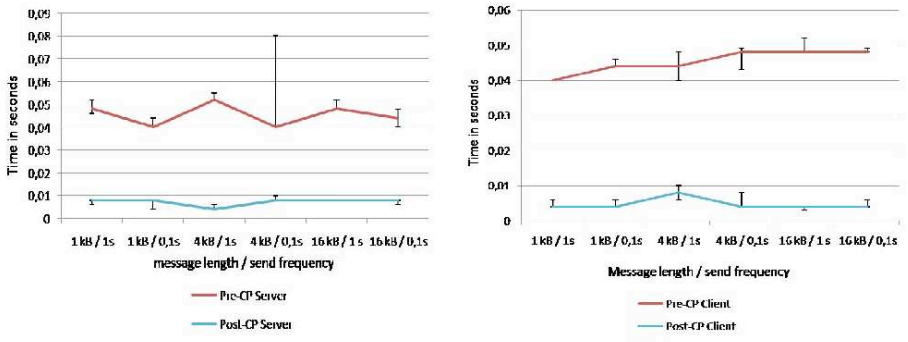


Figure 4.9: Grid channel saving with channel closure and reestablishment

Channel flushing The XtreemOS channel flushing component has been described under [24]. In-transit messages of reliable communication channels must be taken into account during checkpointing to guarantee consistent snapshots.
CF has been evaluated using a client-server application. The server periodically sends messages with varying message lengths and in varying time intervals. Figure 4.9 shows the measurement results if sockets are closed before checkpointing operation. This approach is required for some checkpointing packages which do not support open sockets during checkpointing operation.
Figure 4.9 shows the average duration of the pre- and post-checkpoint phase for a server (left) and fifteen clients (right). Obviously, the pre-checkpointing phase takes longer than the post-checkpointing phase. Pre checkpointing covers the most of the work associated with CF, such as channel control threads coordination, the actual channel flushing and the channel removal. Post checkpoint has a minor workload, it merely covers socket recovery and input of buffered messages which requires less coordination with control threads.
The message length does not impact the pre-checkpoint phase time, it remains at about 0.048 seconds with 1, 4, 16 kB message sizes. Clearly, the high bandwidth available in the testbed does not cause any bottleneck here. The server requires approximately the same time as the clients due to concurrent programming (each channel is handled by a dedicated server thread to avoid blocking). Figure 4.10 shows the measurement values for the pre- and post-checkpointing phases whereas channels are not closed and reconstructed. Pre-checkpointing still takes longer than post-checkpointing because it covers most of the channel checkpointing complexity as described above. The impact of not saving and recreating channels is evident since both phases take less than in figure 4.9.

4.3.4 Test Unit 03: Checkpointing on Linux SSI

4.3.4.1 Responsibilities

The responsible partner for this test unit is XLAB.

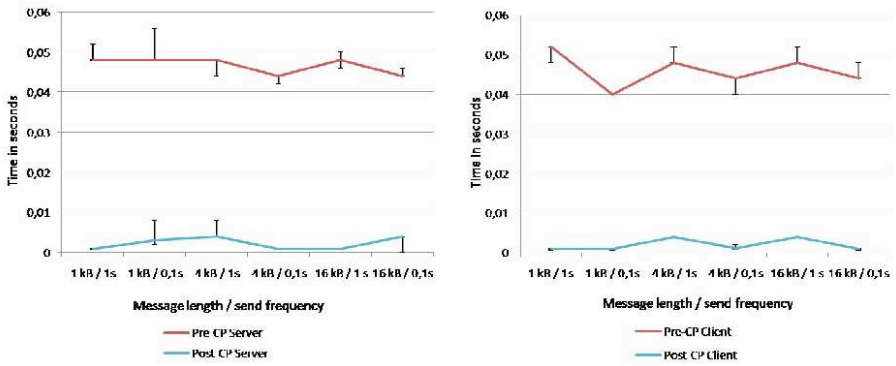


Figure 4.10: Grid channel saving without channel closure and reestablishment

4.3.4.2 Test Specification

Test Items

The test item is LinuxSSI as distributed with XtreemOS release 2.1.2.

Features to be Tested

This tests will evaluate the checkpoint and restart feature in LinuxSSI.

Approach Refinements

We will checkpoint and restart two versions of Galeb, a command-line function fitter based on a genetic algorithm:

- the serial version,
- the multicore version, consisting of one master process and multiple slave processes communicating over System V IPC message queues.

The tests with the serial version consist of simply running an application, checkpointing it, saving its results, restarting, and comparing the results of the restarted version with the original. The multicore tests will be done in a similar fashion. However, they will stress LinuxSSI more because the slave processes will be distributed over multiple nodes of the SSI cluster. Furthermore, the checkpoints will be taken at various points of the execution - before launching the slave processes, during the calculation, and after calculation but before reading the results from the message queue.

4.3.4.3 Test Results

The first requirement for this test was setting up a two-node testbed. Unfortunately this step failed, thus the test could not be executed.

We followed all the steps in the instructions in the Administrator Guide. We noticed that installing `kanif` was redundant because it is a dependency of `task-xtreemos-linuxssi`. Then, when executing the command `start_linuxssi` on the core node, the connection to the other node failed, even though before issuing this command the network connectivity in both directions was checked. We have submitted bug no. 312 to the tracker.

We also tried starting the cluster with `modprobe kerrighed ; krgadm cluster start`, which resulted in a partially working cluster. The console of the head node was frozen, although `top` on the other node did show the CPUs and RAM of both machines.

We then re-installed both machines from scratch, this time omitting the X desktop and checking LinuxSSI during installation. However, LinuxSSI did not actually install (submitted bug no. 311) and could also not be installed later because of a missing kerrighed module.

The installation will be repeated once these bugs are fixed and the planned tests will be done once the installation succeeds.

4.3.5 Test Summary Report

4.3.5.1 Summary of Tests and Results

The openVZ container checkpointing mechanisms could be executed successfully. The time taken to submit a job to an openVZ container is around 5.5 times greater than native submissions. Still the execution in a container may be profitable for applications demanding a high degree of isolation. The scalability test of openVZ checkpointing integrated into XtreemOS reveal practically no difference to the openVZ checkpointing on native Linux.

Tests with the XtreemOS Grid Checkpointing Service (GCS) show that GCS overhead is negligible. Only in one case there are differences in the range of seconds caused by dynamic network traffic behavior and non-linear disk access for these large checkpoint file images and also seek times of the disk to read this large scattered checkpoint-images. Long restart times could be measured for restarting large images which could be explained by the bottleneck wrt. disk read and write bandwidth which is responsible for the tremendous overhead. Also the average duration of the pre- and post-checkpoint phase was measured. The pre-checkpointing phase takes longer than the post-checkpointing phase. Pre checkpointing covers the most of the work associated with channel flushing, such as channel control threads coordination, the actual channel flushing and the channel removal. Post checkpoint has a minor workload, it merely covers socket recovery and input of buffered messages which requires less coordination with control threads. No significant correlation could be found between the message length the pre-checkpoint

phase time.

Tests with checkpointing on the current LinuxSSI failed because the setup of the appropriate testbed could not be prepared. Bugs were reported accordingly and it is planned to repeat the tests when fixes have been released.

4.4 Evaluation of LinuxSSI

LinuxSSI is the base component for the XtreemOS cluster flavor and is developed within WP2.2. LinuxSSI is composed of the Kerrighed cluster operating system providing Single System Image, and additional advanced features. The Single System Image property of Kerrighed allows application programmers and users, as well as administrators to use a cluster as if it were a standard standalone Linux PC. This section presents the evaluation of the latest advanced features, that were developed during the last year of the project. Those evaluations were done under the responsibility of Kerlabs in Task 2.2.13.

4.4.1 Test Plan

4.4.1.1 Responsibilities

WP2.2, Kerlabs, XLAB

4.4.1.2 Test Items

The software tested is Kerrighed 3.0.0-xos, which will be packaged for XtreemOS 3.0. Kerrighed 3.0.0-xos consists of Kerrighed 3.0.0, released to the public in June 2010 and available on Kerrighed's community web site[8], and advanced features (global IP and dynamic streams) that will appear in later releases of Kerrighed.

4.4.1.3 Features to be Tested

The following features of Kerrighed 3.0.0-xos should be tested:

- Checkpointing and restart of System V IPC objects.
- Global external IP address, that is the ability to access the cluster through a unique IP address without making communications go through a front-end node.

4.4.1.4 Features not to be Tested

- Advanced scheduling of applications on a cluster. An evaluation of this feature was already presented in Evaluation Report D4.2.6[10], Section 4.8.9, and the scheduling components had no significant changes since this evaluation.
- Dynamic streams with checkpointing, which is the ability to efficiently migrate and checkpoint applications communicating through sockets inside the cluster. A stabilization phase was planned for this feature during the extension period of the XtreemOS project but this feature could not be stabilized enough to present an evaluation.

4.4.1.5 Overall Approach

The tested features (System V IPC checkpoint / restart and global external IP address) are unparalleled. For this reason, we evaluate them performance-wise through micro-benchmarks.

4.4.2 Test Unit 01: Checkpointing and Restart

4.4.2.1 Responsibilities

WP2.2, Kerlabs

4.4.2.2 Test Specification

Test Items

In this section, the LinuxSSI kernel checkpointer is evaluated. LinuxSSI kernel checkpointer provides fault tolerance for applications running on top of XtreemOS SSI based flavor. We have tested the LinuxSSI kernel checkpointer both in completeness and performance.

The LinuxSSI kernel checkpointer is installed as a part of LinuxSSI kernel and utilities, 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 part of this test unit, we will measure the following LinuxSSI checkpointer characteristics:

- The impact of application memory footprint size variation on checkpoint/restart execution time
- The performance of checkpoint/restart of Inter-Process Communication (IPC) objects. IPC objects are kernel persistent objects that do not offer backup or copy interfaces. Those objects implement various mechanisms useful for synchronization and communication of several processes in a workflow. The LinuxSSI checkpointer implements checkpoint/restore interface for the following IPC objects : System V shared memory segment, System V message queue, System V semaphore.

Approach Refinements

To check the completeness of LinuxSSI checkpointer and study the impact of application memory footprint on the time needed to checkpoint / restart the

application, we have chosen to use Blender as a practical example of application. Blender is a 3D graphic multi-threaded application that can be used for modeling, rendering and simulation.

During the execution of the test, Blender instance is rendering a 3D scene into a range of PNG files. To change the memory footprint, the resolution of the rendered images have been configured differently for different runs. Anyway, since Blender allocates/frees memory dynamically during the computation, its memory footprint is not constant during a run. We have measured around fifty times the time needed to checkpoint Blender instance. Between each checkpoint, the application is progressing and we have checked that the results of the application equal the results of a run without interruption. After reboot of all cluster nodes, to avoid file-system cache effect, we have measured the times to restore the application from each previous checkpoint.

We have measured the time needed to checkpoint and to restart IPC SYSV shared memory segment. For different sizes of the segment (1MB, 10MB, 100MB, 200MB, 1024MB), we have taken 10 measures. By default, size of shared memory segment is limited to 35MB. This limit has been increased for the test. Checkpoint is triggered from a node on which no process has used the segment. Shared memory segments have been created using command *ipcshm-tool* provided with standard distribution of Kerrighed. Average time and standard deviation have been computed for both checkpoint and restart.

We have measured the time needed to checkpoint and to restart IPC SYSV semaphore array with different number of semaphores per array (1, 100, 250, 1000). By default, the maximum number of semaphores per array is 250. This limit has been extended for the test. Semaphore arrays have been created using command *ipcsem-tool* provided with standard distribution of Kerrighed. Average time and standard deviation have been computed for both checkpoint and restart.

We have studied the impact of variation of the size and number of messages on the checkpoint and restart time of message queues. We have changed the number of messages and the size of each messages to have a message queue that always measure 16384 bytes, which is the default maximum size. For each configuration, we have measured checkpoint and restart times 10 times. Message queues and their messages have been created using command *ipcmsg-tool* provided with standard distribution of Kerrighed. Average time and standard deviation have been computed for both checkpoint and restart.

The test unit has been realised on a four nodes cluster running LinuxSSI 3.0, each having the following specifications:

- Dell Optiplex 330
- Intel Core 2 Duo E4400 2 GHz
- 2 GB RAM
- Gigabit Ethernet

Checkpoint files are read/written from/to a NFS file-system shared by a fifth node.

All checkpoint and restart execution durations have been measured from user-space using the `time` command.

4.4.2.3 Test Results

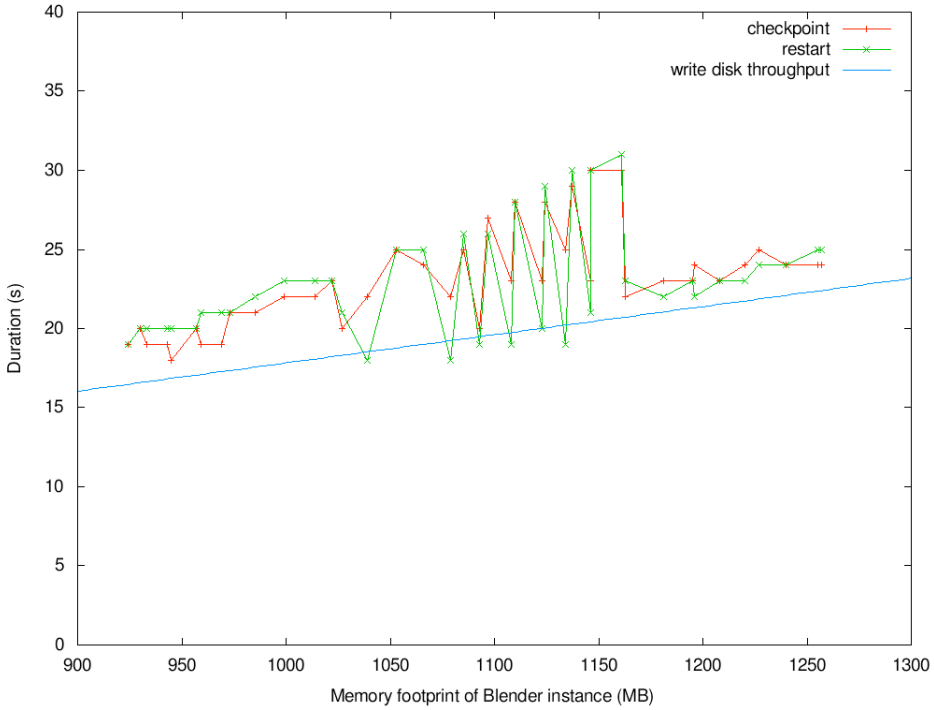


Figure 4.11: Checkpoint/restart execution duration of Blender instance

The checkpoint and restart execution durations are presented in figure 4.11. Memory footprint increasing interval is not perfectly regular because we do not control Blender memory application. As expected, the duration increases with the memory footprint. Checkpoint and restart durations are very similar, meaning than extracting data from the kernel and writing it to stable storage takes approximatively the same time as reading data from stable storage and recreate the kernel structures from the data. Peaks of both checkpoint and restart execution durations for the same checkpoint are explained by unusual in-kernel structures complexity that make the structures parsing and restoration cost more.

Figure 4.12 and table 4.2 report results of checkpoint and restart of shared memory segments. As expected, checkpoint and restart duration increases with size of the checkpoint. Checkpoint is slower than disk throughput because we have tested the worst case, where no pages of the shared memory segment are locally

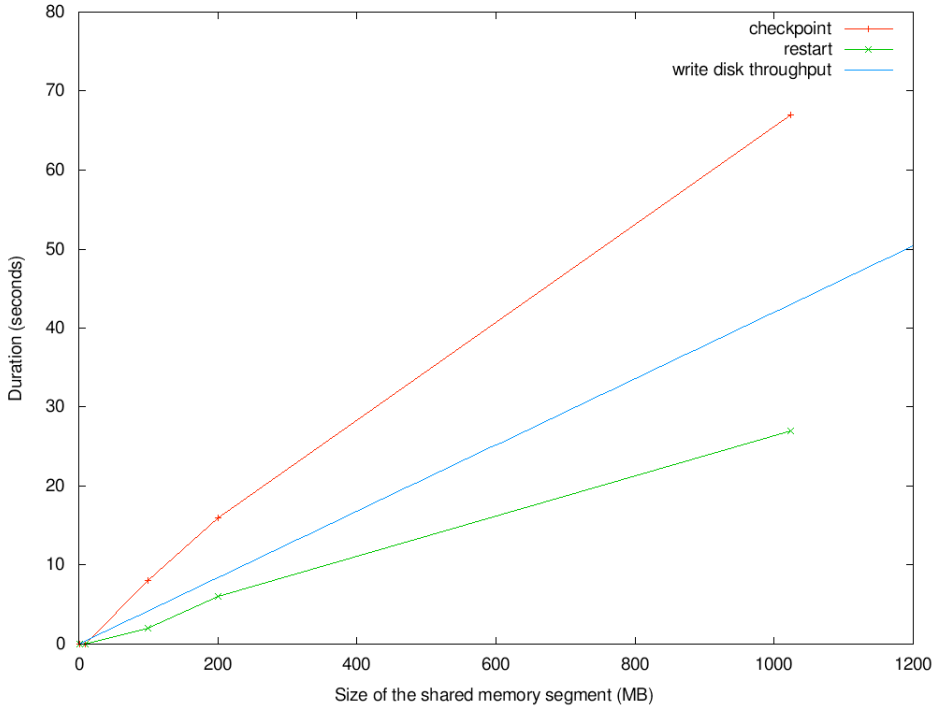


Figure 4.12: Average shared memory segment checkpoint and restart execution duration

shm size (MB)	checkpoint duration (s)		restart duration (s)	
	average	standard deviation	average	standard deviation
1	0,18	0,09	0,03	0,03
10	0,91	0,38	0,03	0,06
100	8,48	2,81	0,12	0,08
200	16,49	6,05	0,66	0,46
1024	67,40	27,60	1,11	1,77

Table 4.2: Average and standard deviation of shared memory segment checkpoint/restart execution duration

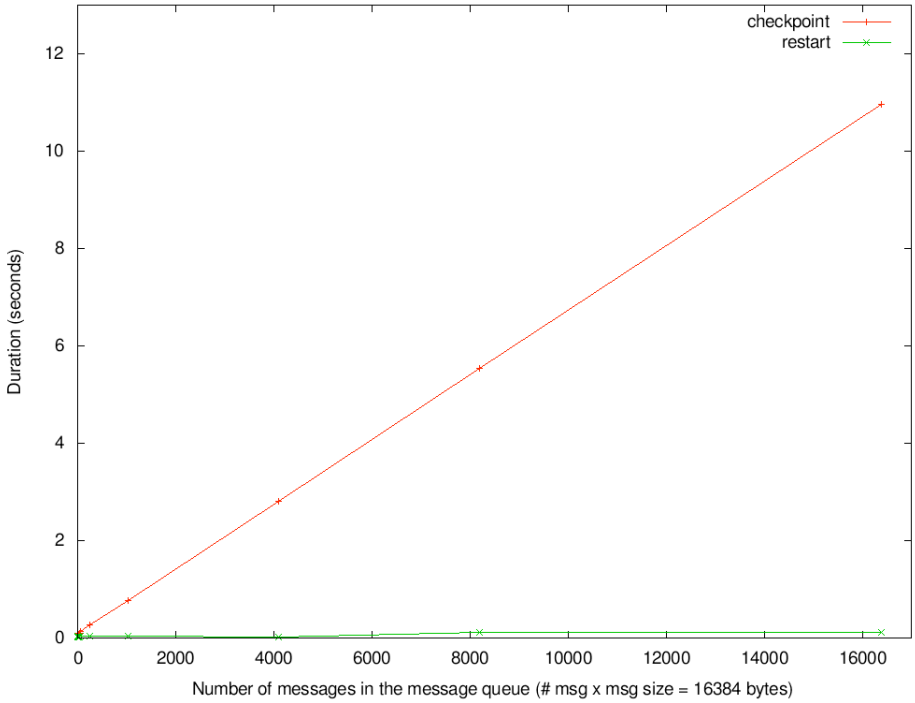


Figure 4.13: Average message queue checkpoint and restart execution duration

number of messages	message size (bytes)	checkpoint duration (s)		restart duration (s)	
		average	standard deviation	average	standard deviation
2	8192	0.08	0.02	0.10	0.05
4	4096	0.08	0.02	0.07	0.04
8	2048	0.09	0.03	0.11	0.04
16	1024	0.08	0.02	0.09	0.03
32	512	0.10	0.02	0.10	0.06
64	256	0.13	0.03	0.09	0.03
256	64	0.26	0.03	0.10	0.04
1024	16	0.76	0.03	0.10	0.04
4096	4	2.80	0.02	0.13	0.04
8192	2	5.53	0.12	0.03	0.04
16384	1	10.96	0.12	0.02	0.04

Table 4.3: Average and standard deviation of message queue checkpoint/restart execution duration

Number of semaphores in the array	Checkpoint duration (s)	Restart duration (s)
1	0.01	0.01
100	0.01	0.01
250	0.01	0.01
1000	0.02	0.04

Table 4.4: Average and standard deviation of semaphore array checkpoint/restart execution duration

available on the node from which the checkpoint is triggered.

Figure 4.13 and table 4.3 present measurements of checkpoint and restart of message queue. As shown, the checkpoint size mostly depends on the number of messages in the message queue which is consistent with the implementation. Indeed, when checkpointing a message queue hosted on a remote node, at least one network packet is received for each message, leading to a bad throughput in case there are a lot of small messages in the message queue.

The checkpoint and restart execution durations of IPC semaphores are presented in table 4.4. It is really fast and does not depend much on number of semaphores in the array, which is not surprising since a semaphore is basically one byte only.

4.4.3 Test Unit 02: Global external IP

4.4.3.1 Responsibilities

WP2.2, Kerlabs

4.4.3.2 Test Specification

Test Items

In this test unit we test the global external IP feature using Kerrighed 3.0.0-xos, provided by packages `kerrighed-kernel` and `kerrighed-tools` in XtreemOS 3.0. The users and installation guides are available in the `kerrighed-tools` package.

Features to be Tested

Global external IP.

Approach Refinements

The goal of this test is to evaluate the performance of communicating with a Kerrighed cluster through the global external IP. To this end we used a standard benchmark for networking called Netperf. Version 2.4.4 was used and is available on Netperf's website[15] as well as in major Linux distributions. Performance comparisons were also done with a vanilla Linux 2.6.30 kernel, on which the Kerrighed 3.0.0 kernel is based, and available from Linux kernel's website[40].

Netperf provides two classes of tests, allowing to benchmark the bandwidth and latency between a client and a server over an IP network. The bandwidth is benchmarked as the bulk unidirectional data transfer rate, in the TCP_STREAM (using TCP) and UDP_STREAM (using UDP) tests. The tests output the data transfer rate in 10^6 bits per second. Parameters that can be tuned are the send and receive socket buffers size on both sides (client and server), the size of the messages sent through the `send()` system call (sender side), and the size of the buffer passed to the `recv()` system call (receiver side).

The latency is benchmarked as the request-response rate, in the TCP_RR (using TCP) and UDP_RR (using UDP) tests. The client sends a request to the server (beginning of a transaction), and wait for the server's response (end of the transaction) before sending the next request. The tests output the request-response rate in number of transactions per second. Parameters that can be tuned are the send and receive socket buffers size on both sides (client and server), and the request and response size.

In order to evaluate the performance observed and thus to evaluate the impact of Kerrighed on the network performance, we ran the netperf benchmark in four different cases reflecting the different levels of impact brought by the Kerrighed patch on Linux 2.6.30. The levels of impact result from the current design of Kerrighed, that builds an SSI cluster out of Linux containers (see [38] for details about Linux containers). A Linux container being part of a Kerrighed cluster is called a Kerrighed container. The first level of impact is at the host system, and results from the Kerrighed hooks inserted in the code and the additional network stack setup for the Kerrighed container on each node of the cluster. The second level of impact is in the Kerrighed container, and results from the Kerrighed hooks enabled as well as the specific network configuration for the container and the global external IP address. The third and last level of impact is in the Kerrighed container, when network communications go through the global external IP address.

Because of the three levels of impact above, the netperf benchmark was run in four different cases. In all cases, a cluster of 4 PCs was used, using an NFS server on a fifth PC (The hardware is described later). The netperf client was always run on the NFS server, and the netperf server was always run on the nodes. Finally, in all but the first case, the nodes run a Kerrighed kernel with an SSI cluster setup. The four cases are the following:

1. The nodes of the cluster run a vanilla Linux 2.6.30 kernel. This case serves as a reference for all other measures.

2. The netperf server runs in the host system of the nodes. The netperf client is

run once for each node. This case measures the impact of having Kerrighed hooks inserted (but not used by netperf) in the code, and a specific network stack setup with the global external IP address (but not used by netperf) for the Kerrighed container.

3. The netperf server runs in the Kerrighed container of the nodes. The netperf client accesses the netperf server through the node's own IP address for its Kerrighed container. The netperf client is run once for each node. This case measures the impact of using Kerrighed hooks and the specific network stack setup for the Kerrighed container, without making netperf use the global external IP address.

4. The netperf server runs in the Kerrighed container of the nodes. The netperf client accesses the netperf server through the global external IP address of the cluster. The netperf client is run once for each possible location (that is once for each node) of the netperf server. This case demonstrates the global external IP address feature (the netperf server should behave equally, whatever its location). This case also measures the impact of using the global external IP address.

The parameters used for the different netperf tests are the following:

TCP_STREAM

- Send message size (in bytes): successively 1, 4, 16, 64, 256, 1024, 4096, 16384, 65536.
- Receive message size: 32768 bytes.
- Socket send and receive buffers size: 32768 bytes requested, 65536 bytes allocated[1] for both the client and the server.

UDP_STREAM

- Send message size (in bytes): successively 16, 1024, 1460 (maximum UDP message size fitting in an Ethernet frame for the virtual VLAN interfaces of the Kerrighed containers).
- Receive message size: 32768 bytes.
- Socket send and receive buffers size: 32768 bytes requested, 65536 bytes allocated for both the client and the server.

[1] Linux usually doubles the size of socket buffers, in order to keep space for internal book-keeping.

TCP_RR

- Request size/Response size (in bytes): successively 1/1, 64/64, 100/200, 128/8192.
- Socket send / receive buffers size: 2048 bytes / 256 bytes allocated for both the client and the server.

UDP_RR

- Request size/Response size (in bytes): successively 1/1, 64/64, 100/200, 1024/1024.
- Socket send / receive buffers size: 124928 bytes / 124928 bytes allocated for the client, and 120832 bytes / 120832 bytes allocated for the server.

The five PCs have identical hardware, with one dual-core 2GHz Intel Core2 Duo E4400 processor, 2GB of RAM, and one Broadcom Corporation NetLink BCM5787 Gigabit NIC plugged in the PCI Express bus. The five PCs were interconnected with a Gigabit Dell PowerConnect 2708 switch.

The network stack in the Kerrighed containers relied on a virtual NIC implemented as a VLAN over the hardware NIC, which in particular results in an 1488 bytes MTU, that is the 1492 bytes MTU of the hardware NIC used in the host system minus 4 bytes for the 802.1q VLAN tag.

4.4.3.3 Test Results

We present the results of the four differents netperf tests run. For each of these tests, we have displayed on one graph the four cases in which the test was run. We can notice that on all graphs and for any of the four cases, the four nodes have equal performance, in particular when the cluster IP address is used. This confirms that all nodes participate equally to the cluster IP address. The following paragraphs detail the analysis for each netperf test.

TCP_STREAM and UDP_STREAM

The results of the TCP_STREAM and UDP_STREAM tests are shown in Figure 4.14 and Figure 4.15 respectively. Since UDP packets can be silently dropped in case of network congestion, the bandwidth displayed for UDP_STREAM was measured on the receiver's side.

We can notice that the host system on a Kerrighed setup and a vanilla Linux kernel have the same performance on this test. In other words, Kerrighed hooks as well as the specific network stack for the Kerrighed container don't impact significantly the bandwidth that can be obtained with TCP and UDP.

However, we can notice that with the network stack of the Kerrighed container, the bandwidth obtained is a bit lower than in the host system. The performance

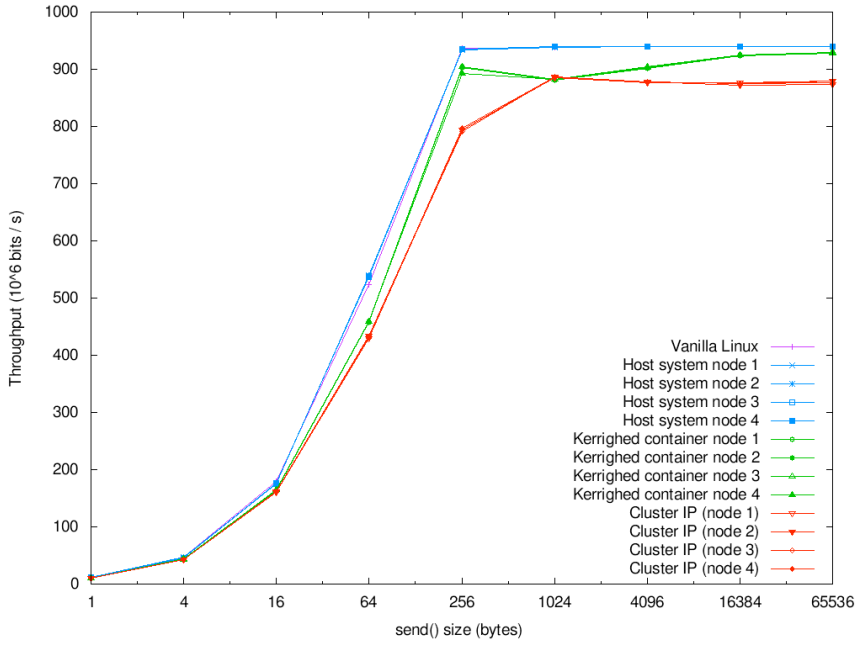


Figure 4.14: TCP data transfer rate from an external client to the cluster

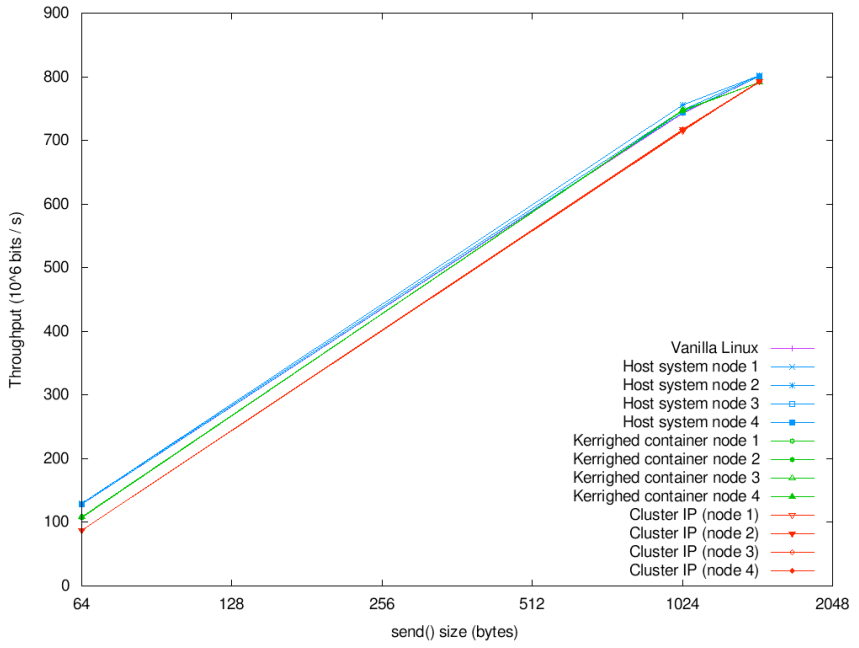


Figure 4.15: UDP data transfer rate from an external client to the cluster

impact is also higher when the cluster IP address is used. Two factors can explain this result. First in the Kerrighed container's network stack the latency is higher than on the host system. Indeed the MTU of the virtual network interface is 1488 bytes large instead of 1492 bytes in the host system. Moreover every packet handled by this network stack traverses the IP tables rule that was defined to setup the cluster IP address, and when the cluster IP address is used, this rule also checks whether the packet should be handled by the local node or not, which increases the overhead.

Second, the cluster IP address relying on Ethernet multicast to send packets to the cluster, the network switch is more loaded by network traffic going to the cluster through the cluster IP address than by network traffic going to a single node's IP address.

TCP_RR and UDP_RR

The results of the TCP_RR and UDP_RR tests are shown in Figure 4.16 and Figure 4.17 respectively.

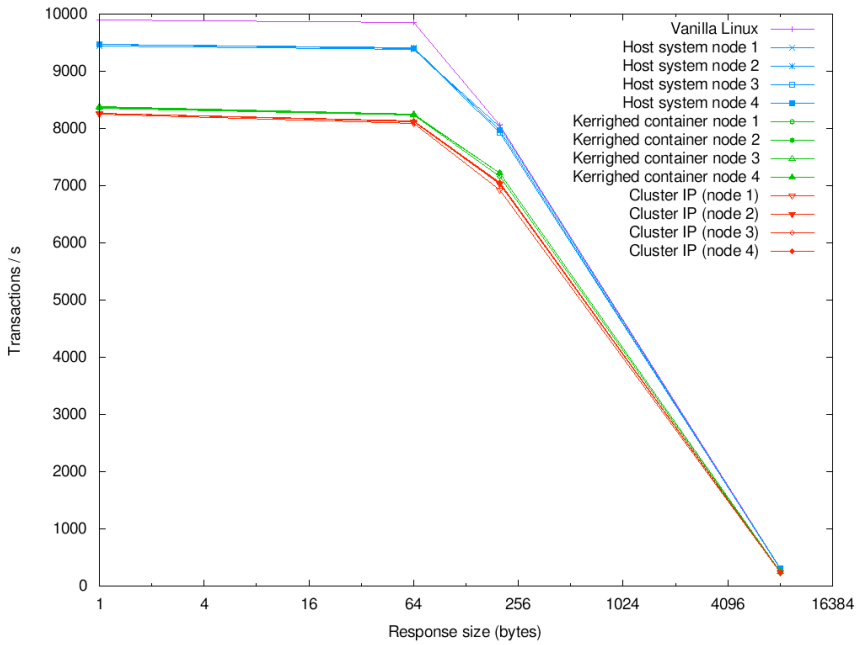


Figure 4.16: TCP request / response rate between an external client and the cluster

Contrary to the TCP_STREAM and UDP_STREAM tests, we can notice that the transaction rate obtained on a vanilla Linux kernel is higher than the one obtained on the host system of a Kerrighed setup, especially for small message sizes. This confirms the overhead introduced by the specific network stack setup of the Kerrighed container. We can also notice that this overhead decreases to zero when

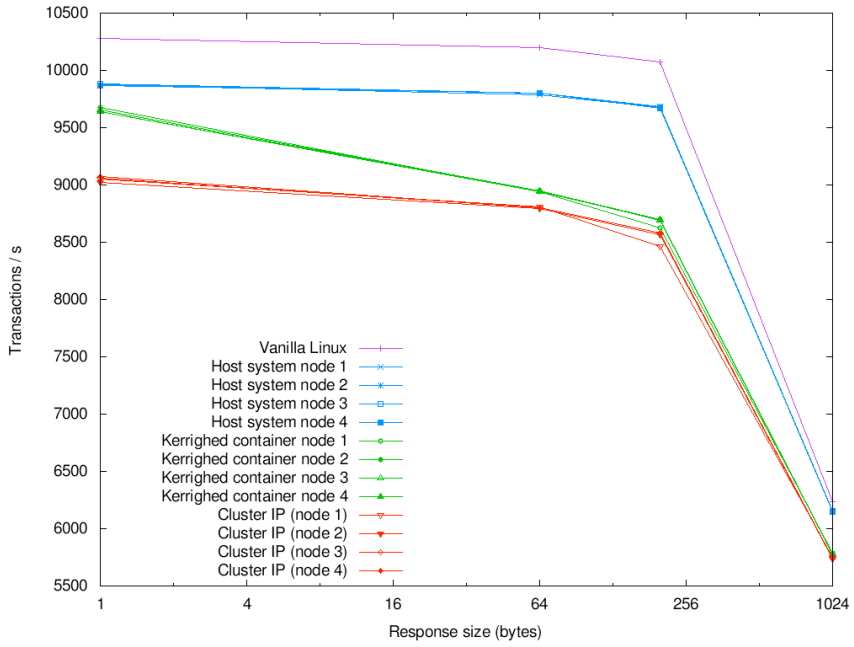


Figure 4.17: UDP request / response rate between an external client and the cluster

the size of responses increases, since the transaction rate becomes more sensitive to the TCP (resp. UDP) bandwidth than to the round trip lantency.

Similarly to the TCP_STREAM and UDP_STREAM tests, we can observe that the performance in the Kerrighed container (using or not the cluster IP address) is lower than in the host system. However, with TCP_STREAM this difference decreases to zero as the response size increases, which confirms that the cluster IP address adds an overhead in latency, but not in bandwidth for traffic going out of the cluster. It is not possible to confirm this observation with UDP_RR because of the limited response size imposed by the network interface MTU, but we can still observe that the performance difference decreases when the response size increases.

4.4.4 Test Summary Report

4.4.4.1 Summary of Tests and Results

We presented tests of two advanced features of Kerrighed: checkpoint / restart, and global external IP address. The tests on checkpoint / restart have shown the completeness and efficiency of the checkpoint / restart implementation in Kerrighed, through intensive checkpoint / restart of the 3D modelization application Blender, and micro-benchmarks of System V IPC objects checkpoint / restart.

The tests on the global external IP address have demonstrated the feature and shown preliminary performance results of this prototype implementation. Both the

achievable data transfer and transaction rate were evaluated through the netperf micro-benchmark. As expected, negative performance impacts were observed, especially in term of bandwidth for traffic going in the cluster through the cluster IP address. The performance obtained with Kerrighed's cluster IP address is however comparable to the performance obtained on a vanilla Linux kernel.

4.4.4.2 Conclusion and Directions for Future Work

We have evaluated advanced features of Kerrighed that improve the quality of service (through advanced checkpoint / restart) and the Single System Image property (through a global external IP address) of Kerrighed. The dynamic streams feature could not be evaluated because of the low quality of the current implementation. One of the primary goals of future work will be to improve the quality of this implementation.

The checkpoint / restart facility of Kerrighed has achieved a high level of completeness, making it now suitable for a broad range of computing applications. Future work on this topic will focus on increasing the flexibility, allowing programmers / users to making applications collaborate with the system's checkpointer in order to optimize the checkpoints' size and time.

The implementation of the global external IP address feature is still at a prototype-stage, and will need further evaluation and optimization to make it suitable for a broad range of applications. Optimization will likely focus on reducing the latency overhead in the Kerrighed container's network stack, by experimenting other network virtualization alternatives like bridged veth interfaces, and by minimizing the decision path when filtering received network packets received on the cluster IP address.

4.5 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 service access points and service call invocation that is abstracted from the means of the service message exchange. The core development was the responsibility of XLAB as a part of the WP3.2.

4.5.1 Test Plan

4.5.1.1 Responsibilities

The DIXI framework is the responsibility of WP3.2 and has been developed by XLAB. The tests have also been carried out by XLAB.

4.5.1.2 Test Items

Here we revisit the components tested for the previous deliverable [10]. Since then, the components have undergone many changes in terms of improved stability, optimisation and added features. The features mostly show an easier way for the developers to use the libraries, but we also expect the lowering of the latencies reported in [10].

We therefore test the libraries that we expect to be packaged in the `dixi-main` containing the service hosting environment and the messaging bus, and the `dixi-xati` to contain the client-side (XATI) libraries, both to appear in the upcoming XtreemOS 3.1 release.

Another major difference from the previous tests is the use of the Grid5000 [5] for hosting the tests.

4.5.1.3 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 the 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.

4.5.1.4 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 only be used at the design time, and their outcome is implicitly tested within the features used during the runtime.

4.5.1.5 Overall Approach

A messaging bus and the staging environment should be as transparent to the user and the underlying services as possible. In our tests we wanted to measure the latencies introduced to the client request and service's response round-trip under various levels of stress. To reduce the external (or service's internal) influences to the time of the request being spent within the system, we implemented a simple echo service which, in its service call, returns the input provided by the clients as its response. The values sent by each clients are different for each request, but comparable in their length (the payload size).

4.5.2 Test Unit 01: client-server timings

4.5.2.1 Responsibilities

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

4.5.2.2 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

- 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.
- Secure communications using the SSL.

Approach Refinements

In this test we wanted to obtain the timings performed by the framework without any delays incurred by the implementation of the user's services staged within the framework. For the test we used two *paramount* nodes on the Rennes Grid5k site. The set-up included:

- the echo service running on a designated node, and
- a client service (i.e., a service invoking the echo service) running on either nodes.

When the client service is running on the same node as the echo service (in fact this means that it is co-hosted within the same Linux process as the echo service), only the memory bus queue is involved in the exchange of the messages. When on different nodes, additional built-in services and the network become involved as well.

The test in either scenarios involves sending an increasing number of requests, measuring the time between just before the service invocation and the point right after the reception of the response. The service calls are asynchronous, meaning that the response arrives in the form of a call-back method invoked by the framework within the client service. The client service sends the next request in the series in the call-back, assuring only one request is active at a time. The series contains up to 1600 requests in a row. We repeat each test 10 times to ensure the results' consistency.

Our tests involve the use of the plain socket communication without using the SSL. However, in this test we also compared the results with those obtained when the SSL is enabled between clients and the service.

In the extension of the test unit, we raised the number of the Grid5000 nodes to 16. In this case, the set-up consists of the following:

- the echo service running on a designated node, and
- all the nodes (including the one running the echo service) run a client service.

In the test, we start each client service to individually run the request series of the original set-up, but we invoke them concurrently. As a result, the echo service receives the requests from all the clients within each test intervals.

4.5.2.3 Test Results

We carried out the test in a sequence of stages, where in the first stage the client sent 200 requests to the service, and in the consequent stages the number of requests increased by 200, the last stage performing 1600 requests. We noticed that the size of the request series does not influence the time spent waiting for the response.

Number of requests	Network	SSL	Average delay	Standard deviation
1600	No	No	1.1 ms	0.5 ms
1600	Yes	No	2.9 ms	0.6 ms
1600	Yes	Yes	3.3 ms	0.7 ms

Table 4.5: The delays (response times) measured in a simple scenario of a single client and a single service.

Of course any other outcome would mean a bug or an architectural problem in the implementation.

Due to the constant outcomes, the table 4.5 shows the summary of the final stage only. We can see that the response times of the service calls invoked within the same process, involving the message queues only, occur on the edge of measurability, at around 1 ms.

When the service message needs to use the network to reach another node, we notice an increase of the delays to around 3 ms per request. In the increase we count additional operations such as the service call redirection to another node, transcoding of the Java objects into the byte arrays, and the actual networking transport. Enabling SSL which, effectively increases the networking payload size as well as adds to the complexity of the communication, on the average represents another 0.4 ms of the delay. Please note that we carried out the remainder of the tests with SSL disabled.

Increasing the number of clients that concurrently compete with a single service naturally increase the waiting times for each individual client. By using a larger number of nodes and running on each node a higher number of clients we test the kind of load it takes to raise the average response times. Again, the number of subsequent requests issued by each client does not influence the response times. Therefore we used the number 1600, granting us stable results and assurance that the overlap between the execution cycle of all the clients is as long as possible.

The Figure 4.18 shows the average delays depending on the number of clients connected and issuing requests. Since we hosted one client per node, the chart at the same time represents the number of connected nodes. For the sake of fairness, the chart shows a separate curve for the delays measured on the client residing on the same node. The curve marked as **two nodes** thus represents an average for all the nodes that do not host the service, but only the client.

The chart shows that for 6 client services or less the latencies are constant. With the increase of the client number, the latencies grow, but due to the low resolution in the measurement timings, the curve is not fully smooth. From the results we also notice that the additional delay caused by the network transport is a constant addition rather than a three-fold slowdown as we may guess from the Table 4.5. Further, the delays increase slowly with the number of requestors, but the performance gives the confidence in the framework.

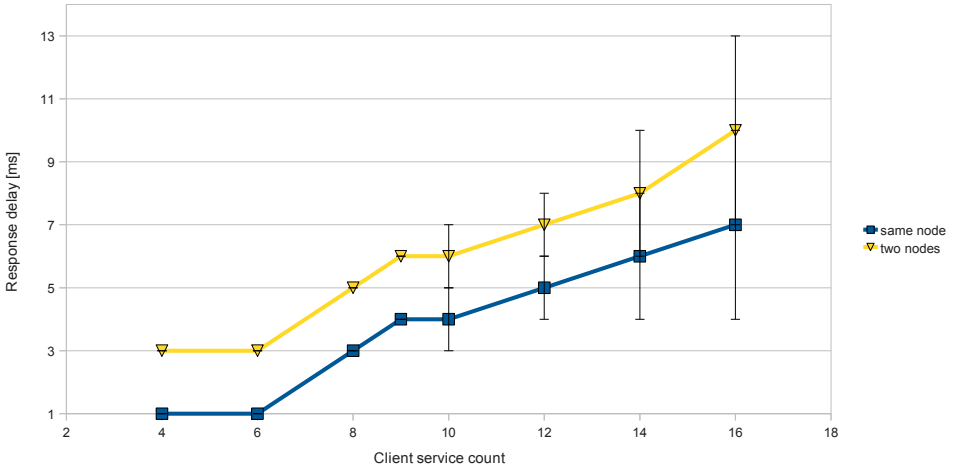


Figure 4.18: Average response times of requests in the test where multiple client services request from the same service.

4.5.3 Test Unit 02: multi-client timings

4.5.3.1 Responsibilities

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

4.5.3.2 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

- Staging the services and exposing their interfaces to the message bus.
- The invocation of the service calls defined by the service interface using client library (XATI), which occurs in a synchronous manner.

Approach Refinements

This test unit logically continues what Test Unit 01 has started by increasing the demands on the echo service by severalfold, again measuring the impact in terms of the response times of the service calls. Again we use the 16 Grid5000 nodes in a set-up as follows:

- the echo service runs on a designated node,
- each node runs up to 20 XATI clients concurrently.

Each instance of the XATI clients internally invokes a number of requests in a fast succession. This is comparable to the calls invoked by the client services in Test Unit 01. However, now we run an increasing number of the XATI clients as Linux jobs, so that each node runs many of the clients at the same time as parallel processes. This effectively simulates up to 320 client nodes requesting the same service.

In this test we also have a number of clients that are hosted with the service on the same node. The difference in this case, however, is, that even in this case they do not share the process, so some inter-process communication takes place.

4.5.3.3 Test Results

The results of the Test Unit 01 hinted at the trend in the increase of the delays as the load intensifies, but we find 16 clients to be a low number, so we needed to add to the load further. In this test, in each round all 16 nodes are involved. What we vary is the number of clients running on each node.

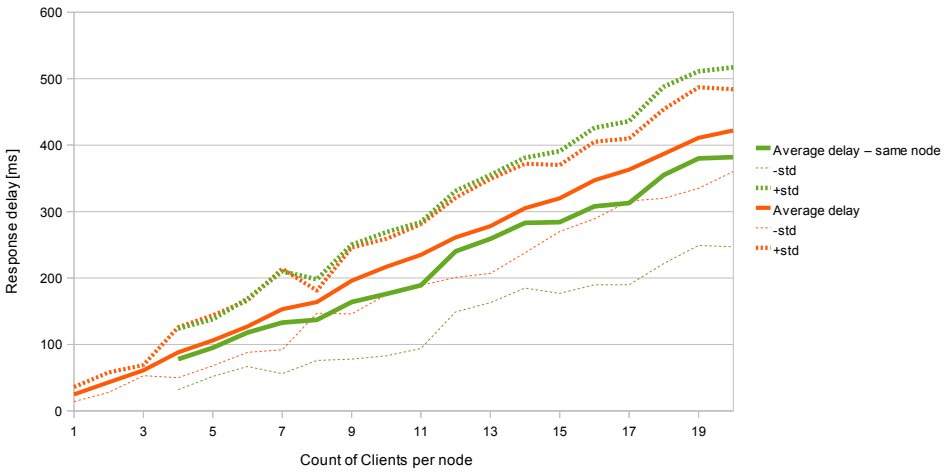


Figure 4.19: Average response times of requests on a single service from multiple nodes, each running multiple concurrent clients.

The Figure 4.19 shows the results of the test. The X axis represents the number of clients on each node. For instance, the client count of 7 per node means that the service receives requests from 112 clients all running in parallel. The rightmost point on the chart therefore represents 320 clients.

Again, we show a separate curve for the co-hosted clients from the remote ones. The dotted lines also show the standard deviation around each of the curve.

The chart shows a steady linear continuation of the trend shown from the previous result, but with a larger extent of the test itself. Along with the average growth, the deviation also spreads slowly, though it is difficult to account the errors fully to the framework itself, as the operating system and the platform running the nodes may introduce some noise.

The clients co-hosted with the service show slightly smaller delays, suggesting that their requests get served with a slightly higher priority on the average. This is not surprising, because they are free of the latencies required by the network transport and can queue their next request faster. However, the larger deviation suggests that it is more likely a particular request will be delayed more than one from a remote client.

4.5.4 Test Unit 03: parallel requests in the queue

4.5.4.1 Responsibilities

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

4.5.4.2 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 possibility to send the service requests in parallel within the service call.

The switch from implementing the synchronous calls to being forced to implement complex solutions using an asynchronous paradigm requires a steep learning curve from many developers. However, once the difficulties are overcome, there are benefits in developing services such that they do not block the execution of a service call any more than necessary.

The previous test cases showed that using callbacks leads to a well-behaved framework. In this test we want to test the effects of using the call-back paradigm only partially. More precisely, instead of keeping at most one request live at a time within a client, we send all the service requests to the same service at the same time.

Approach Refinements

For this test we refer to the set-up of the Test Item 01, i.e.,

- the echo service running on a designated node, and
- all the nodes (including the one running the echo service) run a client service.

The client service, when invoked, sends to the echo service 1200 service calls. It stores the time of the service sending and, when the response arrives, also the time of the response arrival. The responses still return as the call-backs, but none of them perform any further calls.

We run two independent batches of tests, one with the co-located client and server, another one with the client and the server residing on separate nodes. We run each batch 10 times to ensure the stability of the results.

4.5.4.3 Test Results

The Figure 4.20 shows the results of the test. This time, the chart has a form of the time relative to the events of starting and finishing of each individual request. The X axis thus shows the index of the request. The time taken by the request to return is represented by the line from the point on the **start time** line to the point on the **end time** line.

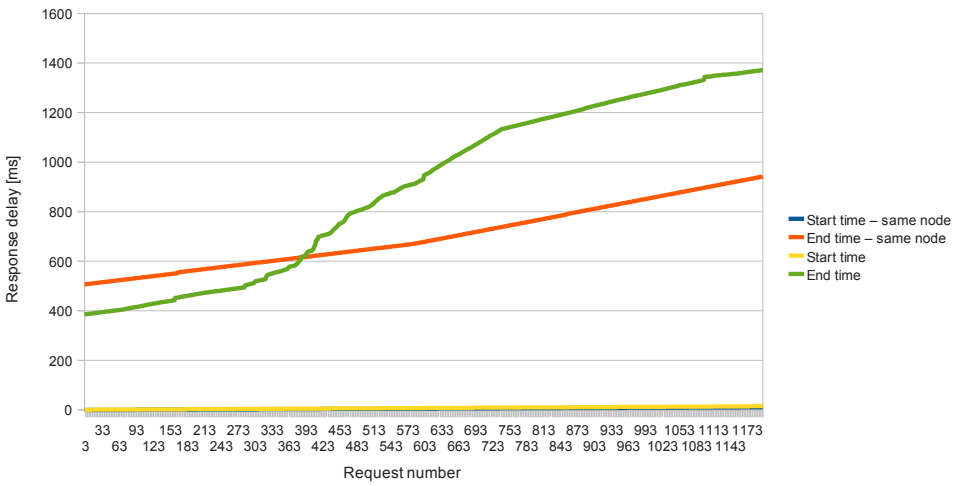


Figure 4.20: Start and end times of the requests issued in an asynchronous batch.

The chart shows that the 1200 requests enter the queue in around 5 ms. The first request's response arrives in around 500 ms for the co-hosted case, and in just under 400 ms for the two-nodes case. The next response in both cases arrives

within 1 ms later, and the subsequent ones follow in a steady succession. It is important to stress that, unlike in the previous test units, here the duration of the whole batch is the time between the start of the batch and the time when the last request's response arrives rather than the sum of the individual delays.

An interesting outcome of the test is that the co-located case receives the first response later than the separated case. We speculate that this is due to some level of distribution of the queueing load when two nodes are involved. However, this soon switches when more and more requests arrive. In either case the delays grow steadily and close-to-linearly without escalating towards a breaking point.

In this case we can tell that the number of requests does influence the response times. This is simply due to the fact that the message bus uses a FIFO queue, therefore all the requests need to be handed out first before the first responses can be provided to the call-backs.

This unit case does not show a realistic scenario, but it explores one possible use of the service calls. It shows that, if ever needed, it is considerably better to construct the call sequence in a single line of execution rather than fanning out extensively. This does not apply for the cases where we need a service to scatter requests to a varous set of nodes in order to gather their responses later on.

4.5.5 Test Summary Report

4.5.5.1 Summary of Tests and Results

In this report we presented a new iteration of the tests performed on the DIXI framework and the message bus. For the tests we did not have access to the same hardware where we carried out the previous tests, so it is difficult to compare the outcomes. However, being able to use the Grid5000 platform, we were able to create set-ups much closer to what the production environment for the XtreemOS would be.

In the tests we progressed from first making simple calls with single client on a single server. We gradually increased the number of the clients, adding to the stress level of the service. All along we noted the slow linear increase in the average response times. We concluded that the number of requests issued by an individual client does not influence the delays, but the number of clients concurrently making service calls has the expected impact on the individual request's response time.

In a special test case we explored the influence of putting a lot of requests quickly into the message bus. Here we found that the length of the queue of messages to be processed has a noticeable influence in terms of the initial request's response time.

4.5.5.2 Conclusion and Directions for Future Work

The purpose of the DIXI was to host a number of services, connected both internally on each node and throughout the network by the built-in message bus. Ideally,

the staging framework would host the front-ends of the services which process incoming service calls quickly, but moves all the lengthy and complex computation "off-line". For these purposes our tests show that the services can be hosted by DIXI efficiently.

Of course it is not always practical to optimise for fast service call processing, particularily when prototyping new services, which is another strong point of the DIXI. On the other hand, the growing grids will provide an increasing service stress. In either case it is difficult to construct a framework to alleviate the slowdowns, but in this case it is better to provide a higher number of replicated services to distribute the load of the clients.

We see a possibility of further enhancements and improvements for DIXI in the ability to distribute the actual load on the distributed services when the service request does not call for a particular host's service. The already built-in service call redirection could take decisions on the service message traffic to use the less-visited parts of the grid.

4.6 Evaluation of XtreemOS API

The XOSAGA XtreemOS API, created mainly by WP3.1, offers an API for the applications to manage jobs, resources and files within XtreemOS.

4.6.1 Test Plan

4.6.1.1 Responsibilities

This test plan is carried out under WP4.2 and involves BSC and EADS.

4.6.1.2 Test Items

First series of tests: AEM and XOSAGA versions available at the SVN on 20/06/2010.
Second series of tests: XOSAGA API v2.1.

4.6.1.3 Features to be Tested

XOSAGA job submission, resource reservation, job monitoring.

4.6.1.4 Features not to be Tested

Rest of XOSAGA features, not relevant or not implemented by the test-case applications.

4.6.1.5 Overall Approach

Series of tests:

- Comparison of AEM and XOSAGA: resources are reserved and jobs are submitted using equivalent API methods in both AEM and XOSAGA, which guarantees fairness of the results. Also the environment used (cluster and number of processors) is the same for both AEM and XOSAGA.

- Job failure rate: the methodology used to calculate the failure rate consists in keeping track of the number of failed jobs out of the total submitted.

4.6.2 Test Unit 01: Java XOSAGA – Performance comparison with AEM

4.6.2.1 Responsibilities

WP4.2, Enric Tejedor from BSC

4.6.2.2 Test Specification

Test Items

For both AEM and XOSAGA, we tested the following items:

- Job management
- Resource management

Features to be Tested

For both AEM and XOSAGA, we tested the following features:

- Job management: submission and monitoring (state checking)
- Resource management: create a reservation, release a reservation

Approach Refinements

While in the previous deliverable we used the hmmpfam application, executed by COMP Superscalar (COMPSs), to evaluate the performance of the Application Execution Management (AEM) component, in the tests presented here we compare the performance of AEM against the one of XOSAGA. The porting of COMPSs on top of XOSAGA allows to establish a direct comparison between the two APIs.

The job and resource management features of both AEM and SAGA are exploited as follows. 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 and then reserve 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.

4.6.2.3 Test Results

In order to evaluate COMPSs-hmmpfam on top of XtreemOS-AEM and XtreemOS-XOSAGA, we conducted some tests to measure the execution time of the application for both portings of the COMPSs runtime.

The testbed that we used is Grid5000. In such testbed, we reserved a set of nodes where a XtreemOS image was deployed, and then we installed COMPSs on each node. We also installed the AEM and XOSAGA versions available at the SVN on 20/06/2010. The nodes reserved correspond to a single cluster of Grid5000, called 'paraquad'. We chose to run the tests in only one cluster, so that the test environment is uniform, the results have less variability and are comparable. If the set of nodes were chosen randomly for each execution, aspects like

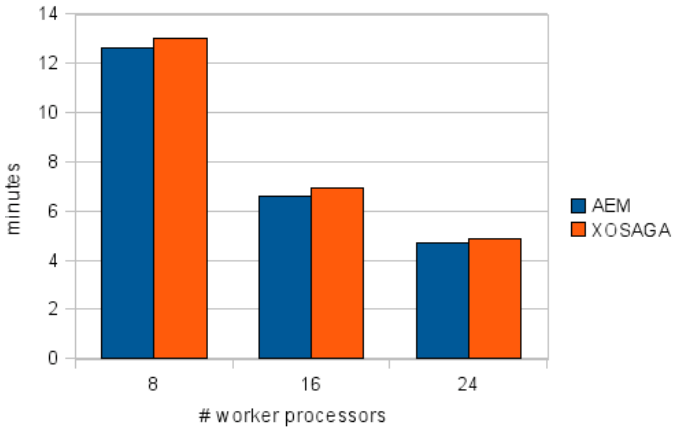


Figure 4.21: Execution times of the hmmpfam application on top of COMPSs, both with the AEM and XOSAGA flavours.

the processor performance or the interconnection network would strongly interfere with the results.

The characteristics for each node of the cluster are the following:

- Intel Xeon X5570 2.93 Ghz
- Memory: 24 GB
- Storage: 500GB / SATA AHCI Controller
- Gigabit Ethernet

For job execution, we used one node as the master (thus hosting the COMPSs runtime) and a variable number of worker nodes (from two to six nodes, using four processors on each node).

In order to compare the performance of AEM and XOSAGA, we ran the same series of tests using two different configurations of the COMPSs runtime: first, the COMPSs runtime ported to AEM; second, the same runtime ported to XOSAGA.

Concerning the data accessed by the application, 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 copied using the 'scp' command to the destination node before the task execution.

Figure 4.21 shows the performance results of running COMPSs-hmmpfam, both using AEM and XOSAGA.

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, although XOSAGA introduces some additional overhead to the job and resource management features of AEM.

4.6.3 Test Unit 02: Evaluation of Parallel Job Submissions using XOSAGA

4.6.3.1 Responsibilities

WP4.2, Lokendra Singh (EADS)

4.6.3.2 Test Specification

Test Items

The test item, mainly, is the variation of Failure rate of parallel jobs when the number of reserved nodes is varied. The jobs were submitted using XOSAGA API v2.1.

Features to be Tested

The main feature to be tested is the effect of size of grid (w.r.t number of nodes) on resource and job management using XOSAGA. Hence, the number of nodes is varied for fix number of parallel jobs submitted, and failure of jobs is recorded.

Approach Refinements

The result are obtained when a large number of jobs are submitted simultaneously (parallel) on XtreemOS Grid, and the number of failed jobs is recorded for each parallel submission. The evaluation study, basically, submits a definite number of parallel jobs on a number of reserved nodes. The jobs which fail, out of the simultaneously submitted jobs, are collected and submitted/run again upto a certain number of such 'Retrials' (after which the job is declared 'Failed'). The failure of jobs is recorded for each of these 'Retrials'.

The number of reserved nodes is varied, to test if the failure rate depends upon number of nodes, hence deducing whether a large grid induces overheads on jobs.

The tests were conducted on Paradent clusters of Grid5000 on XtreemOS-2.1.

The *failureRate* of a trial is defined as *(Number of jobs Failed) / (Number of Jobs submitted simultaneously)*.

4.6.3.3 Test Results

Our applications mainly Amibe and OpenFOAM were ported on XtreemOS using XOSAGA API. We will take the example of Amibe in this study as the illustration would be easy.

The application Amibe, is a meshing application, and creates 3D meshes in three steps. The first step is descretization of geometrical edges (1D elements). The second step is 2D descretization and meshing of every face. The third step involves 3D descretization and merging of meshes to produce final mesh.

The second step of Amibe makes use of parallel resources by submitting all the Mesh2D processes as seperate jobs, simultaneously to the Grid.

The porting of Amibe was completed using Java implementation of XOSAGA-2.1. Following pseudo-code snippet, shows the simultaneous submission of 2D meshing Jobs:

```
    def jobList = [];
while ( i <= noFaces ) {
jobname = "mesh2D"+i;
Job job = jobService.createJob(jobDesc);
job.run();
jobList.add(job);
}
monitorJobs (jobList)
```

The `monitorJobs (jobList)` method takes care of monitoring a job as well as re-submitting/re-running a job, if it is failed. This method, basically, collects a list of failed jobs and makes a 'Retry' to submit/run them all once again, simultaneoulsy. Such retrials are made upto a fixed number of times (we kept it 10).

In a test-case, for having large number of parallel jobs, we used a particular Geomtry CAD file, called as 'lego.brep', which has 35 faces. Hence, 35 jobs are submitted, in parallel, in the second step of Amibe meshing.

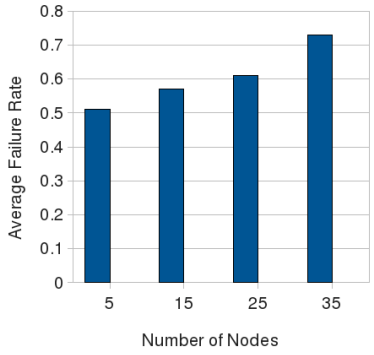


Figure 4.22: Parallel Job Failure Rate vs Number of Nodes

The *failureRate* is defined as *(Number of jobs Failed) / (Number of Jobs submitted simultaneously)*

The average Failure Rate, is average of failure rate from all the retrials made for a set of parallel jobs. The number of nodes is varied from 5 to 35 in steps of 10.

The failure rate has been very high (>50%), for which we later learnt that, there were some bugs related to parallel job submission in 2.1 version of XtreemOS and which will be fixed in later versions of XtreemOS. So, this could be a possible

reason for the observed failure rate.

It is also observed that failure rate of jobs is increasing with increase in number of reserved nodes. Hence, one can deduce that increasing the size of grid is inducing overheads on parallel job execution.

4.6.4 Test Unit 03: XOS MPI – Performance testing

4.6.4.1 Responsibilities

Ronald Fowler from STFC

4.6.4.2 Test Specification

Test Items

This test was designed to show that the MPI (Message Passing Interface) could be used effectively with XtreemOS to run a real scientific application.

Features to be Tested

The aims of this test were:

- To how that a complex real world MPI application can be built on XtreemOS
- To show that such applications can be submitted using AEM
- To report the scaling performance seen in a simple test case

Approach Used

This task used the 2.1 Permanent Test Bed system and looked at the ease of building an MPI application on the system and a basic test of performance scaling. The system had of the order of 10 to 20 computers across Europe connected together in one VO.

The central XtreemFS file server is located at IRISA, France, with resource nodes available in France, Holland, Germany, Italy and the UK. It was hoped to run MPI jobs across the entire VO, but this was not possible due to technical issues with back porting of fixes from XtreemOS 3.0 into version 2.1. These issues are not present in 3.0 and have subsequently been resolved in 2.1, but not in time to rerun the test cases with these updates.

For the highest performance results it would be best to run the application on a dedicated local cluster, close to a XtreemFS file server. This could be done using a cluster of machines within the Grid5000 system with XtreemOS nodes. However there was not time within the extension to do this.

Due to the problems with Mpich, test results are only reported using mpiBLAST linked against the Mpich-2 library. These jobs are launched using the *xsub*

command to send a JSDL file to a single XtreemOS resource. Using a resource with 8 processors allowed testing of the scaling of this application. Very similar results would be expected using the XOS Mpich library since both have similar communication performance.

4.6.4.3 Test Results

Building XtreemOS MPI

The only minor issue was the need to alter the configuration file to ensure that the C++ compiler was correctly recognized, and the wrapper script *mpic++* was generated. This problem arises due to standardization changes in C++ since the MPI library was written. The Mpich-2 package was built without any problem.

Building mpiBLAST on XtreemOS

The mpiBLAST package was successfully linked against both the XOS Mpich and Mpich-2 libraries. Due to the problems with running XOS Mpich jobs results are only reported using Mpich-2.

Building the test data for mpiBLAST

The test data for mpiBLAST was obtained from public resources at NCBI. A database of Baculovirus protein sequences was built using mpiformatdb. For this small set of data it was not necessary to partition the database, as is the case with larger problems.

Parallel performance of mpiBLAST

Figure 4.23 shows the parallel speed up of the mpiBLAST problem on an 8 core XtreemOS node within the 2.1 test bed system. The speed up is close to linear since the search is efficiently parallelised by mpiBLAST. The results are reported for data located on the /tmp partition rather than on the users XtreemFS partition. The XtreemFS file system on the test bed showed variable performance with nodes located in Inria (close to the XtreemFS servers) showing better performance than those at remote sites. In particular file write speeds, writing to XtreemFS, were as low as 0.1MBytes/sec on some remote nodes in the test bed. This can have a noticeable impact on the timings for mpiBlast particularly if the jobs are small compared to the amount of output generated. mpiBlast ran about 10 times slower for a small test case using XtreemFS compared to local disk. Using local storage for database and temporary output files may help performance. The restricted performance of XtreemFS for remote clients may be due to network limitations or configuration problems of the test bed.

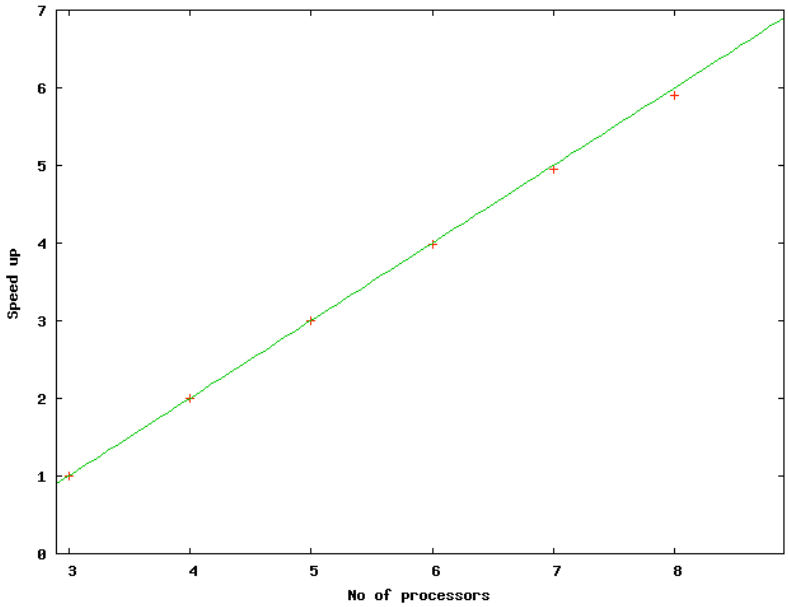


Figure 4.23: The speed up of mpiBlast on the test problem. Note that two processors are dedicated to management tasks, so the speed up is close to linear in NCPUS-2.

4.6.5 Test Summary Report

4.6.5.1 Summary of Tests and Results

Series of tests:

- Comparison of AEM and XOSAGA: the performance regarding job submission, resource management and job monitoring is quite similar for both AEM and XOSAGA, although XOSAGA introduces some additional overhead.
- Job failure rate: the failure rate of jobs is increasing with increase in number of reserved nodes. Hence, one can deduce that increasing the size of grid is inducing overheads on parallel job execution.

4.7 Evaluation of the Resource Selection Service

The resource selection service implements the first step to schedule jobs in XtreemOS: starting from a specification of static node properties that are required for the job, the RSS selects a number of suitable nodes that match the specification. This list of nodes is then further refined by the Service and Resource Discovery Service (SRDS) then the Application Execution Management (AEM) service before the job can actually start.

The RSS is implemented in the form of a fully decentralized peer-to-peer overlay [11]. This provides great properties such as scalability to huge numbers of nodes (see Deliverable D4.2.6 on this topic). On the other hand, the good performance of the RSS depends on the correct selection of a few internal parameters. Choosing these parameters correctly is a non-trivial task, as the optimal choice depends on the statistical distribution of node and attributes. This section evaluates the performance of the self-adaptation algorithms that we have designed to automatically control internal parameters in the presence of sudden or gradual changes in the underlying node or query properties [39].

4.7.1 Test Plan

4.7.1.1 Responsibilities

This performance evaluation is under the responsibility of VUA within WP3.2 (task T3.2.3).

4.7.1.2 Test Items

This evaluation focuses on RSS's self-adaptation *algorithms* rather than their *implementation*: it is realized in simulation environments which allow us to control the evaluation scenarios in ways that would no be realistically feasible in a real deployment.

4.7.1.3 Features to be Tested

This evaluation tests the ability of RSS's self-adaptation algorithms to maintain a low query overhead across various changes in the node property distribution and the query workload. We consider two test cases in our evaluation: an adaptation to changes in the node population, and an adaptation to changes in query workloads. In both test cases, we simulate two types of changes: sudden changes (e.g., an addition of a new computing cluster to the system, or a switch from one type of application to another), and gradual changes (e.g., a system in which old machines are gradually replaced by new machines, or a slow transition in the type of jobs that users tend to run in the system).

We assess the improvement in the RSS performance by measuring the RSS routing overhead, defined as the average number of nodes traversed by a query.

This metric captures both the query routing cost and query latency, since RSS queries traverse nodes sequentially. We also investigate the impact of our self-adaptation protocol on the RSS responsiveness by measuring the query delivery rate, defined as the average fraction of nodes correctly discovered by a query. Finally, we measure the extra maintenance cost introduced in the RSS by our self-adaptation protocol.

4.7.1.4 Features not to be Tested

The performance of RSS's implementation.

4.7.1.5 Overall Approach

Although we evaluate our system through simulation, we use real-world data to initialize node attribute values and several types of queries that closely resemble the workloads from current Grid systems. Specifically, we obtained descriptions of over 300,000 machines that participated in the the BOINC volunteer computing project between 2004 and 2008 [2]. Based on these machine descriptions, we initialize the following four node attributes in the RSS: measured CPU performance in FLOPS, measured downstream bandwidth, amount of installed memory, and amount of installed disk space.

We exercise the system with several types of synthetic query workloads that have similar characteristics to the workloads observed in real Grid systems. Although a number of job traces from Grid systems are available [17], we could not use them directly in our experiments because they mostly contain information about job runtime characteristics (e.g., total running time, amount of used memory) and give very little information about node characteristics required for job execution. In our experiments, we use the following three workload types:

- **bag-of-tasks:** a workload in which a few specific queries appear very frequently. This corresponds to the "bag-of-tasks" type of jobs, that contain a large number of very similar tasks (and thus, a large number of identical job submission queries).

- **coarse-grained:** a workload which simulates user-generated queries. In such queries, attribute ranges are specified in course-grained units. For example, the amount of RAM is specified in multiples of 512 MB.

- **random:** a workload in which all the queries specify random intervals for attribute values. We use this workload as a base for comparison with the other workloads.

4.7.2 Test Unit 01: Adaptation to Changes in Node Properties

4.7.2.1 Responsibilities

This performance evaluation is under the responsibility of VUA within WP3.2 (task T3.2.3).

4.7.2.2 Test Specification

Test Items

RSS self-adaptation algorithms [39].

Features to be Tested

Adaptation to changes in node properties.

Approach Refinements

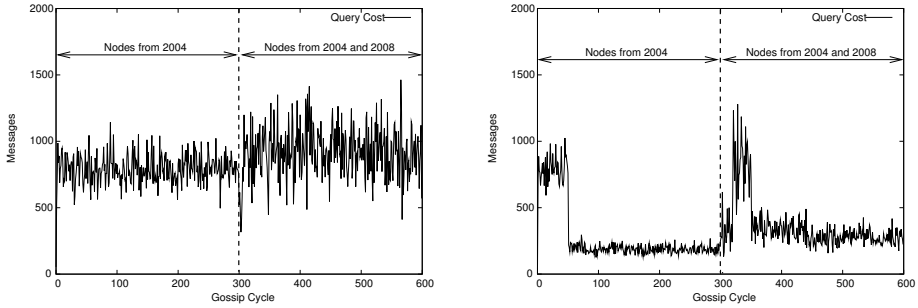
The statistical distribution of node properties may change dramatically when new machines are added to the system, or when they replace older ones. To simulate such situations, we use two sets of node properties based on the BOINC traces from years 2004 and 2008. For this particular experiment we replaced one node attribute (available downstream bandwidth) with the installed kernel version: this attribute suffers much more changes across the years, and allows to stress our system better.

The case when a new computing cluster is added to the system creates a sudden change in the statistical distribution of node properties. We simulated this case by starting the RSS with 5,000 nodes with attribute values obtained from the 2004 traces. After 300 gossip cycles, we added 5,000 more nodes with attribute values from the 2008 trace.

We then show similar simulation results for a situation in which the node properties gradually change from one distribution to another. We create this change by starting with 5,000 nodes from the 2004 BOINC trace, and subsequently replacing a few nodes at each gossip cycle with new ones drawn from the 2008 trace.

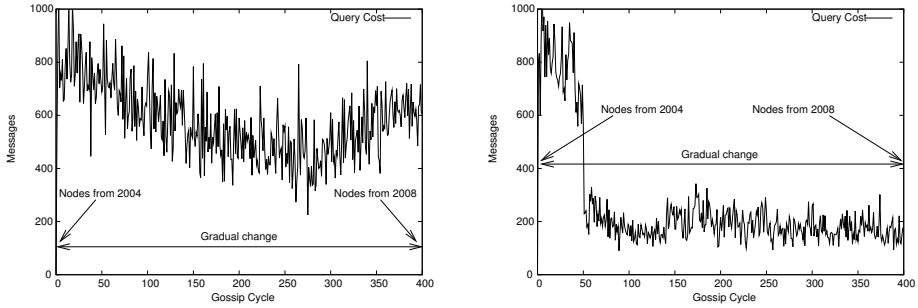
4.7.2.3 Test Results

Figure 4.24 shows the query routing overhead in the case of a sudden addition of many nodes, with and without the self-adaptation protocol running. The first part of Figures 4.24(a) and 4.24(b) show the effect of self-configuration in the RSS. Both systems start with the same set of query boundaries chosen by the human operator, and experience a query cost in the order of 800 messages per query. In the adaptive system, these costs drop by a factor 4 after the first system reconfiguration. At time 300, both systems see a cost increase. Part of this increase is due to the fact



(a) Without self-adaptation (b) With self-adaptation

Figure 4.24: Routing overhead for a sudden change in the node properties.



(a) Without self-adaptation (b) With self-adaptation

Figure 4.25: Routing overhead for gradual changes in the statistical distribution of node properties.

that the size of the system is doubled, and therefore the number of nodes matching the queries also roughly doubles. The adaptive system also sees an additional cost increase due to the fact that its configuration is suddenly ill-suited to the workload. It however quickly adapts to this new situation and returns to an average cost four times lower than the non-adaptive system.

Figure 4.25 presents similar simulation results for a situation in which the node properties gradually change from one distribution to another. Again, the adaptive system shows much better performance than the non-adaptive one. The non-adaptive system sees a relative performance improvement until cycle 250. This is explained by the fact that, in that phase of the experiment, there is a balance between the number of old and new machines, and the nodes are distributed more evenly into cells. The adaptive system, on the other hand, issues several relatively minor reconfigurations, and maintains a constant performance despite the workload variations.

4.7.3 Test Unit 02: Adaptation to Changes in Query Workload

4.7.3.1 Responsibilities

This performance evaluation is under the responsibility of VUA within WP3.2 (task T3.2.3).

4.7.3.2 Test Specification

Test Items

RSS self-adaptation algorithms [39].

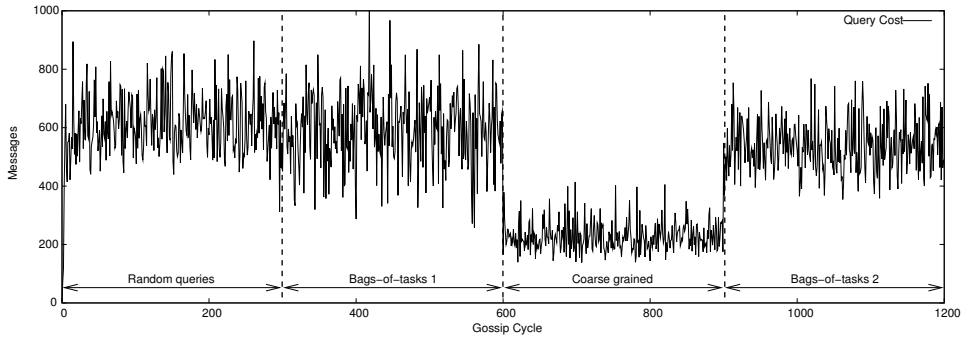
Features to be Tested

Adaptation to changes in query workloads.

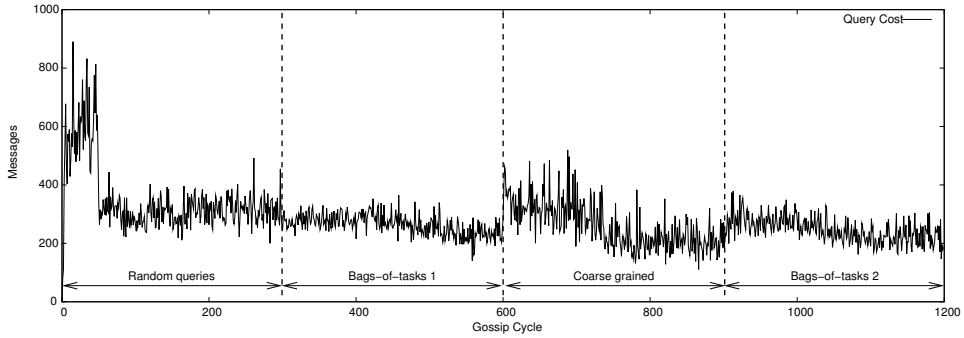
Approach Refinements

We now evaluate RSS adaptation to variations in the query workloads it receives. We simulated 10,000 nodes, with attributes drawn from the 2008 BOINC trace.

We first consider sudden workload changes, by switching the query workload from one type to another at some point of time. We start the experiment with random queries, then switch to bags-of-tasks (where three frequent queries account for 25% of the workload each, and the last 25% of queries are random). We then switch to a coarse-grained workload, and finally another bags-of-tasks workload (similar to the first one, but with a different set of frequent queries).



(a) Without self-adaptation



(b) With self-adaptation

Figure 4.26: Routing overhead for sudden changes in query workloads.

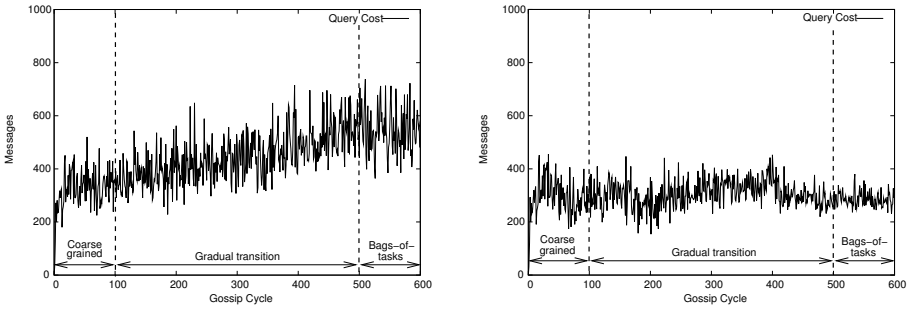
Second, we evaluate the system's behavior for a (more realistic) gradual change of workload. We model a slow transition from the coarse-grained workload to a bag-of-tasks.

4.7.3.3 Test Results

Figure 4.26 shows the performance of the RSS in the adaptive and non-adaptive cases. The non-adaptive system observes no significant cost difference between workloads, except for the coarse-grained workload. This workload can in fact be considered as a best case for the manual configuration of the system, since the query ranges are aligned to the same values as the cell boundaries.

We can observe that here as well the self-adaptation protocol brings a significant cost improvement. When the workload changes at gossip cycle 600 and 900, we see a small cost increase due to the fact that the previous configuration does not work best with the new workload. However, the costs quickly decrease again thanks to self-adaptation. In particular, for the coarse-grained workload, we can see that the self-adaptation algorithm finds a configuration very close to the manually-configured "optimal" one from the non-adaptive system.

Figure 4.27 shows the results of gradually changing the query workload. In the



(a) Without self-adaptation (b) With self-adaptation

Figure 4.27: Routing overhead for a gradual change in the query workload.

first 100 gossip cycles, all the queries submitted to the system are coarse-grained. Then, we introduce bag-of-tasks queries with an increasing frequency besides the coarse-grained queries, until the last 100 gossip cycles when all the queries are bag-of-tasks. At the beginning of the experiment both systems use the same "optimal" set of boundaries so their performance is similar. When the workload starts to change, however, the non-adaptive system sees its costs increase twofold while the adaptive system efficiently controls reconfigurations and maintains a constant performance.

4.7.4 Test Unit 03: Impact of RSS Self-adaptation on Query delivery

4.7.4.1 Responsibilities

This performance evaluation is under the responsibility of VUA within WP3.2 (task T3.2.3).

4.7.4.2 Test Specification

Test Items

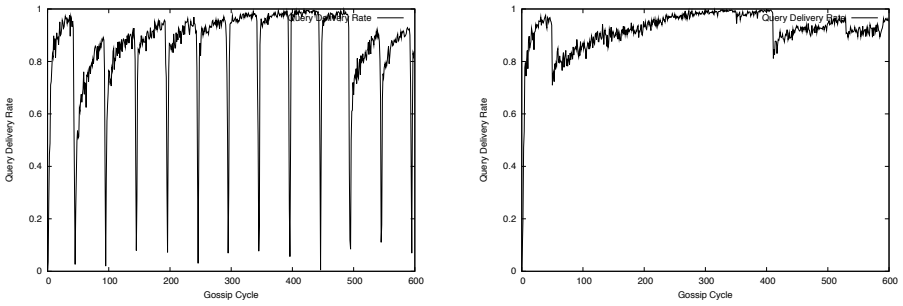
RSS self-adaptation algorithms [39].

Features to be Tested

Impact of RSS self-adaptation on query delivery.

Approach Refinements

We now evaluate the impact of a runtime reconfiguration on the query delivery – that is, the number of nodes found by the RSS divided by the total number of nodes that actually match the query.



(a) (b)

Figure 4.28: Query delivery rate, without (a) and with (b) caching older configurations.

When the system starts, it takes 100 to 200 gossip cycles for each node to build a full set of neighbors. In a system with no churn nor runtime reconfiguration, the query delivery converges to 100%. When a reconfiguration occurs, each node needs to rebuild a new list of neighbors according to the new cell boundaries. However, when the reconfiguration is small, most of the previous neighbors can be reused in the new list. Only very few neighbors need to be found anew.

Reconfigurations have a second type of impact on query delivery: once a query is submitted to the system the routing algorithm assumes that all nodes use a single consistent set of cell boundaries. When a node receives a query that refers to an old set of boundaries that it does not maintain any more, all it can do is terminate the query, leading to poor query delivery.

4.7.4.3 Test Results

Figure 4.28 shows the query delivery during the same experiment as in Figure 4.27: the workload gradually changes from coarse-grained queries to bags-of tasks. We show two cases: one in which each node immediately forgets its previous configuration when it receives a new one, and the case where nodes maintain a read-only cache of recent configurations. When previous configurations are not cached, the system experiences a large drop in query delivery at each adaptation. This is due to the fact that most queries present in the system at the time of reconfiguration will be terminated prematurely due to configuration inconsistencies. Figure 4.28(b) shows that this effect disappears when using the caching policy. In this case, delivery decreases only at the times of major reconfigurations when nodes need to seek for new neighbors. In all cases, even during reconfiguration, delivery remains high, which should remain sufficient for ensuring continuous service of the RSS within the computing grid.

4.7.5 Test Unit 04: Self-Adaptation Cost

4.7.5.1 Responsibilities

This performance evaluation is under the responsibility of VUA within WP3.2 (task T3.2.3).

4.7.5.2 Test Specification

Test Items

RSS self-adaptation algorithms [39].

Features to be Tested

Cost of RSS self-adaptation.

Approach Refinements

An important goal of the adaptation algorithm is to incur only a small cost overhead compared to the system that is optimizing. The most important part of this overhead is the protocol's communication cost, which we estimate next.

4.7.5.3 Test Results

The two main protocol phases that involve communication among nodes are the attribute CDF estimation through the Adam2 protocol and the dissemination of new boundary sets. As shown in [37], an Adam2 aggregation instance with 25 gossip rounds typically requires sending and receiving 40 kB of data per attribute at each node. If we consider a periodicity of one round per second, during the aggregation phase of one attribute distribution each node would need an average upstream bandwidth of 1.6 kB/s for each attribute, and a similar average downstream bandwidth. For an overlay with 4 attributes, as the one used in our tests, the needed bandwidth during aggregation is 12.8 kB/s for each node. The dissemination of new boundary sets has a significantly lower communication overhead. In order to decide whether it is necessary to reconfigure the boundary sets, the nodes periodically exchange their current timestamps of the sets. This information can be added to the regular gossip messages used to maintain the overlay, increasing their size with only 4 B. When a new boundary set is issued, each node receives it only once; for one attribute, the size of the set is normally less than 150 B.

According to our evaluations, 3 or 4 aggregation instances are usually sufficient to generate an accurate distribution approximation. Taking into account the time needed to propagate the new boundary sets after they are calculated, it takes from 100 to 200 gossip cycles to effectively reduce the routing overhead after a change in the system. If we start a gossip cycle per second, the system will be properly

reconfigured within less than 200 seconds. In case such a fast reconfiguration is not necessary, gossip cycles can be initiated less frequently, resulting in a lower bandwidth consumption at the nodes.

4.7.6 Test Summary Report

4.7.6.1 Summary of Tests and Results

These tests demonstrate that the self-adaptation algorithms can effectively adjust internal RSS parameters to maintain optimal performance across a wide range of fluctuations in node properties and query workloads. Reconfigurations have a limited impact on query delivery. The overall protocol is very inexpensive in terms of memory and bandwidth requirements.

4.8 Evaluation of Application Execution Management

Application Execution Management (AEM), created mainly by WP3.3, is the responsible of execute and manage jobs. AEM also interacts with resource nodes (via SSRS) obtaining a set of resources to execute jobs.

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 3.0.
- 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 replica_duplication branch in SVN of XtreemFS SVN rev: 1831

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 measure the scalability of AEM and present some improvements using cooperation between XtreemFS and AEM to schedule jobs near files.

4.8.1.4 Features not to be Tested

Replication with SchedFS will not be tested due to bugs present on XtreemFS software.

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 have performed scalability tests with the hardware available to us (GRID5K). The scalability test involved a hundred of nodes running the last version of AEM that will be in XtreemOS 3.0. The scalability includes, submission and job status check. We also measure the benefits of a collaboration between location service in XtreemFS (Vivaldi) and scheduling of jobs in AEM.

4.8.2 Test Unit 01: SPECweb2005

4.8.2.1 Responsibilities

WP4.2, J. Oriol Fitó from Barcelona Supercomputing Center (BSC). The tests included here are partially submitted in [29].

4.8.2.2 Test Specification

Test Items

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

- Submission of jobs
- Job monitoring
- Resources reservations

Features to be Tested
The main goal is to test and evaluate the AEM component of XtreemOS. Actually, the focus of this evaluation is on the following features: job submission, monitoring and reservations of resources.

Approach Refinements
As stated before, we decided to use *job submission*, *job monitoring* and *job reservation* features in order to evaluate the AEM component. We deployed the SPECweb2005 application [3] with the aim to validate these features. Notice that we decided not to change to the newest version, that is SPECweb2009 [4], because it is equal to the previous version except the inclusion of the power workload. The rest of the workloads of this software 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 4.29).

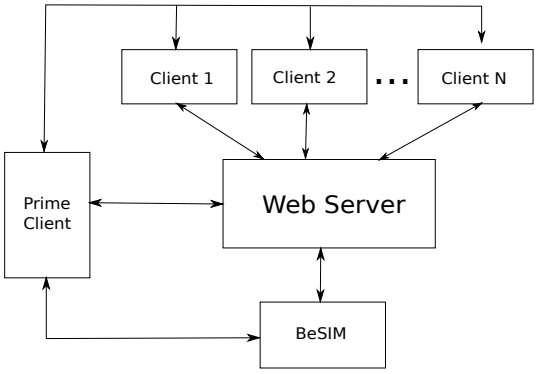


Figure 4.29: SPECweb2005 benchmark architecture.

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.

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 needed, 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 repeatedly 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, two, three and four CPUs allocated to the resource reservation of the web server requested to the AEM. Through these experiments we will be able to see how the performance of the web server scales up when we provide to it an increasing number of resources.

Finally, we checked 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 show the web server's performance high-level metrics from the benchmark results: throughput (requests per second), and response time (seconds).

4.8.2.3 Test Results

In this case, the testbed used for this evaluation is Grid5000, in which XtreemOS images are available to be used easily. In comparison with the results shown in the past deliverable D4.2.6, this highly-distributed environment allows us to contribute with more interesting results in terms of scalability. For our purposes, we installed and configured the SPECweb2005's components needed for its execution. It is noteworthy that the nodes used are of a single cluster, that is the *Griffon* one from the Nancy site.

In particular, the characteristics of each node of this cluster used are the following:

- CPU: Intel Xeon L5420, 2.5 Ghz
- Memory: 16 GB
- Network: Gigabit Ethernet
- Storage: 320 GB / SATA II

In order to evaluate the AEM component of XtreemOS, we carried out several executions of the SPECweb2005 benchmark [3] 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 in terms of web server's performance. These tests were performed by 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 the testing of this type of scalability by means of AEM reservations. In order to obtain these results, we used a range of one to seven client machines (each one simulating a maximum of 1000 simultaneous user sessions), acquired through the resource reservations as needed. In particular, we expose herein the results obtained by performing tests with a maximum of 6700 user sessions. On the contrary, in D4.2.6 we tested only until 2000 user sessions. In this sense, the use of Grid5000 allows us to perform more scalable tests for this deliverable. The possibility to reserve efficiently as many resources as wanted is a remarkable strength of XtreemOS and, in particular, of AEM. In addition, we present the vertical scalability results, in terms of the performance (i.e. throughput and response time metrics) offered by the web server running on top of XtreemOS.

Web server's performance metrics Figures 4.30 and 4.31 illustrate the results regarding the performance of the web server. Note that all of these results are expressed depending on the number of simultaneous user sessions emulated by the clients of the benchmark, and the workload used is the E-commerce one [7].

As shown in these figures, the performance of the server increases as long as the amount of input load is greater. However, when the server has not enough resources to serve a given number of simultaneous clients, its performance offered meets diminished. This fact is easily seen in both the throughput and the response time offered. For instance, when the server has 2 processor units, its response time is more or less the same when the amount of users is less than 2200; and increases linearly when there are more users than this maximum supported.

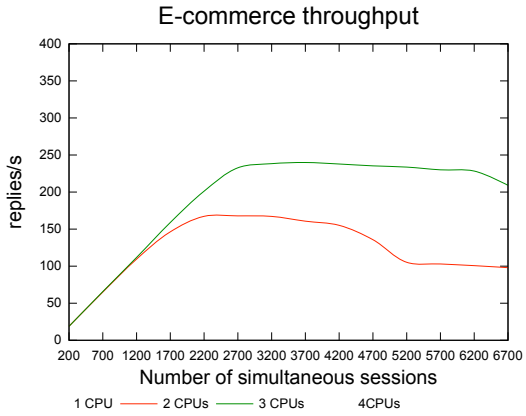


Figure 4.30: e-Commerce throughput (requests per second) when running with one to four processors.

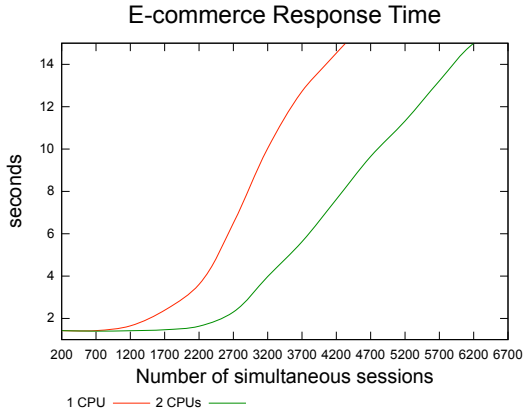


Figure 4.31: e-Commerce response time (seconds) when running with one to four processors.

As a result, 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 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 aforesaid distributed operating system. Therefore, by 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 web applications deployed.

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

4.8.3 Test Unit 02: Job Submisssion

4.8.3.1 Responsibilities

The test in this unit will be performed within WP4.2 and is the responsibility of XLAB.

4.8.3.2 Test Specification

Test Items

We will be testing AEM through XOSAGA. This test also depend on XtreemOS and VO support, although these are not explicitly being tested.

Features to be Tested

The following features of XOSAGA/AEM will be tested:

- job specification: specifying the executable, the command-line arguments, input and output file,
- job submission,
- checking job state in order to wait for the calculating jobs to finish.

Approach Refinements

The goal of this test is to verify whether the jobs are submitted and ran successfully, whether they are scheduled to available resource nodes adequately, and whether the overhead is acceptable. This is a user-level test, thus the latest stable release of XtreemOS will be used and the configuration will be default.

The test application will be Galeb, which submits as many calculation jobs as the user requests, waits for them to finish, reads their partial results from XtreemFS files, and selects the optimal subresult.

We will use a testbed consisting of three virtual machines, two of which act as resources. The host will be based on a 3.33 GHz Intel Core 2 Duo with 8 GB RAM, which is sufficiently powerful that the two single-core resource nodes will not compete for host's memory or CPU time. The host machine will not be used for anything else during testing.

On the software side, all VMs will use XtreemOS 2.1.2 with all updates available at the time of testing. The host will use Ubuntu 10.04 and VirtualBox OSE 3.1.6.

Each run of Galeb/SAGA will use two calculating slave processes. Ideally, one should run on each resource node. The serial version of Galeb will also be timed in order to calculate the overhead of SAGA and AEM. To ensure a similar CPU payload in both serial and parallel (SAGA) tests, the serial version will always be run on both resources concurrently, although just one of the two instances will be timed. As Galeb uses a heuristic search (genetic algorithm), the seed of the random number generator must be the same for all tests.

Two problem sizes will be tested: 500 generations of the genetic algoritm with population size of 200, and 3000 generations with population size 500.

In all cases we will assess the correctness of the result (the result of Galeb/SAGA must be the same as that of the serial version if we use the same random generator seed) and the overhead of Galeb/SAGA over serial version.

4.8.3.3 Test Results

The tests were executed on 2010-09-29 by Marjan Šterk, XLAB. On the first try the application failed because each job was reported as finished (SAGA job state 'DONE') as soon as it was submitted, which resulted in the master process trying to parse an empty file instead of waiting for the slave jobs to fill it with their results (submitted bug no. 293, which is supposed to be solved in release 3.0). The command `xps -a` also reported these jobs as finished, thus the error was in AEM and not in XOSAGA. After restarting `xosd` on all nodes this error did not occur again.

The job scheduler did not always perform as expected – on the contrary, often both slave jobs were submitted to the same node. It looks to be using random scheduling algorithm, which may be perfectly adequate for a heavily used system, but in our case it was suboptimal. In release 3.0 of XtreemOS the user will have the option of specifying the sceduling policy.

Table 4.6: Timings and overhead of Galeb/SAGA

Galeb version	avg. runtime [s]	std. deviation	avg. overhead
Small test case (500 generations, pop. size 200)			
serial	36.4	0.8	
SAGA - 1 or 2 nodes	55.6	14.9	53 %
SAGA - 2 nodes	44.2	1.1	21 %
Large test case (3000 generations, pop. size 500)			
serial	454	19	
SAGA - 1 or 2 nodes	675	312	49 %
SAGA - 2 nodes	517	53	14 %

Each test was repeated 12 times and the highest and lowest measured runtimes were discarded. Table 4.6 gives the timings of the remaining 10 runs for both serial and SAGA-parallelized version of Galeb. The SAGA version does twice the work of the serial one and also has has twice as many, i.e. two, resource nodes at its disposal. As said above, the sceduler may or may not choose to use both nodes, thus the average overhead is quite high. The line "SAGA - 2 nodes" gives the statistics of just those tests that did use both nodes, in which case the overhead is much lower but still significant.

The results for the larger test case are similar. The high overhead in the default case is again caused by the scheduling algorithm that is suboptimal when the system load is low. The overhead in the 2-node case is smaller than in the first test case because of the larger amount of computation in each job. The standard deviation is higher than with serial version, which we attribute to the effects of running the tests in virtual machines.

4.8.4 Test Unit 03: AEM Job Submission scalability

The tests included here are partially submitted in [29].

4.8.4.1 Responsibilities

WP3.3, Ramon Nou (BSC)

4.8.4.2 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. AEM has DHTs disabled to avoid interferences.

Features to be Tested

We test the following features of the AEM interface:

1. Scalability of single job submission with n processes.
2. Scalability of Job status query (with a Job with n processes).

Approach Refinements

The evaluation of the features enumerated will be performed using the GRID5K infrastructure with the AEM that will be shipped in XtreemOS 3.0.

The test will use synthetic benchmarks.

For the first feature, test presented and explained in Figure 4.32 and with results in 4.33 , compares the scalability on the submission of a job with n processes (sequentally and in parallel) in n resources, this test uses the automatic reservation mechanism. In these results we have the resource discovery and reservation overhead for the 100 nodes, plus XtreemFS mounting. We repeat 100 times each point and present the results with a 95% confidence interval. Between each submission we wait 10 seconds to cleanup. When submitting a process, a suitable node from the reservation must be found and selected. We use a random scheduler but others, such as Round Robin or Least Used Resource, are also implemented.

The last feature uses a submission of a very large job (sleep) that will run during the whole test to measure the scalability of asking for the job and process status of a job (master process) and its $n-1$ processes distributed along n nodes. In the second test, the user request goes to the JobMng for the whole job status and through all the nodes in the system (ExecMng) for the process status. The measurements are done in 30 series of 100 sequential job status queries for each point. Results are presented with a c.i. of 95% in 4.34.

4.8.4.3 Test Results

The environment used is setup formed by 100 heterogeneous nodes where we deploy one XtreemOS core and 99 resource nodes. We use Grid5K infrastructure.

In Figure 4.33, we show the scalability results in the second environment. This test uses the automatic reservation mechanism. In these results we have the resource discovery and reservation overhead for the 100 nodes, plus XtreemFS mounting. When submitting a process, a suitable node from the reservation must be found and selected. We use a random scheduler but others, such as Round Robin or Least Used Resource, are also implemented. The lower line shows the results of the same test submitting the $n-1$ processes in parallel in an eight core XtreemOS node for reference. In summary, the cost of sequential job submission is $0.0062x^2$ for a random scheduler in the actual scenario (where x is the number of processes created). This x^2 constant cost will be reduced with some optimizations inside the code such as reducing credentials checking using Single Sign On

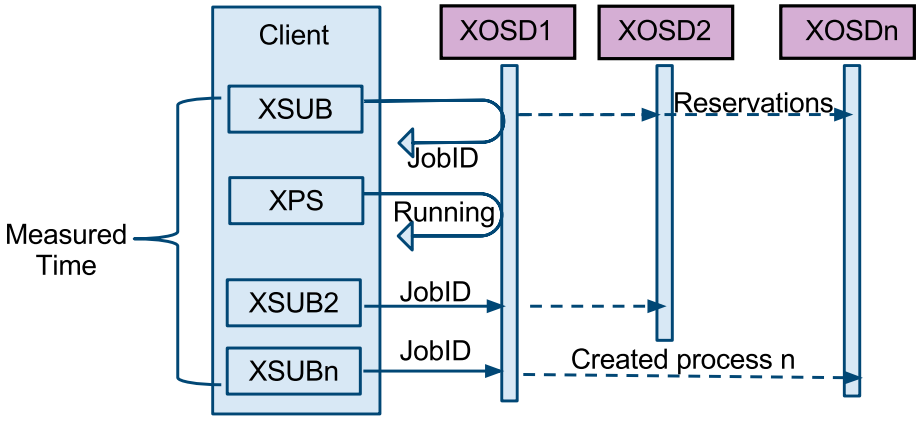


Figure 4.32: Job submission scalability test diagram.

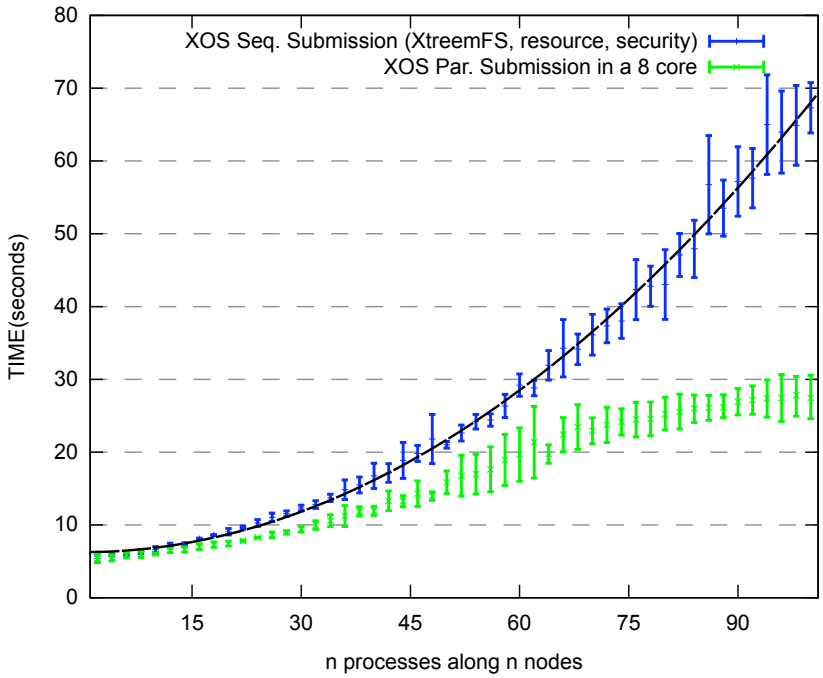


Figure 4.33: Job Submission scalability (Parallel - Sequential).

technology. However it will be difficult to reduce the x^2 bound without reducing features. This cost is produced mainly by the resource timetable checking, as resources and reservations are dynamic and cannot be centralized for scalability reasons. Nevertheless, submitting jobs without processes reduces those checks and lowers x^2 bound. Scalability in the x^2 scenario is obtained distributing the jobs load between different jobMng, for example a VO can have his own JobManager and still have a Global view of the system via the Job Directory (DHT). The cost

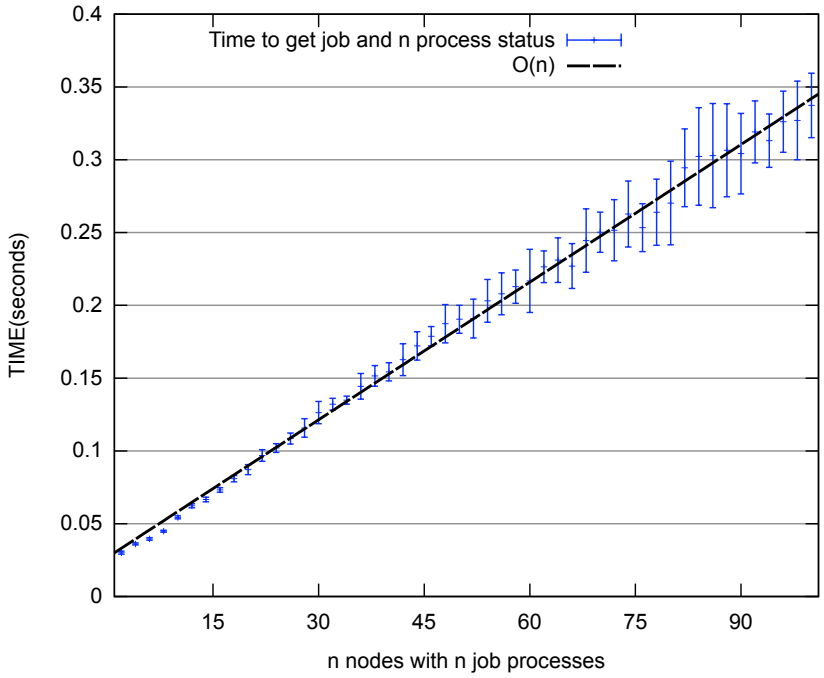


Figure 4.34: Scalability of a status query.

is based on local network times, and is bounded by its latencies. To reproduce a similar scenario in Globus we would need a Job queueing system like Condor and multijobs (GT5 capability). But XtreemOS provides the Job-Process concept relating job to their processes, and Globus considers them different jobs. However we included Globus in the results subsection when submitting a single job.

Figure 4.34 shows the time of getting the process status as the number of processes increases. In Globus we would have to make one job status query to each node with a different JobID. Additionally, in XtreemOS, as Execution Manager requests are done in parallel, the time to ask all involved Execution Manager, in order to gather results from each individual process, is reduced. This is why the time is lower than $O(n)$. Checking job status in a loaded system does not imply an overhead as far as we distribute the load between different Job Managers. The result does not depend on the number of jobs running in the system but only on the number of nodes used for each job. It's worth noting that we can also ask only for job status. To do so, we select a smaller metrics set. Removing the communication with Execution Manager, the line is constant and independent on the number of processes.

The different tests done, and the real use of the system during the implementation phase, marks a tendency that scalability will be as good as was designed for. More precisely the decentralized design of job metrics inside JobMng and processes of a job unit inside its ExecMng node, decouples information provid-

ing a higher throughput. Only jobs with a higher number of processes, distributed among a large number of nodes, may be affected. When this situation is produced, the user can reduce the depth of the information and obtain job status without process status; this will cut the utilization from n nodes (ExecMng) to 1 (JobMng) and get faster if the job is running or not. This decentralization is followed in every layer of the AEM, avoiding scalability problems.

4.8.5 Test Unit 04: AEM SchedFS Benefits

4.8.5.1 Responsibilities

WP3.3, Ramon Nou (BSC)

4.8.5.2 Test Specification

Test Items

For this test we used last replica_duplication branch in SVN of XtreemFS SVN rev: 1831.

Features to be Tested

We are going to test the benefits of using information from XtreemFS file location (via Vivaldi coordinates) with the AEM scheduler. AEM will select the nearest node to the data that the job is going to use (specified via jsdl:vvd extension by the user). We will present the results with SchedFS activated and deactivated.

Approach Refinements

Our approach is use 40 Grid5K heterogeneous nodes with simulated network latencies between each pair. Those simulated latencies are dividing the nodes in 8 clusters of 5 nodes with latencies ranging from 10 ms to 500 ms. Each test has the same latencies scenario. XtreemFS vivaldi system is stabilized by more than 24 hours, and we start from the same stabilization point. We have a setup of 40 OSD, 1 MRC and 1 DIR service.

In this scenario we create 10 files of 100Mb, that are distributed over 10 OSD of the 40 available. Finally, we have a set of 100 jobs that are using 1,2,3 or 4 files of those 10. The JSDL of the jobs are generated in a way that we try to create clusters of files. The measured time is the time to have 100 reads of 1K along all the files used in the job.

We execute jobs 1 by one, sequentally with a cleanup time of 10 seconds between job.

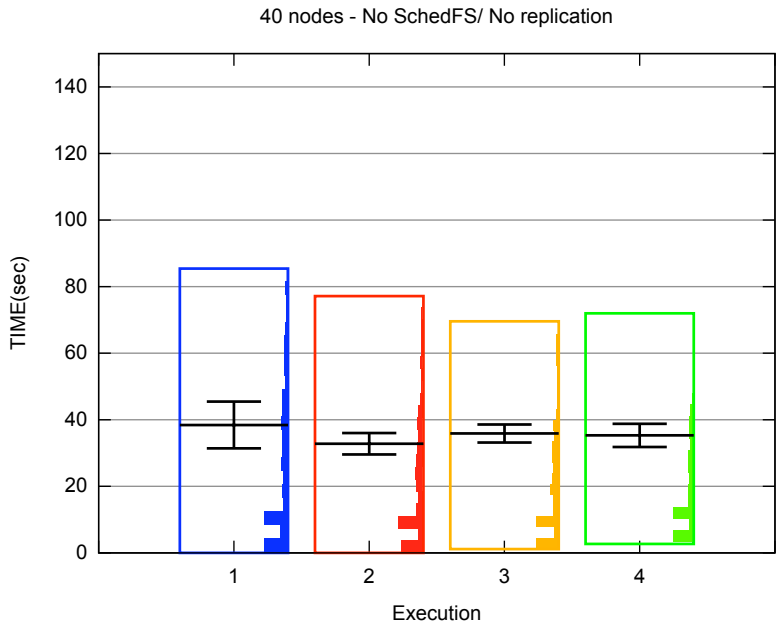


Figure 4.35: Job Execution (Job with 1,2,3 or 4 files) without SchedFS.

4.8.5.3 Test Results

Figure 4.35 shows the results without SchedFS, Figure 4.36 shows the result with SchedFS (without replication). X-axis are the kind of jobs (reading 1,2,3 or 4 files), Y-Axis is the time to do it. Boxes represent maximum-minimum values, black line is the mean with a 95% c.i, and finally inside the boxes we have the distribution of the times using an histogram (%).

Replication in XtreemFS has been left out of this evaluation as it has a bug and do not work.

As the figures show, collaboration with XtreemFS via the AEM scheduler to execute and distribute jobs produces greatly benefits. However, increased performance could be obtained if replication is used together with SchedFS. We can reduce from a mean of 40 seconds to less than 15 seconds. In some cases, with one file we execute always with 0 latency from the network as the correct node (if available) is always selected.

4.8.6 Test Summary Report

4.8.6.1 Summary of Tests and Results

We performed tests to validate AEM functional requirements, evaluate its performance and scalability. Results of this part are published in [29]

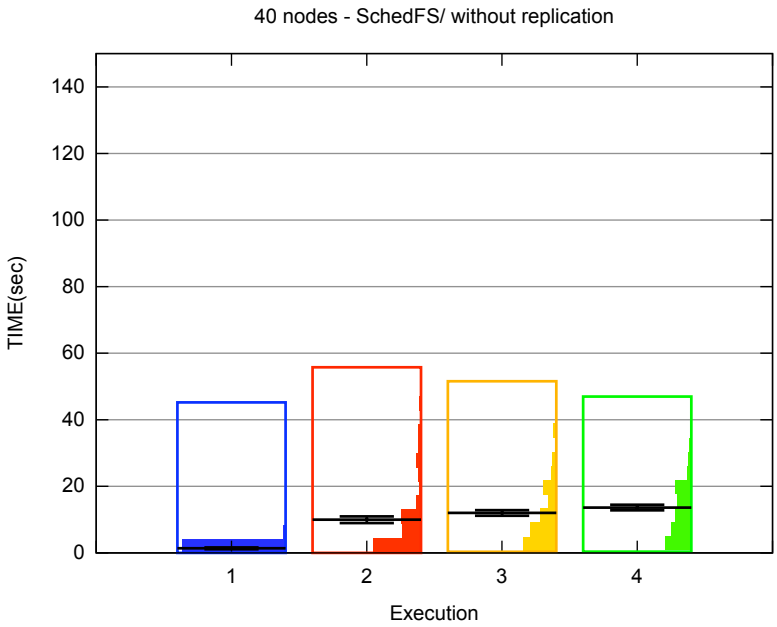


Figure 4.36: Job Execution (Job with 1,2,3 or 4 files) with SchedFS.

4.8.6.2 Conclusion and Directions for Future Work

AEM in XtreemOS has obtained good results on scalibility terms, using only one job manager and a hundred resource nodes. Scalability is also mantained when asking for job status information. Performance in this part is better than Globus. This shows the benefits of having interaction with the kernel and our architecture. XtreemFS collaboration with SchedFS has not been completed due to a notified bug in replication code.

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.

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:

- UDUS: An interactive multi-user 3D virtual world application for testing OSS,
- UDUS: A distributed microbenchmark application for performance stress test of OSS,
- UDUS: A distributed word frequency analysis application using the MapReduce computing model for OSS testing,
- CNR: The NTFS-3G suite for testing XtreemFS POSIX-compliance,
- SAP: MaxDB business application for testing scalability, stability and performance of XtreemFS,
- SAP: MaxDB business application for comparative performance analysis of several distributed file systems,

- SAP: MaxDB business application for testing XtreemFS replication for fault tolerance and performance.

4.9.1.2 Test Items

The test items include:

- The XtreemFS version tested are the releases 1.1 and 1.2.1 . Source and documentation are available from the internal XtreemOS SVN. Publications are available on the XtreemFS website: www.xtreemfs.org.
- Object Sharing Service provided by XtreemOS release 2.0.
- Wissenheim as provided by XtreemOS release 2.0
- The MaxDB distribution and its documentation is available under https://www.sdn.sap.com/irj/sdn/maxdb.
- Information about Wissenheim can be found under: http://www.wissenheim.de.
- The Pawel Jakub Dawidek's 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/.

4.9.1.3 Features to be Tested

The following features will be tested:

- Scalability of OSS with up to 128 nodes.
- Data consistency of OSS during high contention.
- Stability regarding full meshed p2p-connection handling with up to 128 nodes.
- Performance of the Distributed Transactional Memory (DTM) when operating with dynamic access patterns.
- Performance optimization of OSS through dynamic adaption of object access granularity during runtime.
- POSIX compliance of XtreemFS (open, read, write, close, ls, rm, touch, mv, cp, mkdir, cd, rmdir)
- Stability of XtreemFS under large number of clients, varying IO load and the number of OSD servers of XtreemFS.

- Performance of XtreemFS under large number of clients, raising IO load and varying number of OSD servers of XtreemFS.
- Performance benefits and fault tolerance that XtreemFS provides for the consuming business applications thanks to the replication feature of XtreemFS.

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 reasons pointed out in the respective test units:

- OSS basic network setup.
- OSS distributed transactional memory.
- Fault tolerance and performance benefits of using write replication will not be tested since the current official release supports read-only replication.

4.9.1.5 Overall Approach

The performance of XtreemFS and OSS will be evaluated using applications and benchmarks. The tests will also provide comparative performance analysis of XtreemFS, CEPH 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 POSIX compliance of XtreemFS will be tested at Test Unit 4.9.6 using the Pawel Jakub Dawidek's POSIX filesystem test suite (PJD-fstest).
Application Focus We will test XtreemFS using the MaxDB application. The tests are performed by installing XtreemFS on testbeds consisting of local machines and executing the MaxDB replay and Wissenheim (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 Units 4.9.6, 4.9.7 and 4.9.8.

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 Unit 4.9.2. Whereas the SAP experiments test a database-centric engine scenario, the experiments with Wissenheim test are file-based.

4.9.2 Test Unit 01: Object Sharing Service (OSS) – Wissenheim

4.9.2.1 Responsibilities

WP4.2, Michael Sonnenfroh (UDUS)

4.9.2.2 Test Specification

Test Items
Test item is 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 multi-user virtual-world among all participating nodes. Concurrent data accesses from different nodes are synchronized by using speculative transactions provided by OSS.

Measuring the performance of an distributed interactive application is not a trivial task. The most common value used is the *graphics frame rate* which measures how often the screen can be rendered per second. This frame rate incorporates all elements of the application but depends heavily on the computational power of the used graphics adapter. Another frame rate used is the so called *game frame rate* which determines how often the game state is updated per seconds, by running physics and/or animation calculations and incorporating changes of remote peers. For the following measurement we are addressing the game frame rate only whenever we refer to the frame rate.

To avoid the performance effects of the render engine we are running the tests without OpenGL output and simulate the user-interaction-load synthetically. The physics and animation calculations are reduced to a minimum. In general, games aim at achieving a frame rate of about 60 fps but for our measurements we have chosen a loadfree frame-rate of 200 fps to increase the resolution of the measurement results. Furthermore, we are mainly interested in the average frame rate with respect to the number of participating nodes and the standard deviation of the frame rate. Like for video and audio streaming the deviation is disturbing and therefore an important factor.

Wissenheim implements two different ways to use transactions for sharing the game state. In the first one, called the *naive approach* Wissenheim is using transactions for all phases of the application. This includes operations like the rendering or the physics calculations which are accessing the distributed game state in a read

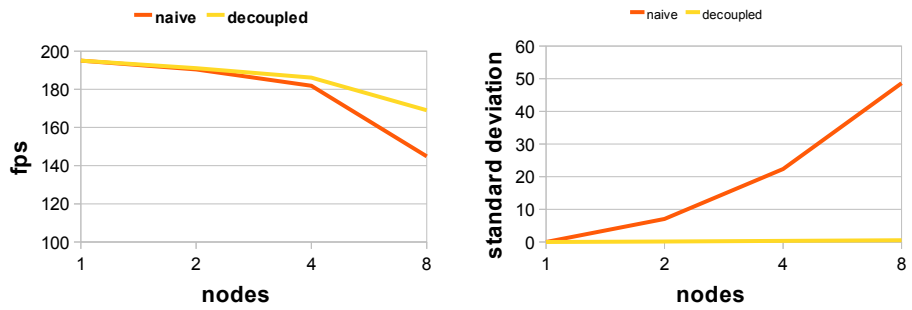
only manner. This means, that the physic phase gets aborted if any other node is e.g. altering the movement of its avatar.

The *decoupled approach* optimized the first one by allowing transactions of the formerly mentioned read only phases to finish despite conflicts. This might result in temporary inconsistencies like an avatar being an inch off its real position but these errors will be undone during the next run of the phase and won't be noticed by the user.

All tests have been performed on a private AMD Opteron cluster (16 nodes). Each node has two CPUs (singe core) running at 1.8 GHz, 2 GB RAM, and all 8 nodes used are interconnected by a Gigabit Ethernet network.

4.9.2.3 Test Results

The frame rate results of the measurements are shown in figure 4.37(a) and the deviation of the frame rate is shown in figure 4.37(b). Additionally, we have measured an average token request time of around 250μs. The token is a special network packet used by OSS to serialize transaction commits of participating nodes. With eight nodes the naive approach needed an average bandwidth of 88 KB/s per node and the decoupled 112 KB/s.



(a) Frame Rate.

(b) Deviation.

Figure 4.37: Wissenheim using OSS.

As expected the naive solution with OSS worked for up to four nodes but scaled very badly. The high conflict rate lead to a very high deviation making it nearly impossible to play Wissenheim. The decoupled approach with OSS scaled significantly better and showed very promising results.

4.9.3 Test Unit 02: Object Sharing Service (OSS) – Performance stress test

4.9.3.1 Responsibilities

WP3.4, Marc-Florian Müller (UDUS), Kim-Thomas Rehmann (UDUS)

4.9.3.2 Test Specification

Test Items

Test items are the Object Sharing Service and a distributed microbenchmark application.

Features to be Tested

The following features have been tested:

- Scalability with up to 128 nodes
- Data consistency during high contention
- Stability regarding full meshed p2p-connection handling with up to 128 nodes

Approach Refinements

In this test the Object Sharing Service (OSS) shares objects of a microbenchmark application by using its distributed transactional memory. OSS transparently synchronizes the objects in the background. The microbenchmark application uses OSS for testing the scalability under different transaction loads and network constraints. Tests have been performed with a best and a worst case scenario. Each increment operation is encapsulated in one transaction.

This test shows the performance measurements of the transactional memory whereas the focus relies on conflict and network related penalties for a best and worst case scenario. Therefore all optimizations (local commits, linked transactions, consistency domains etc.) have been turned off. In the worst case scenario all peers concurrently increment a common variable stored in the transactional memory, in the best case scenario each peer increments its own variable. Furthermore, we have expanded the transaction duration and pause between successive transactions to simulate real live applications. For the measurements we have used our P2P commit protocol with two different token mechanisms for transaction serialization. The token was passed among the peers either by a dedicated coordinator or P2P based approach.

We have used a heterogeneous test environment with 128 nodes (each containing two AMD Opteron 246/250 processors with 2.0/2.4 GHz) from two clusters of the Grid'5000 platform. The nodes are interconnected via a switched 10 Gigabit Myrinet and 1 Gigabit Ethernet network. All nodes are running under a Linux 64-bit operating system with kernel version 2.6.26.

A detailed description about the tests and how to achieve an efficient ordering of speculative transactions in distributed systems can be found in the paper *Efficient Commit Ordering of Speculative Transactions* [27]

4.9.3.3 Test Results

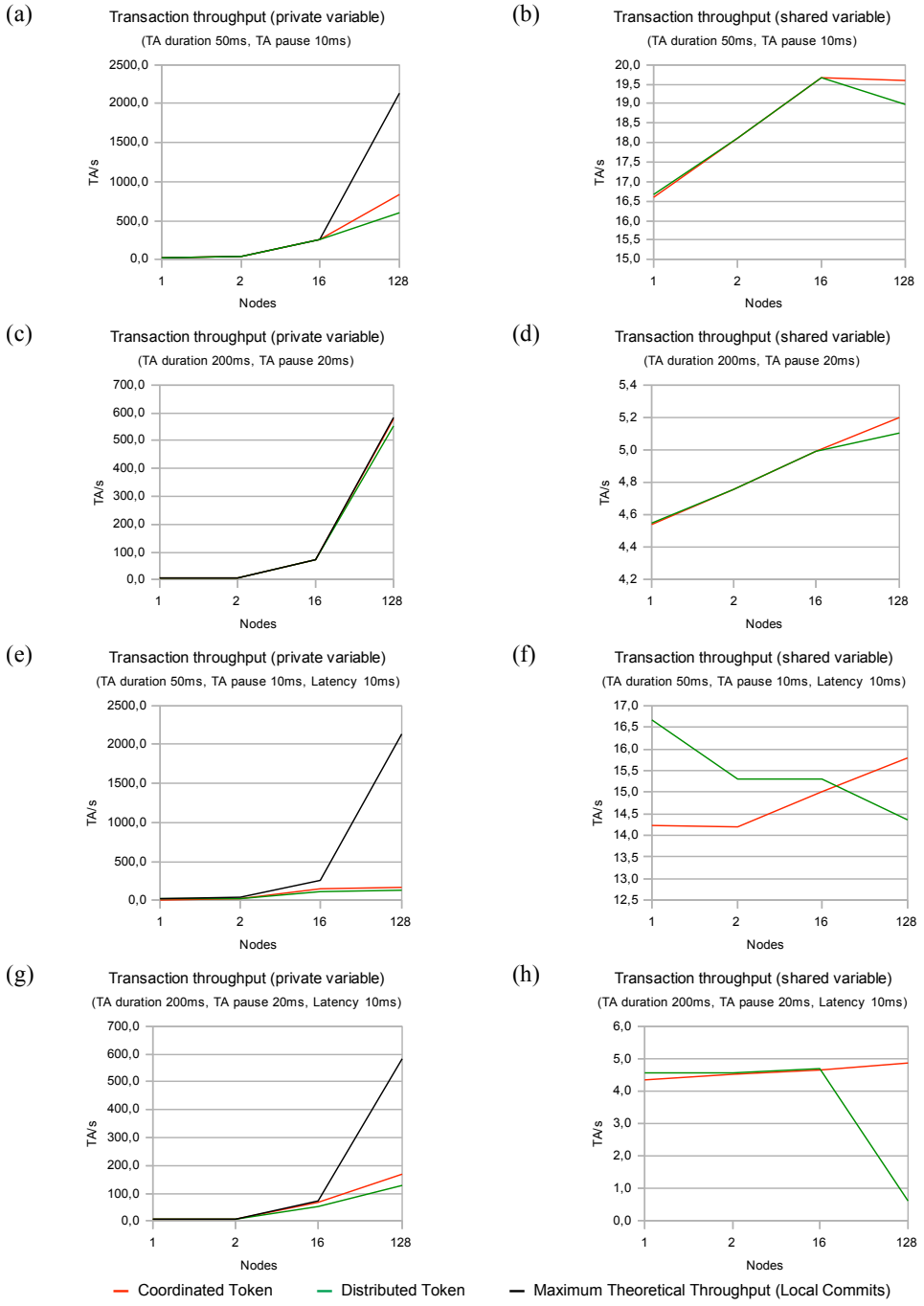


Figure 4.38: OSS performance measurements with high and low contention

The charts in Figure 4.38 show performance measurements of the distributed transactional memory. The results show the conflict and network related penalties for the best and worst case scenario. The best case scenario (private variable) solely shows the network overhead produced by the commit protocol. The worst case additionally causes penalties due to a high conflict rate which results in transaction aborts and restarts. The red and green lines show the overall transaction throughput by using the coordinated and p2p based token passing mechanism. The black line shows the maximum theoretical throughput based on the transaction time and pause, presumed no conflicts occur.

We have examined different transaction workloads by using different transaction runtime lengths with a sleep after incrementing the variable (sleep called within the transaction). Furthermore, we examined different pauses between transactions by using a sleep after EOT (end of transaction). Thus we can simulate different transaction patterns as they occur in real applications.

Figure 4.38 (a, b) shows the transaction throughput for the shared and private variable scenario for short transactions (50 ms execution time) and a short pause (10 ms). As expected network commits are expensive and the central approach scales better because it avoids token races. But it is remarkable that the throughput increases with the increasing number of nodes (for conflict-free transactions). The maximum theoretical throughput (shown in the left figure) is reached by local commits.

The next Figure 4.38 (c, d) shows the same scenario for longer transactions (200 ms) and a bit longer pauses (20 ms). Here we see more or less the same except that the overall pressure on the transaction system is lower.

The following Figures 4.38 (e, f) and 4.38 (g, h) show the same scenarios again but with a synthetic network latency of 10 ms (for all connections). Obviously, network latency is crucial for transaction processing and transaction throughput drops. Although, the maximum theoretical throughput could be reached using local commits. The incrementation of the shared variable offers contrary results of both token passing algorithms. Due to many transaction aborts, incrementing the shared variable scales bad. This scenario shows the lower boundary of transaction performance, but we do not expect this access pattern in real applications.

4.9.4 Test Unit 03: Object Sharing Service (OSS) – Word frequency analysis

4.9.4.1 Responsibilities

WP3.4, Marc-Florian Müller (UDUS), Kim-Thomas Rehmann (UDUS)

4.9.4.2 Test Specification

Test Items

Test items are the Object Sharing Service and a distributed word frequency analysis application using the MapReduce computing model.

Features to be Tested

The following features have been tested:

- Performance of the Distributed Transactional Memory (DTM) when operating with dynamic access patterns
- Performance optimization through dynamic adaption of object access granularity during runtime

Approach Refinements

The application performs a word frequency analysis of a given text. Therefore a master node subdivides the text into distinct parts and maps them to the nodes. After the mapping phase each node counts the words of its text part. Afterwards, the reduce phase collects all partial results and sums up all counters of identical words to gather the global result containing the overall word counters.

The word frequency application operates on a tree (26-ary tree referencing 26 child nodes) stored in the DTM, where each child of a node represents one of all alphabetical characters, and a word is represented by a path from the root node to any other node of the tree. Furthermore, each node stores the counter value of the word. A node's associated word is identified by concatenating all characters along the node's path of the tree. The provided fine-granular object access-detection scheme (independent of the hardware's access detection granularity) allows avoiding high transactional abort rates caused by false conflicts (false sharing). Furthermore, OSS adaptively manages conflict unit size by grouping objects together and splitting them up again in case of conflicts.

As a test platform we have used a private cluster with 16 nodes (each containing two AMD Opteron 244 processors with 1.8GHz). The nodes are interconnected via a switched 1 Gigabit Ethernet network. All nodes are running under a Linux 64-bit operating system with kernel version 2.6.26.

A detailed description about adaptivly modifying the object access granularity to avoid false sharing conflicts can be found in the paper *Adaptive Conflict Unit Size for Distributed Optimistic Synchronization* [35].

4.9.4.3 Test Results

Figure 4.39 shows that transactional conflicts depend on the granularity of object access detection. In case of coarse access-detection granularity (multiple objects per virtual memory page) applicable for the *MSpaces* memory allocator, accessing objects can induce false conflicts (false sharing) which results in many unnecessary transaction aborts. Fine-granular access-detection as provided by the *Millipage*

allocator and the adaptive approach are able to resolve this issue without wasting physical memory.

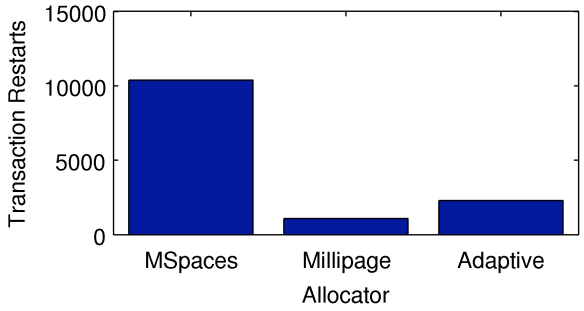


Figure 4.39: Transaction aborts for word frequency analysis

4.9.5 Test Unit 04: XtreemFS POSIX Compliance Tests

4.9.5.1 Responsibilities

This test unit and the included tests on the POSIX compliance are under the responsibility of CNR (WP3.4).

4.9.5.2 Test Specification

Test Items

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

The XtreemFS version tested is the 1.2.1 version, i.e. the most recent release available at this moment. It can be downloaded at http://www.xtreemfs.org/download.php?t=source. Moreover, source and documentation are available also 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 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 experimentally verified that the POSIX-compliant functionality of XtreemFS is invariant with respect to the number of OSD, MRC or DS exploited in the experiments. This is justified by the fact that the POSIX tests query and verify only metadata information, thus its results are affected only by the logic implemented in the MRC (and not by the number of nodes involved in the experiments). 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, we needed to interpret the cause of the problem and reproduce manually the scenario (the sequence of operations) causing it.

4.9.5.3 Test Results

As described above, 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.

Before reporting test results, it is relevant to make two observations. First, XtreemFS does not support FIFOs mechanisms. For this reason, to make our evaluation as much correct as possible, we skipped tests performed by the PJD tests suite on the `mkfifo` system call (totally, 232 tests). Second, XtreemFS supports file names and directory names longer than 256 characters. Since this is considered 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 242 tests from the original PJD suite test set, resulting in 1715 final tests.

Now, let us report the results of POSIX-compliance evaluation on XtreemFS. By considering all the 1715 tests performed by the PJD test suite, actually XtreemFS passes exactly 1559 tests, corresponding to 90.91% of the total. The table 4.7 summarizes a more detailed report of the results obtained in out tests. 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).

4.9.6 Test Unit 05: Scalability, Stability and Performance of XtreemFS

4.9.6.1 Responsibilities

WP4.2, Peter Izsak, Roman Talyansky (SAP)

4.9.6.2 Test Specification

Test Items

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	200	164	82.00%
link	18	167	161	96.41%
mkdir	13	101	101	100.00%
open	24	217	213	98.16%
rename	21	479	382	79.75%
rmdir	16	109	105	96.33%
symlink	13	90	90	100.00%
truncate	14	90	90	100.00%
unlink	14	120	112	93.33%
Total	183	1715	1559	90.91%

Table 4.7: POSIX Test Result

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

The XtreemFS version tested is the release 1.2.1 from 2010-09-08. Source and documentation are available from www.xtreemfs.org.

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

Features to be Tested

The purpose of this test is to test different aspects of XtreemFS in a large scale deployment. The following features and requirements are the subject of this test design specification:

- Stability of XtreemFS under large number of clients, varying IO load and the number of OSD servers of XtreemFS.
- Performance of XtreemFS under large number of clients, raising IO load and varying number of OSD servers of XtreemFS.

Approach Refinements

The goal of the test is to test and evaluate large scale deployment of XtreemFS file system under a transactional load that is generated by a typical business application.

We used Grid'5000 as testbed. A mixture of nodes from the Carri System (paradent) cluster and the Dell (paramount) cluster were used.

Specification of paradent cluster nodes: Carri System CS-5393B nodes; CPU: Intel Xeon L5420; 2.5 Ghz / 6MB; 2 cpus per node; 4 cores per cpu; memory: 32GB; network: Gigabit Ethernet; storage: 320GB / SATA II.

Specification of paramount cluster nodes: Dell PowerEdge 1950 nodes; CPU: Intel Xeon 5148 LV; 2.33 Ghz; 2 cpus per node; 2 cores per cpu; memory: 8GB; network: Gigabit Ethernet; storage: 2x300 GB Raid0 / SATA.

Performing the tests at the same dedicated cluster environment without using virtualization layers, ensures accuracy, reproducibility and fairness of the comparison. We used standard performance metrics: latency and throughput of read and write file system operations.

This test uses recorded IO access traces that the SAP database MaxDB generated while supporting a real-life Sales 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 following, we will refer to concurrently replayed traces as *concurrent IO streams*. During the recording process, MaxDB ran over a commercial 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 file systems 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 find out whether increasing stripe width (and number of OSDs) improves the resulting performance of the system. Each test was performed three times and for each performance metric (throughput and latency) we report the average value of the metric over the three test repetitions.

4.9.6.3 Test Results

Figure 4.40 presents the read latency results for the baseline filer technology and XtreemFS volumes with Stripe Widths (SW) 1, 10, 20 and 30. In our experiments, the number of OSDs equals the SW of each volume. We also strive for collocating the clients with OSD servers to improve the locality of the data accesses. For all SWs a better latency is achieved when moving from 1 IO stream to 4 concurrent IO streams. The explanation comes from the nature of the concurrent IO streams

- to simulate a growing IO load, the same IO trace is replayed concurrently at several client nodes, while the start time of each IO stream is randomly chosen within a certain period of time. In this situation the first IO stream that performs a specific IO access in the trace loads the corresponding file stripe to the cache of the corresponding OSD. When the other IO streams later perform the same access, they find this file stripe already in the cache, avoiding the disk access.

In Figure 4.40 one can see that XtreemFS demonstrates very good resilience to scaling IO load. It is quite interesting to see that the read latency of XtreemFS with 30 OSDs, when the IO load is distributed to so many OSD servers, is indifferent to the growing IO load. Note also the sharp increase in latency for 20 OSDs and 40 streams. It can be explained by the fact that the performance of XtreemFS is dependent on the network loads and spikes in the network loads can cause performance degradation of XtreemFS. Since at the time of running the experiment on the Grid'5000 clusters there were two additional big jobs that ran suimultaniously with our job, they probably generated a network load, which resulted in the XtreemFS performance degradation. We explain this observation by the fact that the accumulated working set of the concurrent IO streams grows linearly 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 linearly with the number of OSDs and that is why this cache is able to incorporate the big working set of several IO streams. In addition to the distributed cache explanation, when more OSDs are available for XtreemFS, the IO accesses of different IO streams are distributed to more OSDs, reducing the probability of access collisions, i.e. when several IO accesses try to access the same OSD simultaneously.

Figure 4.41 shows the "mirror" image of Figure 4.40, where the throughput of XtreemFS increases as function of the distributed cache size and better IO access distribution to growing number of OSDs.

Figure 4.42 shows write latency of the baseline filer technology and XtreemFS volumes with SW 1, 10, 20 and 30. 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 XtreemFS file volumes. As with the latency of the read operations, the latency of write operations of XtreemFS volumes is resilient for the growing number of IO streams. Furthermore, it may be observed that a higher number of OSDs provides with better resilience to the growing number of IO streams, except the case of 1 and 10 OSDs and the number of IO streams beyond 10. Our interpretation of this observation is that with growing number of OSDs, XtreemFS distributes the synchronous write operations among several OSDs and thus effectively load balance the write load and reduces

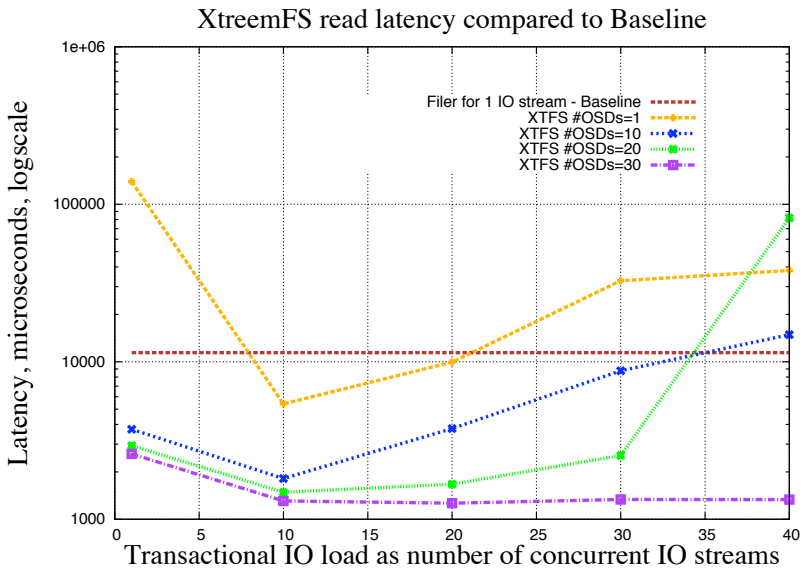


Figure 4.40: Read Latency in MaxDB experiments.

the collisions when several write operations are executed simultaneously.

Figure 4.43 shows write throughput in our experiments. As in the case of read operations, the throughput graphs of write operations provide with the "mirror" image of the write latency. Note, however, that with 20 and 30 OSDs, XtreemFS outperforms the throughput of the baseline filer despite 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 carried out the performance analysis measurements with a large deployment of XtreemFS, which show promising results that support the following conclusions:

- XtreemFS is stable under real-life business application work load.
- XtreemFS scales well facing a growing IO load that is generated by a business application, provided a big enough number of OSD servers.
- XtreemFS effectively utilizes distributed cache whose size grows linearly with stripe width of the volume. This distributed XtreemFS cache is capable of incorporating big application work sets and thus provides the basis for data volume scalability at high performance.

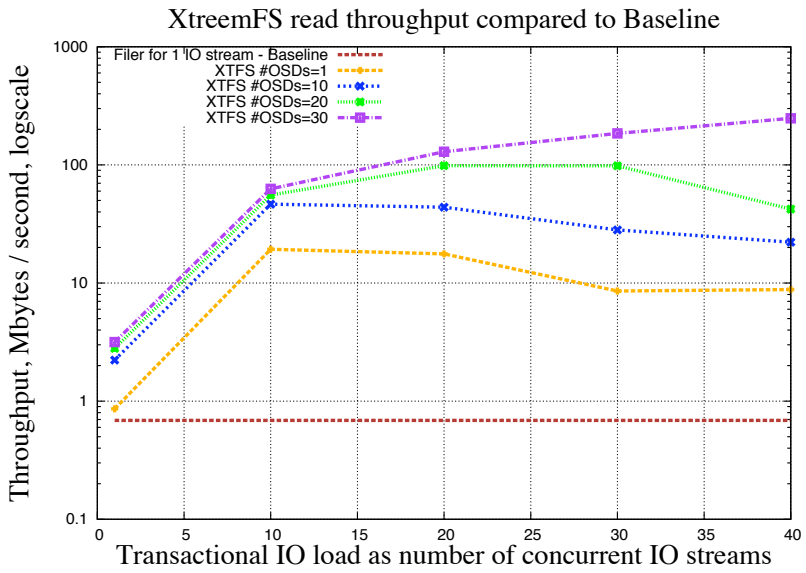


Figure 4.41: Read Throughput in MaxDB experiments.

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

- XtreemFS effectively distributes IO operations over several OSDs enabling lower latencies and higher throughput.

- XtreemFS may help to reduce TCO of running business applications: XtreemFS scales out over commodity hardware - it runs over commodity hardware and in some cases its performance scales almost linearly with stripe width.

- While the write latency of XtreemFS is resilient to the growing IO load, it is still much above the write latency of the filer technology, since the filer used non-volatile memory to speed-up synchronous write operations.

- Overall conclusion: our experiments show the potential of XtreemFS to support transactional load. However more work should be done to improve synchronous write latency, stabilize the replication feature in XtreemFS and further prove its ability to support business application transactional load.

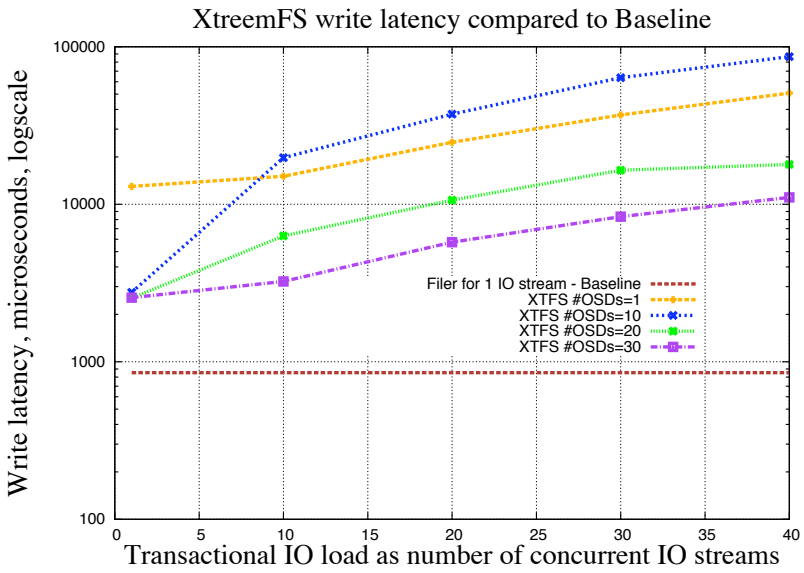


Figure 4.42: Write Latency in MaxDB experiments.

4.9.7 Test Unit 06: XtreemFS, CEPH and NFS - Comparative Performance Analysis

4.9.7.1 Responsibilities

WP4.2, Peter Izsak, Roman Talyansky (SAP)

4.9.7.2 Test Specification

Test Items

The software to be tested is XtreemFS and CEPH.

The XtreemFS version tested is the release 1.1. Source, documentation and publications are available from www.xtreemfs.org.

The CEPH version tested is the release 19.1. Source, documentation and publications are available from http://ceph.newdream.net/.

The NFS version tested is the release 1.1.3-18.17.

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

Features to be Tested

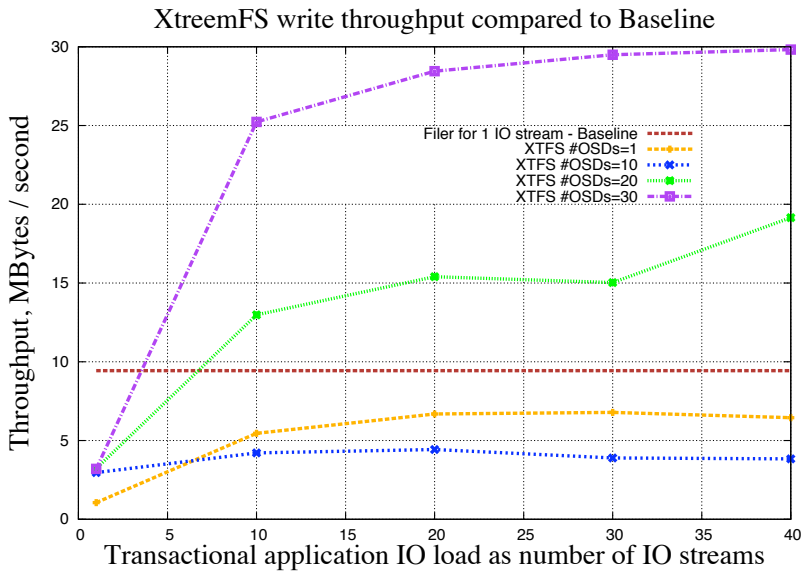


Figure 4.43: Write Throughput in MaxDB experiments.

The purpose of this test is to perform comparative analysis of XtreemFS and CEPH.

Approach Refinements

The goal of the test is to perform comparative analysis of XtreemFS and CEPH under a transactional load that is generated by a typical business application.

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 (Gigabit per second) (E 1Gb/s). Performing the tests at the same dedicated cluster environment without using virtualization layers ensures accuracy, reproducibility and fairness of the comparison.We used standard performance metrics: latency and throughput of read and write file system operations.

This test uses the same IO load as in Test Unit 05. 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.

4.9.7.3 Test Results

Below we provide a comparative performance analysis of XtreemFS and CEPH file systems. Since CEPH is still not stable, we were able to produce a preliminary performance results for it only. In our experiments, the number of OSDs equals the Stripe Widths (SW) of each volume and we used XtreemFS volumes with Stripe Widths (SW) 1, 2, 4, 6, 8, 10 and 12. CEPH volumes had SWs 1, 2, 4, 6, 8 and 10.

Figure 4.44 presents the read latency results for the baseline filer technology, NFS, XtreemFS and CEPH volumes. We can see that for SW=4, both XtreemFS and CEPH have similar performance. For small number of concurrent IO streams CEPH outperforms XtreemFS, while for higher number of concurrent IO streams the opposite holds.Note that NFS and the baseline filer run on a similar hardware and their performance is also similar. We address this good NFS performance to the fact that NFS is a mature file system, that runs for a long time in many enterprise systems. However the performance of XtreemFS with one OSD is slower than of NFS, which may be explained by the young age of XtreemFS without long experience on enterprise workloads. An additional explanation is that the XtreemFS servers are written in Java, which may slow down the performance at the level of one OSD server, and which may not show with more OSDs, due to the distribution of IO load to several OSDs.

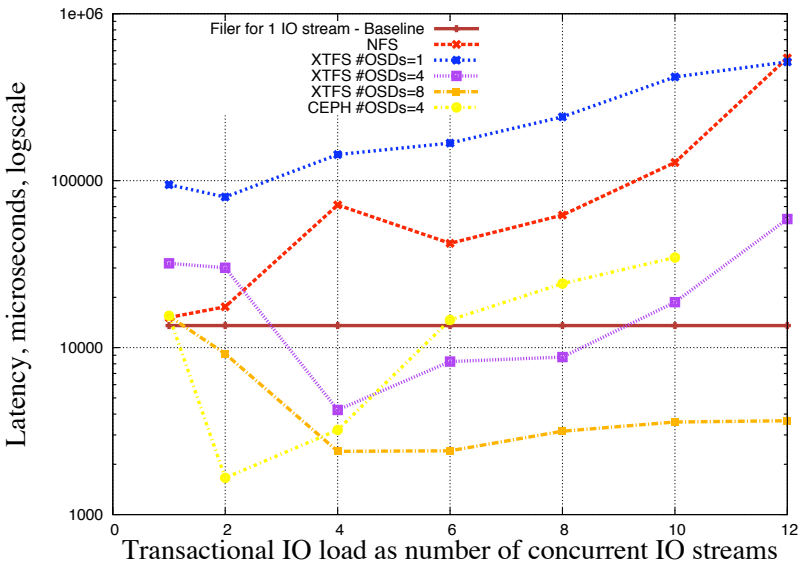


Figure 4.44: Read Latency in MaxDB experiments.

Figure 4.45 shows the “mirror” image of Figure 4.44, where CEPH provides

better results for small number of concurrent IO streams, while XtreemFS for higher number of concurrent IO streams.

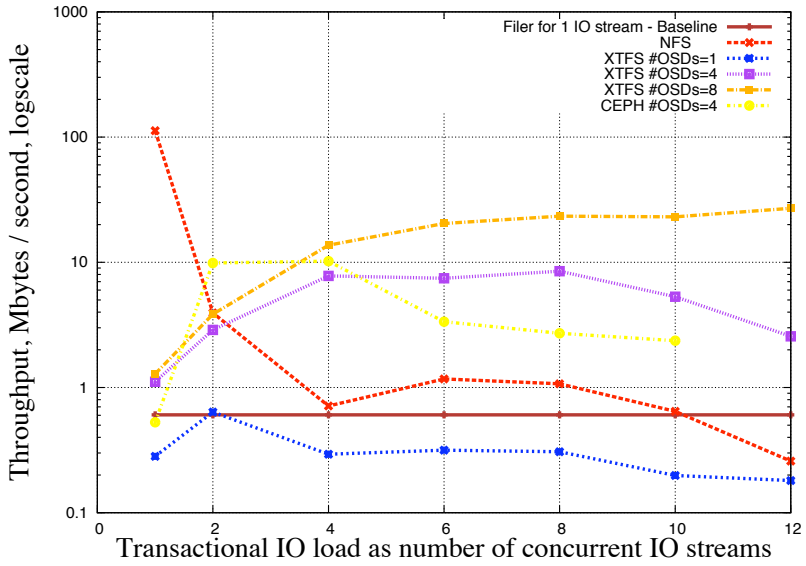


Figure 4.45: Read Throughput in MaxDB experiments.

Figure 4.46 shows write latency of the baseline filer technology, NFS, XtreemFS and CEPH file systems. In general XtreemFS outperforms CEPH, excluding the XtreemFS volume with 1 OSD.

Figure 4.47 shows write throughput in our experiments. As in the case of read operations, XtreemFS outperforms CEPH, excluding the XtreemFS volume with 1 OSD.

Conclusions:

We performed the comparative performance analysis of XtreemFS and CEPH which support the following conclusions:

- More experiments on a large system with hundreds of nodes are required to further compare XtreemFS and CEPH.
- For read operations CEPH outperforms XtreemFS for up to 4 IO streams, while for higher number of IO streams the opposite is true.
- For write operations XtreemFS outperforms CEPH, excluding the case with one OSD server.

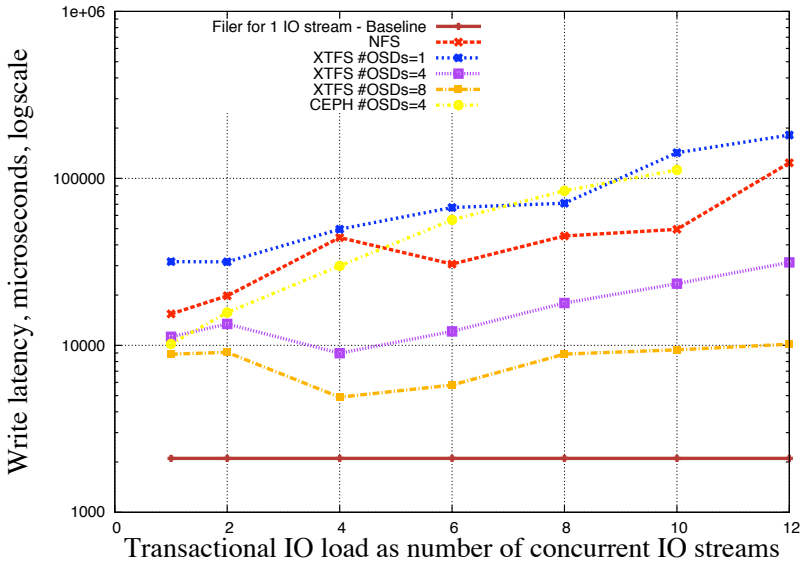


Figure 4.46: Write Latency in MaxDB experiments.

4.9.8 Test Unit 07: XtreemFS replication for fault tolerance and performance, by SAP

4.9.8.1 Responsibilities

WP4.2, Peter Izsak, Roman Talyansky (SAP)

4.9.8.2 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 1.2.1 from 2010-09-08. Source and documentation are available from www.xtreemfs.org.

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

Features to be Tested

The purpose of this test is to test performance benefits and fault tolerance that

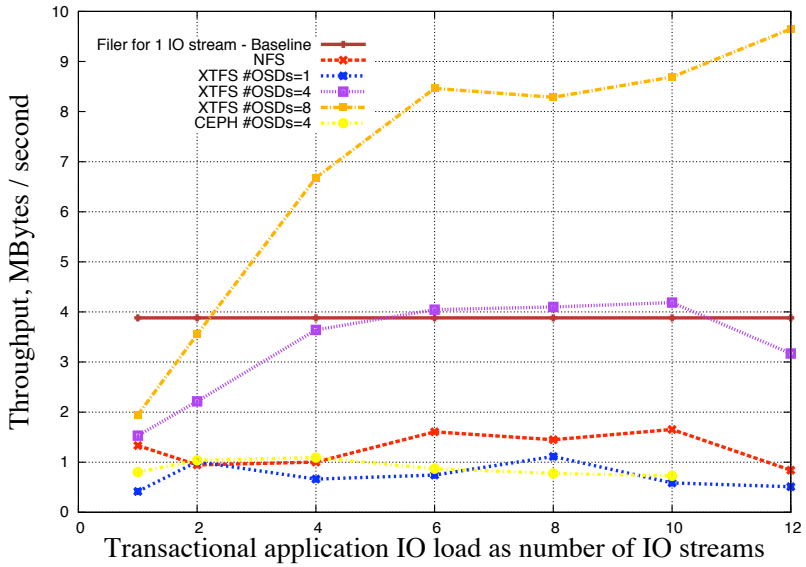


Figure 4.47: Write Throughput in MaxDB experiments.

XtreemFS provides for the consuming business applications thanks to the replicatione feature of XtreemFS.

Approach Refinements

Replication for fault tolerance

The goal of the test is to assess the fault tolerance and performance benefits that XtreemFS provides to its cunsumers, when the feature of XtreemFS to be tested is replication. The IO load that is used in our experiments is the transactional load that is generated by a typical business application.

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 (Gigabits) (E 1Gb/s).

This test uses the same IO load as in Test Unit 06.

In the experiments the number of OSDs equals the stripe width of the XtreemFS volume. The test is performed with replication level 2. The application is started over an XtreemFS volume, where the application files are replicated to several OSDs. During the application execution we removed up to two OSD servers or added new OSD servers and verified that the application is still running and is being able to access the XtreemFS volume correctly.

Replication for performance benefits

Using an XtreemFS volume with replication level at least 2 to store an application data increases the locality of the client IO reads. This is due to a higher probability of serving the data for the client's IO read operation from a local data replica. Thus using replication should provide the application with a better performance in the case of read-oriented IO load. In this test we use read-only IO traces, produced by MaxDB application to verify the potential performance benefits for the application. We use the mean latency and accumulated throughput over all concurrent application clients as the performance metrics.

4.9.8.3 Test Results

Replication for fault tolerance

Below we bring several screenshots that show testing the application stability under failures of XtreemFS OSDs.Figure 4.48 shows staring up the MaxDB replay application.

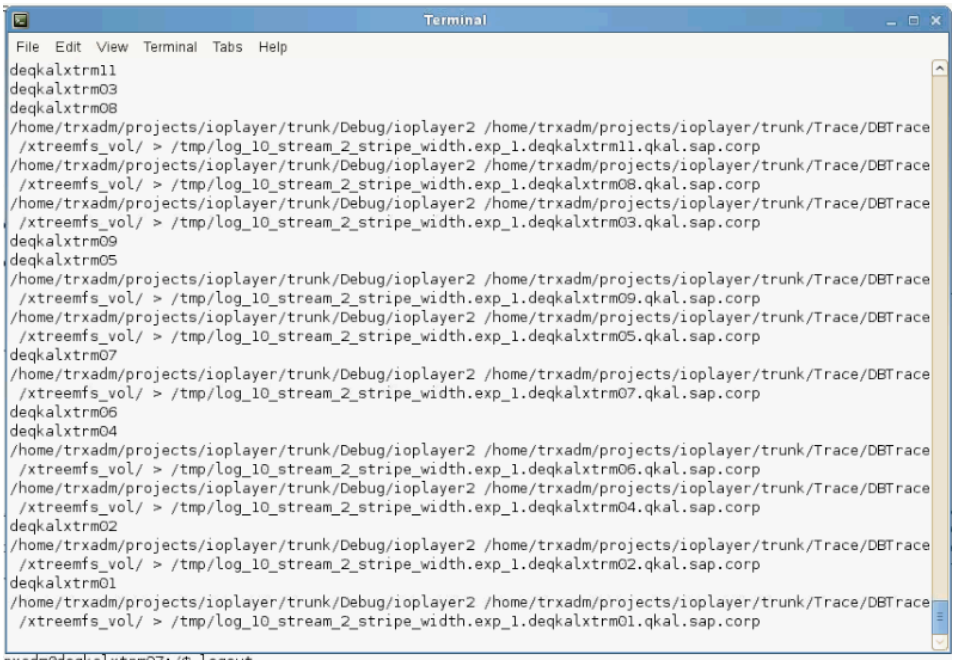


Figure 4.48: Starting up MaxDB application.

Figure 4.49 shows that the replication level of the files at XtreemFS is 2.

Figure 4.50 shows shutting down an OSD server.

Figure 4.51 shows that the application is still running in spite of the OSD server that has been shut down.

As the screenshots above suggest, while performing the tests we observed that

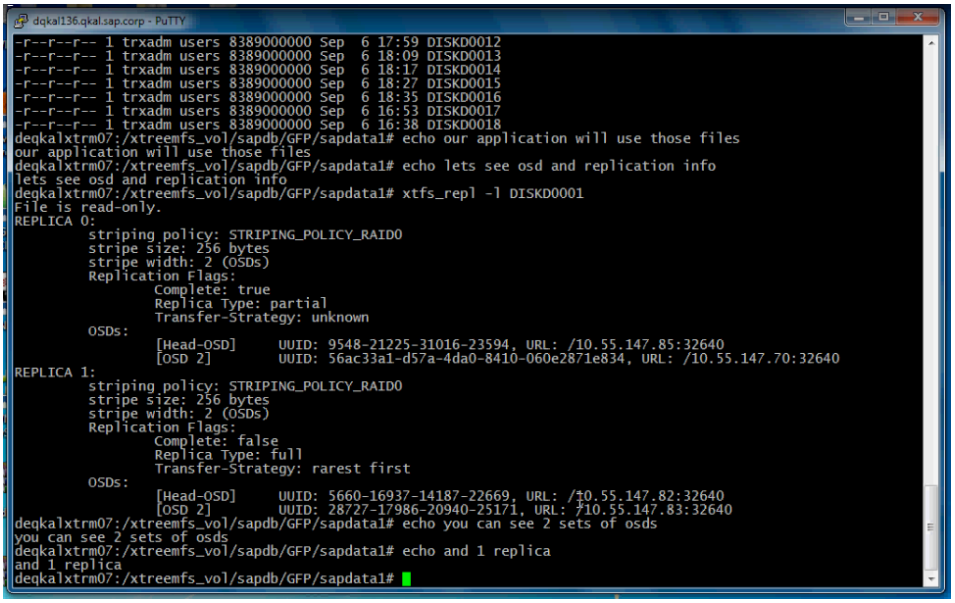


Figure 4.49: Replication level of XtreemFS files.

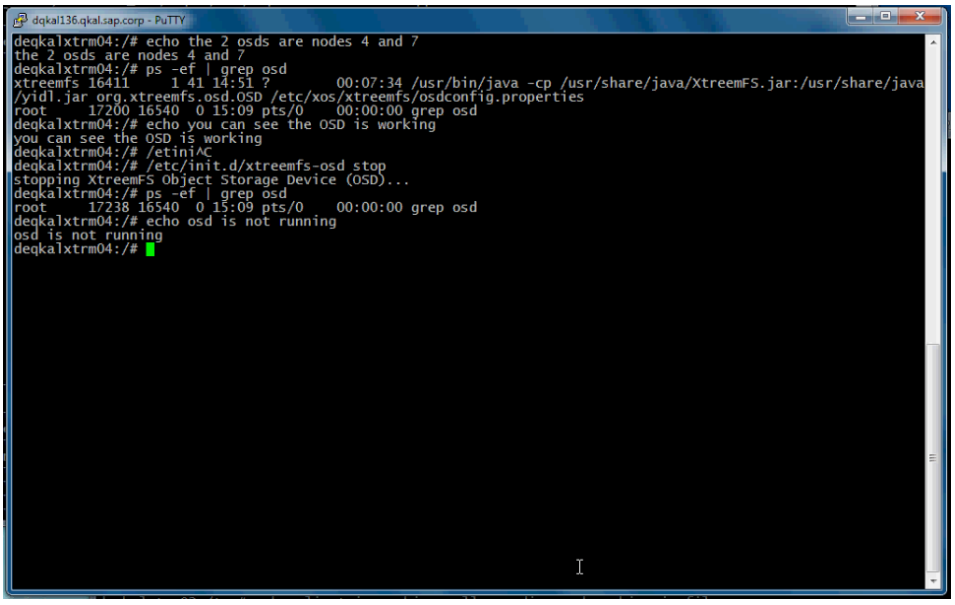


Figure 4.50: Shutting down an OSD server of XtreemFS.

the application is able to run correctly after we remove up to two OSDs that hold some replicas of the application data. That is the application is able to run smoothly

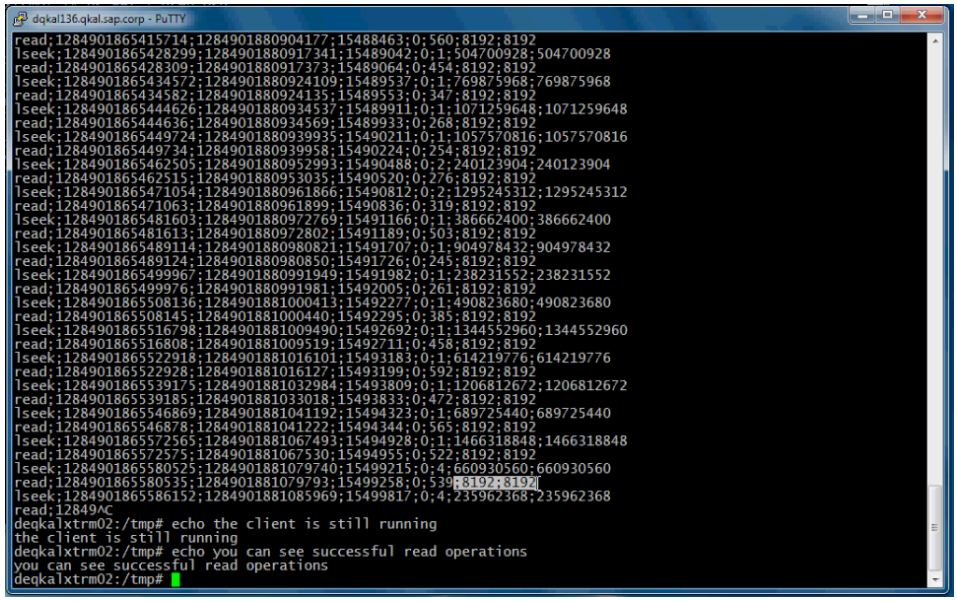


Figure 4.51: The application is still running.

as long as at least one replica for each portion of the application data is found on a still running OSD. We also tested that adding more OSDs, while the application is running, keeps the XtreemFS stable, enabling the application to continue correct execution while the infrastructural layer is dynamically changed.

Replication for performance benefits

In our experiments with using XtreemFS replication feature for performance benefits in a business application, we encountered a bug while running concurrent IO read-only traces over an XtreemFS volume with replication level of 2.Figure 4.52 shows the screenshot on one of the client nodes. In the screenshot one can notice that XtreemFS client and OSD server consume considerable amount of the CPU time. At the same time the MaxDB application is stuck and is not able to proceed. Analysing traces shows that probably the XtreemFS client tries to set O_RDWR and O_NONBLOCK flags to a file in a loop, at each iteration it fails to do so and finally crashes. Note that this faulty behaviour occures towards the end of replaying the IO trace. For this reason we can demonstrate replication advantages in the stability tests and we were not able to test performance advantages of XtreeFS due to the replication feature.

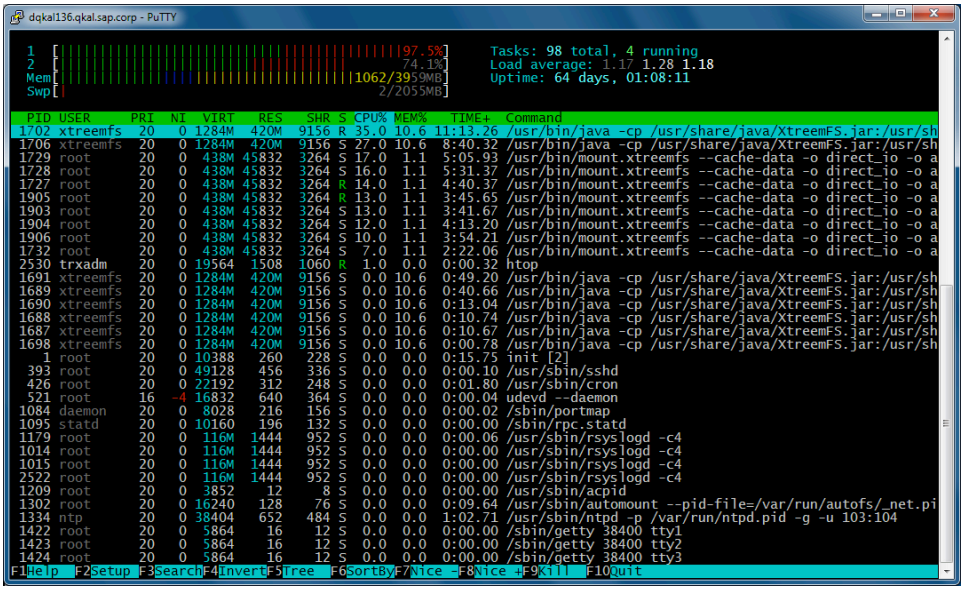


Figure 4.52: Stuck XtreemFS client.

Conclusions:
We performed tests to assess fault tolerance and stability that XtreemFS provides to the consuming applications and found that

- Replication feature of XtreemFS enables to provide the consuming applications with fault tolerance.
- Dynamically changing XtreemFS configuration via adding OSD servers does not disturb the consuming application.
- The replication feature in XtreemFS should be further stabilized in order to bring performance benefits for a business application.
- Overall conclusion: XtreemFS provides consuming business applications with fault tolerance.

4.9.9 Test Summary Report

4.9.9.1 Summary of Tests and Results

We performed tests to evaluate the performance of XtreemFS and compare it to the performance of NFS and CEPH. Tests were also performed to assess the performance under stress of Object Sharing Service provided by XtreemOS. We found that network latency is crucial for transaction processing and transaction throughput drops if latency is too high. Although, the maximum theoretical throughput could be reached using local commits. The incrementation of the shared variable offers contrary results of both token passing algorithms. Due to many transaction

aborts, incrementing the shared variable scales poorly. We also found that transactional conflicts depend on the granularity of object access detection.

In the POSIX compliance tests, XtreemFS passed 90.91% out of the adequate tests.

Our experiments with large-scale XtreemFS deployment show the potential of XtreemFS to support transactional load. However more work should be done to improve synchronous write latency, stabilize the replication feature in XtreemFS and further prove its ability to support business application transactional load.

Our comparative performance tests revealed that for read operations CEPH outperforms XtreemFS for up to 4 IO streams, while for higher number of IO streams the opposite is true; for write operations XtreemFS outperforms CEPH, excluding the case with only one OSD server.

The experiments with XtreemFS's replication show that XtreemFS provides consuming business applications with fault tolerance. However the replication feature in XtreemFS should be further stabilized in order to bring performance benefits for a business application.

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 using normal hard drives, it enables to reach the throughput of the baseline filer technology that uses solid-state drives (SSD).

4.9.9.2 Conclusion and Directions for Future Work

Based on the tests we conclude that POSIX Compliance leaves the room for improvement. Asynchronous write latency under the transactional load is still relatively high as compared to the baseline filer technology. And most probably this shortcoming may be overcome only by means of using solid-state drives (SSD) at OSD nodes. Our experimental results under transactional load look very promising and suggest that XtreemFS may support transactional load. However more experiments need to be performed to support this conclusion.

We can conclude that XtreemFS and Object Sharing Service in XtreemOS Release 2 are adequate, but performance leaving some room for improvement. Further testing is required for comparative performance analysis of XtreemFS with additional advanced file systems such as Lustre. Another set of tests is required to assess the performance benefits that a business application can get from the replication feature of XtreemFS. An additional set of experiments should include a usage of read/write replication 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

4.10.1.1 Responsibilities

This test plan is carried out under WP3.5 and involves ICT, INRIA, STFC, and XLAB. Some of the features under test have been developed in conjunction with WP2.1.

4.10.1.2 Test Items

The listing of test items is given below. Each of these is included in release 3 of XtreemOS and adopt their version information from this release unless otherwise stated.

- VO Policy Service
- Monitoring Service
- Auditing Service
- Isolation
- Single Sign-On
- CDA server, version 0.3.4

4.10.1.3 Features to be Tested

The Virtual Organisation Policy Service (VOPS) developed in WP3.5 will be tested by XLAB. This is included in release 3 of XtreemOS.

The Monitoring Manager was also developed in WP3.5 by XLAB and is integrated into version 3 of XtreemOS. The Monitoring Manager is depended on by the Auditing Manager, which is hence the subsequent test to be done.

Testing the isolation features of XtreemOS developed in WP2.1 and WP3.5 is the responsibility of ICT. This refers to the Account Mapping subsystem, network flow control and virtual memory controls.

4.10.1.4 Features not to be Tested

Key management and delegation are not tested. Key management is implemented by the pre-existing Linux ssh library. Delegation has been tested in the previous deliverable and has not been further developed. There are also no tests of reliability and fault recovery to be performed here, as this is beyond the scope of security services.

4.10.1.5 Overall Approach

Most of these tests are performance and overhead tests. They hence follow the procedure of comparing the performance of XtreemOS without the security features enabled versus with them enabled. The hypothesis is that the performance and overhead should be negligible or at least tolerable, as XtreemOS was designed for integration with the OS as opposed to being an additional software layer. Standard performance analysis tools and techniques are applied, such that the experiments are reproducible. The tests have been performed to avoid instrumentation and intrusion of the code-base, such that timestamped logs have been used in many cases.

4.10.2 Test Unit 01: Virtual Organisation Policy Service (VOPS)

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.

4.10.2.1 Responsibilities

VOPS is under development within WP3.5 under responsibility by XLAB.

4.10.2.2 Test Specification

In the third release of XtreemOS VOPS is implemented as a part of DIXI framework (see section 4.5: Evaluation of the DIXI Message Bus) and its performance depends mainly on the aforementioned framework. The main difference to the second release [44, 10] is implementation of the eXist-db[2] (Open Source Native XML) database. Policies previously stored as plain XML files are now stored as XML documents on this special database providing:

- XQuery 1.0 / XPath 2.0 / XSLT 1.0 (using Apache Xalan) or XSLT 2.0.
- HTTP interfaces: REST, WebDAV, SOAP, XMLRPC.
- XML database specific: XMLDB, XUpdate, XQuery update extensions.

The infrastructure of Grid'5000 is used run the tests. The machines had 2.5GHz Intel Xeon processor and between 0.8 and 3 GB of RAM.

[2] http://exist.sourceforge.net/

Test Items

The focus of the tests is the main VOPS backend library, packed in the **vops** and its DIXI equivalent packed under the name of **dixi-vom-vops**. The client-side component in XATI is packaged in the **dixi-xati** package, which provides client side methods. Test items taken under the probe are:

- Backend library: **vops**.
- DIXI frontend: **dixi-vom-vops**.
- XOSd's client frontend XATI: **dixi-xati**.

The latest documentation of the VOPS API resides in the document **Detailed API description of VOPS** [46]. The user guide [45] and the administration guide [43] provide useful information where to get and how to install and use the VOPS service. The user guide contains description of available user commands provided by the **dixi-xati** package.

Features to be Tested

We will provide tests of the following features of VOPS:

- Policy insertion and deletion.
- The invocation of the service calls defined by the service interface in an synchronous manner from the 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). Number of client services will vary.
- Access request time while incrementing the number of policies on the server (the XATI client invoking will reside on a different node than the server).
- Backing up policies from the policy storage and saving the policies from the storage in memory to disk.
- Reloading policies from the policy storage from the disk.

Table 4.8 presents VOPS' features integrated with the eXist database.

Approach Refinements

Our assumption is that the system has been properly installed and set up. Tests have been conducted in a way that the server has been placed in different situations

Feature	item	Description
Policy insertion	all	Injecting policies into the policy storage
Policy removal	all	Removing policies from the storage
Access request	all	Providing PDP to other VO services
Policy backup	all	backing up policies to the physical storage
Policy restore	all	Restoring policies from the physical storage
PDP test	backend library	User request, policy reload

Table 4.8: Summary of the main features tested in VOPS provided by the VOPS server API. In the test of each feature, all test items have been involved, except in the last test where only the PDP of the VOPS has been tested.

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

- The goal of the test is to benchmark performance of the policy insertion process. The benchmarks have been performed in different set-ups: client and server have been located on different machines (1 server running 4 threads in DIXI stage for the VOPS server, different number of clients – 1,2,4,10) and the server has been exposed to changing load (different number of policies inserted or deleted from the clients).
- Hardware: workstations with the latest (Release 03) DIXI installation (variable number of machines on Grid5000 [5], machines have had 2.5GHz Intel Xeon processor and between 0.8 and 3 GB of RAM),
- Test techniques: the VOPS server has been running on one workstation, the clients have been running on different workstations running XATI (VOPS client side). On the VOPS server there have been running **mpstat** and **vmstat** with a period of 1 second, which have been benchmarking the CPU usage, load and memory usage.
- Input parameters: generic policies have been generated by testing environment written in Java.
- Output parameters: None.
- Metrics to be used: on the client side the time between several operations has been measured between operations, on the server measurements of the user CPU usage, system load and the memory usage have been done.

Policy removal The tests have been done in the same way as described in the **Policy insertion** section.

VOPS Access request and PDP test

- Goal of the tests is to benchmark performance of the VOPS' Policy Decision Point. Benchmarks have been performed in two set-ups: client and server both residing on the same machine (we have tested only the performance of the XACML engine – the PDP test), client and server have been located on different machines (VOPS Access time).

- Hardware: two instances of virtual servers on Grid5000, both with the same DIXI installation, third release.

- Test techniques: VOPS server running on one virtual server, client running on different machine. On the machine with VOPS server there were running **mpstat** and **vmstat** with a period of 1 second, which were benchmarking the CPU and memory usage.

- Input parameters: generic policies on the server side, generated by testing environment written in Java. We have also needed generic user and resource certificates. The job description has also been provided by the testing environment.

- Output parameters: None.

- Metrics to be used: on the client side we have been measuring the time between operations (access request), on the server the user CPU time, system CPU time, and number of interrupts per second have been measured.

Policy Backup This is a feature which enables VOPS to dump current storage into XML files and vise versa - to load XML policies into eXist database.

- Goal of the test is to benchmark performance of the VOPS policy load/write feature.

- Hardware: virtual server on Grid5000 with DIXI installation and VOPS running, third release.

- Test techniques: VOPS server running on the virtual server. There were also **mpstat** and **vmstat** with a period of 1 second benchmarking the CPU usage, system load and memory usage.

- Input parameters: generic policies on the server side, generated by the testing environment written in Java. We have also needed generic user and resource certificates and job description has also been provided by the testing environment.

- Output parameters: None.

- Metrics to be used: on the server side we have been measuring time between operations (loading the policies, saving the policies); user CPU usage, system load, and memory usage were also measured.

Policy Restore The tests have been done in the same way as described in the **Policy Backup** section.

4.10.2.3 Test Results

In this section we present test results for tests specified in former sections.

Policy insertion and removal Here we present insertion and removal tests. VOPS server was set to use 4 threads in a DIXI stage on the server.

Num. of policies	1 client (avg)	st.dev	2 clients (avg)	st.dev	4 clients (avg)	st.dev
1	259.0	0	886	0.0	309.0	0.0
10	44.9	4.79	557.4	24.17	275.7	42.04
100	91.28	31.72	700.48	167.28	375.78	113.90
1000	592.054	323.9	1208.99	448.03	2383.77	1319.19

Table 4.9: Inserting policies into VOPS server simultaneously from different number of clients: numbers present the amount of time which takes to insert one policy in a test. There were totally 10 repetitions of each test performed.

Num. of policies	1 client (avg)	st.dev	2 clients (avg)	st.dev	4 clients (avg)	st.dev
1	23.0	0.0	38.0	0.0	160.0	0.0
10	22.0	2.10	28.2	1.03	69.9	60.0
100	21.15	2.70	27.14	2.35	113.0	183.40
1000	25.09	4.22	22.86	2.38	349.27	1056.92

Table 4.10: Deleting policies from VOPS server simultaneously from different number of clients: numbers present the amount of time which takes to delete one policy in a test. There were totally 10 repetitions of each test performed.

Deleting policies (amount of time for deletion per policy) does not depend on the amount of policies being deleted when the load is small. However, that is not true while inserting policies. We can see deterioration while increasing the amount of policies (1000 policies) or increasing the number of clients. Explanation for that would be indexing performed by the eXist database. On the other hand, this kind of situation when we need to insert a large amount of policies, are rare.

In tables 4.9 and 4.10 we show figures of tests conducted. Table 4.9 presents average times and deviation while inserting different amount of policies. **Number of policies** present the amount of policies inserted into the server from one client. Column **4 clients (avg)** presents times while inserting from 4 clients simultaneously. Numbers present insertion of one policy among **Number of policies**. Same explanation goes for table 4.10.

We conclude, that comparing times to insert and delete one policy under heavy load is linear (approximately linear). In case of 4 clients inserting policies into database simultaneously: 1000 generic policies into VOPS database from a client is about 6 more consuming (per policy) than when inserting and deleting 100 generic policies (comparing 100 to 10 policies it makes approximately 1.3 times). Similar behaviour is with deleting policies. Under heavy load average time to delete a policy from a server takes about 3 times more time than under less load (4 clients inserting and deleting 1000 policies versus 4 clients inserting and deleting 100 policies).

# policies	1	10	100	1000
4 clients avg	553.75	3263.75	$40.38 \cdot 10^3$	$2.41 \cdot 10^6$
4 clients stdev	65.81	139.54	8.58	27.58
avg/#policies	553.75	326.38	403.82	2413.03
10 clients avg	511	3653.44	$82.41 \cdot 10^3$	$5.44 \cdot 10^6$
10 clients stdev	166.96	110.35	276.81	432.31
avg/#policies	511.00	365.34	824.14	5442.03

Table 4.11: Average times for inserting and deleting policies on 4 and 10 clients simultaneously. All numbers are in milliseconds. Average times include generating 1000 policies.

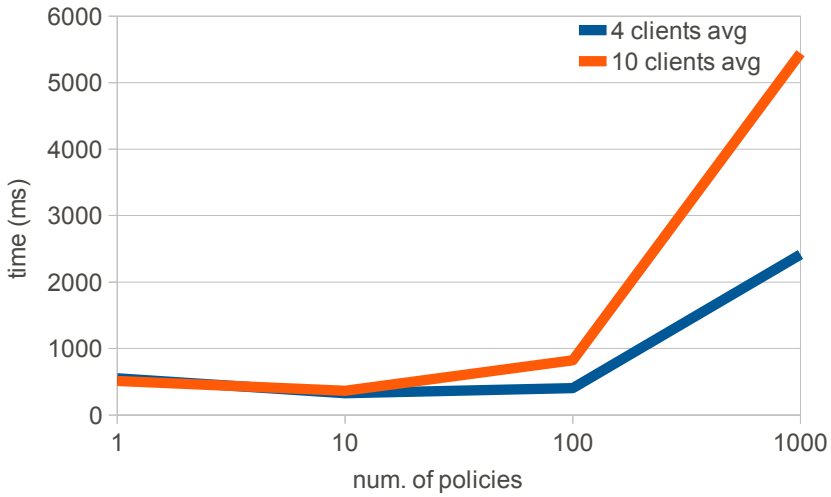


Figure 4.53: Average time for inserting and deleting a policy under high load. Axis X presents number of policies inserted and deleted by each among 4 (and 10) clients simultaneously.

In table 4.11 we present number for inserting and deleting policies from 4 and 10 clients simultaneously. One operation equals to inserting and deleting a policy. Figure 4.53 presents numbers **avg/#policies** for 4 and 10 clients — comparison between times of one operation under different load (axis x).

Difference between load of 1 and 100 policies when doing operation from 4 (10) clients simultaneously is not so drastic. Deterioration can be seen when 1000 policies are inserted simultaneously.

VOPS access request In table 4.12 we times to evaluate a request towards VOPS Policy Decision Point are depicted. Times are computed as an average time of 50 access requests and with different number of policies in the database. We can see that time to evaluate a request does not depend on the number of policies residing in the VOPS policy storage. That is great advantage of integration with the eXist database. Searching for the appropriate policy is does not depend on the number of policies.

Num. of policies	Avg time (ms)	stdev
1	74	20.04
10	77	24.87
100	39	19.02
1000	36	23.12

Table 4.12: Time per request when number of policies on the server incrementally increases.

Policy backup and restore We have conducted tests for loading and saving different number of policies on the server. Table 4.13 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 4.13: Time for an operation of loading and saving specified number of policies into and from a storage respectively.

As we can see in the table time to load or save different number of policies is nearly linear.

PDP test Here we present test results for VOPS Policy Decision Point performance. Tests were conducted on two machines running both server and client.

Our reference for performance evaluation is a detailed comparison between different XACML engines described in [42] where the choice of the Sun's XACML engine over other alternatives is justified. We add our results of timing submissions of generated user requests to VOPS PDP in the same manner, where the requests contain a varying number (10, 100, 1000 and 10000) of generic policies consisting of four rules of which three of them deny the access to the resource and one of the rules always permits the access. First policy conforming to the request and denying the action (no permissive rule resides in the policy) stops the whole evaluation by denying the evaluation. In order to time the evaluation of all policies in the storage, a permissive rule exists in each policy and, therefore, each policy conforms to the request.

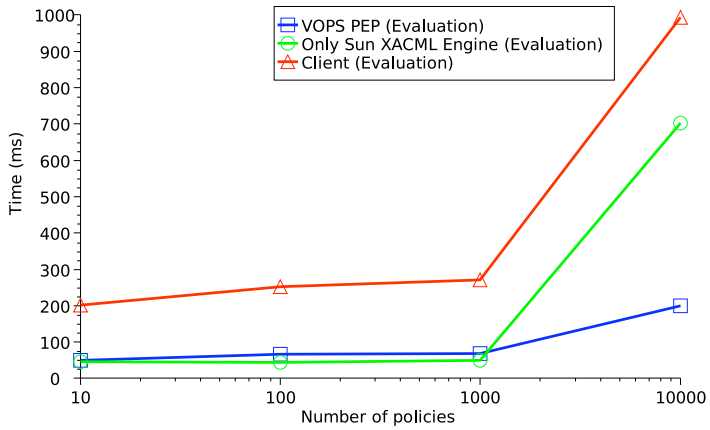


Figure 4.54: Times to evaluate user request with large number of policies in PDP.

The results in Figure 4.54 show that we introduced some overhead to the response times in our implementation consisting of the eXist engine for storing the XACML policies and the XtreemOS service staging the communication bus.

On the other hand, we gained the ease to manage the policies, extract the filter policies from the database and separate the communication part from the core of the service. The blue graph presents average times for XACML engine's PDP, the green graph presents times obtained by authors in [42], which is shown as a comparison. The red graph presents average times of first user's access request evaluation, i.e., when VOPS is started. In case of 10000 policies, not taking into account the first access to PEP, which is the most expensive, we obtain average time of 174 ms with standard deviation of 88 ms (50 tests were performed). By that we get close to the desired blue graph with small addition due to usage of communication framework and network latency of around 40 ms.

Figure 4.55 depicts how much time is consumed for saving the policies to database and loading different amount of policies from database into main memory preparing the policies for evaluation.

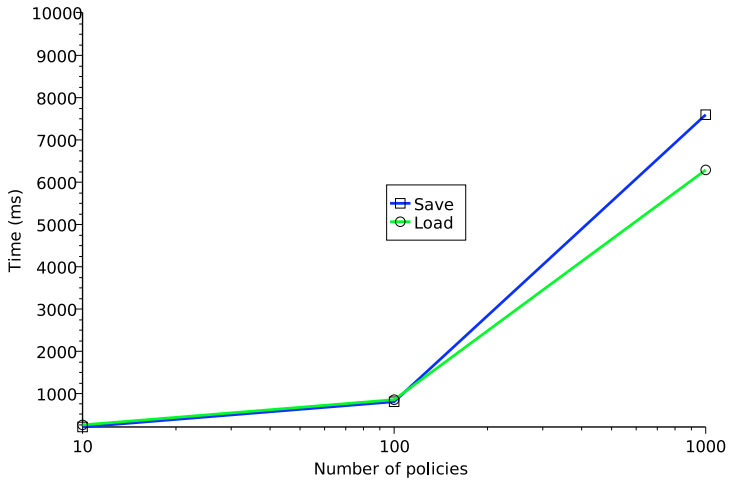


Figure 4.55: Time for loading large number of policies and PDP construction (green line) and time for saving policies from memory to database.

4.10.3 Test Unit 02: Monitoring features and latency of monitoring notifications

Monitoring Manager is a core service which receives monitoring data, implements monitoring rules engine and handles the distribution of monitoring events. Various services send metrics and events which are collected by Monitoring Manager. Users define monitoring rules to which they or other interested parties can subscribe. When monitoring rule's conditions are met, a callback is triggered and subscribers get notified.

4.10.3.1 Responsibilities

This test was carried out within WP3.5 by XLAB.

4.10.3.2 Test Specification

A variable number of virtual machines on Grid5000 were used run the tests. The machines had 2.5GHz Intel Xeon processor and between 0.8 and 3 GB of RAM.

Test Items

We tested the following features of Monitoring services:

- Monitoring rules
- Notifications

Monitoring Serivce can be downloaded from the project repository, as well as the users and installation guide.

Features to be Tested

We tested the following features of the Monitoring DIXI API:

- Creating monitoring rules
- Removing monitoring rules
- Subscribing to monitoring rules
- Unsubscribing from monitoring rules
- Sending events and metrics
- Sending monitoring notifications

Approach Refinements

The main focus of this test unit was to measure the delay when sending monitoring notifications regarding to the number of resources and monitoring rules. In order to test this, we had to use all monitoring features, meaning we tested a much broader aspect of the Monitoring Service than just the delay of the notifications.

Platform used to execute the tests was Grid5000, which allows reservation of variable number of resources. The way we tested Monitoring and Auditing features on different number of resources was to use Grid5000's front-end, which allows reservation of arbitrary number of virtual machines. After reservation the XtreemOS Release 3.0 was deployed on the machines and configured.

To evaluate Monitoring and Auditing features, we created Bash scripts and DIXI applications, which handles the execution of monitoring. Since the scripts are parameterizable, we could easily test the features in different configurations. The main benchmark script ran from Grid5000's front-end, which took care of distributing "node" script to all resources and handling the actual benchmark. "Node" script executed DIXI application on every node which produced the results file.

The procedure was the following:

1. Execution script was deployed to all resources

2. Monitoring rules were created

3. Node script was executed on every resource, which in turn executed Monitoring benchmark class

4. When benchmark finished, results files were copied from resources to the Grid5000's front-end

Although we tested all monitoring features, the delay was measured only with monitoring notifications. This is due to the fact that in practice creating, removing and subscribing to monitoring rules operations are done rarely or onetime only and does not significantly impact on the performance of Monitoring Service.

4.10.3.3 Test Results

Monitoring rules were successfully created and removed using command-line and through DIXI application, as well as subscribing and unsubscribing from monitoring rules. Monitoring notifications were received by benchmark DIXI application, which calculated notifications' latency by comparing current system's time with notification's time stamp. Received notifications' data and calculated delay were successfully written to results file.

To increase performance of Monitoring Service, Monitoring Directory Service was introduced. Monitoring Directory provides means to connect running instances of Monitoring Managers to increase their autonomy. As seen of figure 4.56, with the increasing number of monitoring rules and number of resources, the latency of notifications increases, though due to the use of Monitoring Directory Service, the delay of monitoring notifications doesn't surpass one second until the number of monitoring rules reach around one hundred while ten resources are in use.

4.10.4 Test Unit 03: Auditing features and latency of history database queries

Auditing Manager is used to permanently store monitoring data. It relies on Monitoring Manager to receive monitoring data which is then stored to history database. Auditing history database can be queried in order to analyze monitoring data or generate various reports.

4.10.4.1 Responsibilities

This test was carried out within WP3.5 by XLAB.

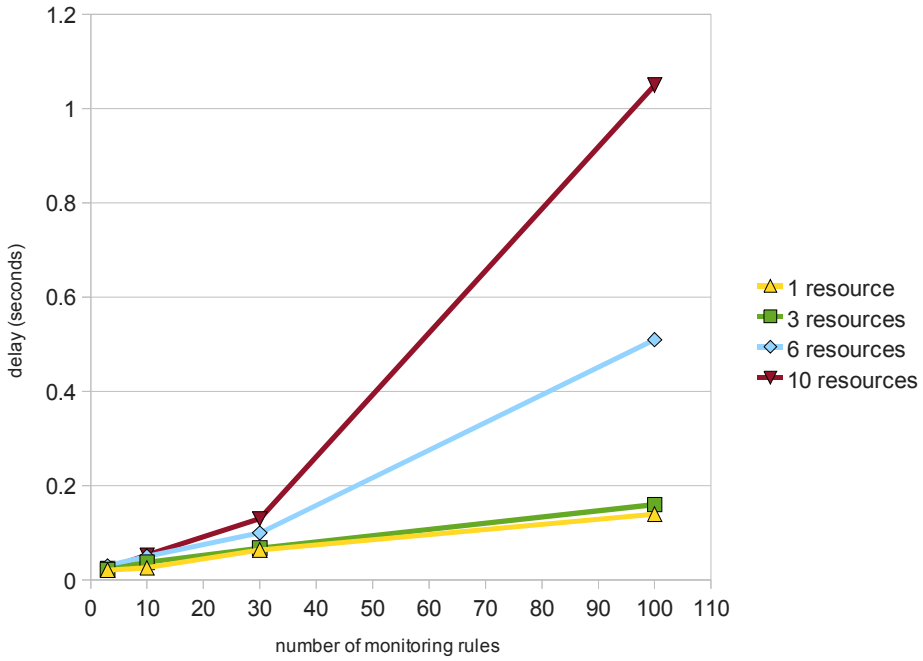


Figure 4.56: Monitoring notifications delay in seconds depending on the number of monitoring rules and resources.

4.10.4.2 Test Specification

A variable number of virtual machines on Grid5000 were used run the tests. The machines had 2.5GHz Intel Xeon processor and between 0.8 and 3 GB of RAM.

Test Items

We tested the following features of Auditing services:

- Handling archiving rules
- Querying of the history database

Auditing Service can be downloaded from the project repository, as well as the users and installation guide.

Features to be Tested

We tested the following features of the Auditing DIXI API:

- Creating archiving rules

- Removing archiving rules
- Querying history database

Approach Refinements

The main focus of this test unit was to test the delay when querying Auditing history database regarding to the number of resources and archiving rules, though in order to achieve this, all features of the Auditing Service were used.

Similar to the Monitoring Service unit test (4.10.3), benchmark scripts and DIXI applications were used to perform the tests.

The procedure was the following:

1. execution script was deployed to all resources
2. archive rules were created
3. node script was executed on every resource, which in turn executed Auditing benchmark class
4. when benchmark finished, results files were copied from resources to the Grid5000's front-end

Although we tested all features of the Auditing Service, the delay was measured only with Auditing history database querying. This is due to the fact that in practice creating and removing archiving rules operations are done rarely or only at initialization and does not significantly impact on the performance of Auditing Service.

4.10.4.3 Test Results

Archiving rules were successfully created and removed using command-line commands. Results file of the test revealed that monitoring data was successfully stored to Auditing history database.

As seen on figure 4.57, the number of resources and archiving rules significantly affects on the querying time of the Auditing history database. This is due to the fact there is only one instance of Auditing Service running.

Although we focused on the functionality of the Auditing Service, the groundwork for making Auditing more scalable is set. There are different approaches that could be used to make Auditing more scalable, e.g. using Scalaris to distribute Auditing history database over the resources.

There is some more manners that could be used to make querying of the database faster. One of them that was not used during the testing was database indexing, which would quite significantly reduce execution time of the queries. Another way

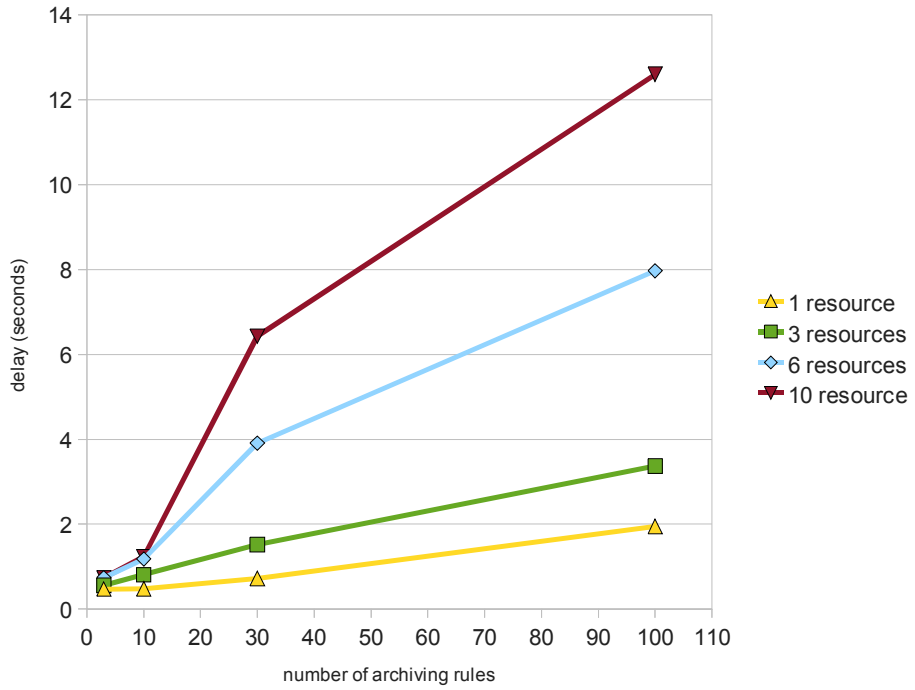


Figure 4.57: Delay in seconds when querying Auditing history database depending on the number of archiving rules and resources.

would also be to use a database with specifications that might better suit Auditing Service. Due to the use of Hibernate for abstracting database access, switching to any other database would be only a matter of updating Hibernate's configuration.

4.10.5 Test Unit 04: Isolation Experiments

The goal of isolation experiments is to evaluate the resource isolation functions of XtreemOS and the performance penalty brought by enforcing islation mechanisms.

4.10.5.1 Responsibilities

WP3.5, Zhouyi Zhou (ICT)

4.10.5.2 Test Specification

Test Items

The tested items are:

- The automatic cgroup filesystem setup invoked by AMS (Account Mapping Subsystem):
  SVN co svn+ssh//scm.gforge.inria.fr/svn/xtreemos/foundation/xtreemos-nss_pam/trunk, compile it, then invoke src/examples/pam_app_aem;

- The performance overhead brought by network flow control: Zhouyi Zhou from ICT has modified the Linux kernel to add the CPU cycle counting hooks (see below);

- The performance overhead brought by virtual memory control: Zhouyi Zhou from ICT has modified the Linux kernel to add the CPU cycle counting hooks.

Features to be Tested

The features to be tested on isolation are:

- When AEM submit a job with isolation requirement to AMS (Account Mapping Subsystem), AMS should automatically mount CGROUP filesystem and setup quoation according to the job description;

- The performance overhead measured by means of CPU cycles brought by network flow control;

- The performance overhead measured by means of CPU cycles brought by virtual memory control.

Approach Refinements

- The test machine used is equipped with a Intel 4 core 2.6G CPU and 2G Memory, 2 1000M Network Cards;

- scm.gforge.inria.fr/svn/xtreemos/foundation/xtreemos-nss_pam/trunk/src /examples/aem/XPamAPIs.c contains test code for automatic isolation configuration;

- To test performance overhead of network flow control, Zhouyi Zhou has modified Linux/net/socket.c to add rdtscll around the isolation hooks in functions __sock_sendmsg and __sock_recvmsg, he also add sysctl entry for userland test program to read the accumulated CPU cycles overhead;

- To test performance overhead of virtual memory control, Zhouyi Zhou has made similar modification to XOS Linux kernel.

4.10.5.3 Test Results

- After invoking ./pam_app_aem XXX.pem /bin/bash There should be a new directory named aem_jobid_XXX entry under /mnt/xos_cgrp/. memory. Content of limit_in_bytes and net.tot should be related to the array "unsigned long long quotas" in scm.gforge.inria.fr/svn/xtreemos/foundation/xtreemos-nss_pam/trunk/src/examples/aem/XPamAPIs.c.

- The performance overhead of network flow control handling a ssh login session is about 9448 cycles, The performance overhead of virtual memory control handling a ssh login session is about 15448 cycles.

  Compared with total CPU cycles spent on performing the ssh login (about 3729768664 CPU cycles), the perfermance overhead of resource isolation is neglectable.

4.10.6 Test Unit 05: Evaluation of CDA Server

The CDA (Certificate Distribution Authority) server is the core of the Public Key Infrastructure used in XtreemOS. It is used to generate user XOS-Certificates, containing the user's identity and VO attributes (details such as VO and group membership). To provide a reliable, trusted service, is has to authenticate user requests against their registration details (username and password) stored in the X-VOMS database.

4.10.6.1 Responsibilities

The CDA server and corresponding client software have been develeoped in WP3.5 by STFC.

4.10.6.2 Test Specification

Test Items
The object of the test is to measure the performance of the CDA (Certificate Distribution Authority) server, version 0.3.4. This is the second major release of the CDA server, which is a standalone component (that is, the CDA server is not part of the DIXI framework). The purpose of the CDA server is to create certificates for a user containing details of their identity and their VO attributes. The CDA server authenticates the user against the X-VOMS database, and returns an X.509v3 certificate (the XtreemOS 'XOS-Certificate) containing the user's Globally Unique Identifier (GUID) in the certificate Subject field, and their VO attributes in certificate extension fields.

The main change in this version of the CDA server is the option to generate service certificates (that is, certificates used to authenticate other XtreemOS services, such as VOPS and RCA, to their corresponding clients). This is an alternative to

the previous mode of working, which required system admistrators to create a Certificate Signing Request file by hand and to get it manually signed by the operator of the Root CA. The CDA client and server are described in [44].

Installation and configuration of the CDA server is described in the XtreemOS Adminstration Guide, and use of the CDA client is described in the XtreemOS User Guide. The code of the CDA server is available in the project SVN at *https://gforge.inria.fr/scm/viewvc.php/grid/cdaserver/branches/cdaserver-service-cert-support/?root=xtreemos*. The source code of the corresponding CDA client is available at *https://gforge.inria.fr/scm/viewvc.php/grid/cdaclient/branches/cdaclient-host-cert-support/?root=xtreemos*

Features to be Tested

The performance of the CDA server will be measured by recording the values of CPU parameters during the time that the CDA server is receiving . These parameters include CPU load and memory utilisation. These measurements will be carried out on the machine where the CDA server is running. The CDA client program, get-xos-cert, will be used to make multiple requests on the CDA server in succession. There will also be parallel streams of requests from CDA client programs running on more than one machine. There will be no CPU measurements carried out on machines where the CDA client is running, but the overall time taken to send multiple requests may be recorded.

Approach Refinements

Here we describe the approach taken to perform the test.

- The goal of the test is to measure the change in memory usage and CPU utilisation while procssing a stream of requests
- The hardware used is the same as for the last set of performance measurements, reported in D4.2.6. This is an Intel dual core processor running at 2.40GHz with 3GB RAM. The CDA server, version 0.3.4, runs on a node configured with the "task-xtreemos-coreservices" package. . XtreemOS 2.1 is installed directly onto this machine, with the latest patches applied. The version of the CDA server running on this machine is v0.3.4, and the version of the X-VOMS database is v0.3.4. This is equivalent, for the purpose of this test, to a machine with XtreemOS 3.0 installed.

  The client machine(s) used: MacBook Pro, Intel dual core processor 2.20GHz running XtreemOS 3.0.

  There is no virtualisation software used on client(s) or server during this test.

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 a short while 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 reported by the "time" command (user+system) for running "get-xos-cert" is recorded.

5. On the core node, the "dstat" command is stopped and the output file preserved.

4.10.6.3 Test Results

The test was run by issuing 100 request on the client side. The CPU parameters measured on the server during the processing of 100 requests is shown in Figure 4.58.

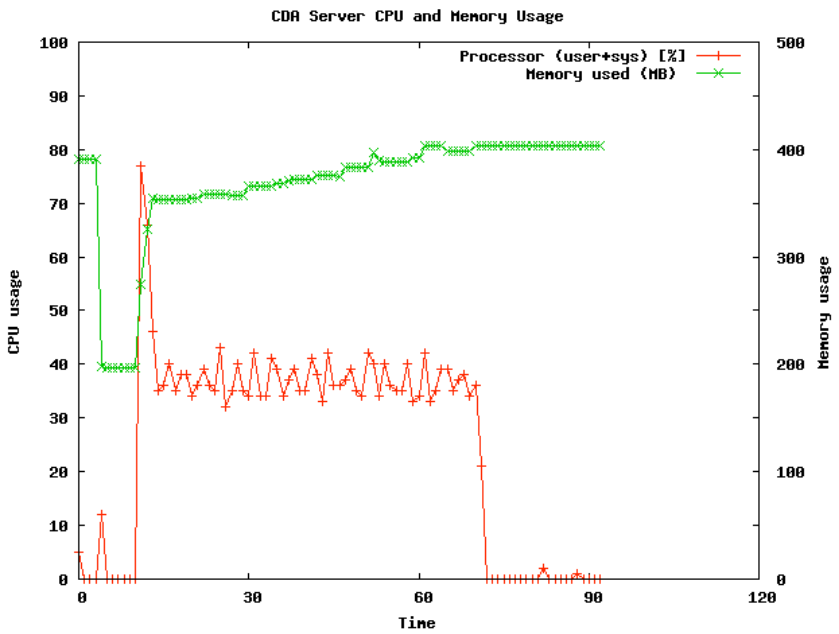


Figure 4.58: CPU load (left axis) and memory usage (right axis) while CDA server is processing 100 requests.

The result shows that the CPU usage of the CDA while processing a stream of 100 requests is around the 40% mark. The memory used during this period increased by 200 MB, the majority of which is due to the CDA server using more memory as it processes the requests.

4.10.7 Test Summary Report

4.10.7.1 Summary of Tests and Results

The main security features of XtreemOS have been evaluated. These are consistent with the features expected from any Grid platform but enhance the state of the art by integrating them with the operating system. For example the integration of OS-level isolation features is an improvement to existing practice. The security architecture is comparable to the Security Architecture for Open Grid Services proposed by Nagaratnam et al.[28], but does not include features such as intrusion detection, anti-virus managment and secure conversations, as these are not fundamental to the Grid security problem.

4.10.7.2 Conclusion and Directions for Future Work

It is possible to implement Grid security mechanisms that are integrated with OS-level mechanisms without introducing significant overheads. Future work is to apply these security principles in the context of Cloud computing, where there is more emphasis on virtualization and collocation of services. In this case the isolation features become even more important for management and not just security.

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.
- XtreemOS-MD mechanisms for resource sharing (G-layer).
- 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

FiXme Note: *Santiago*

No for these 3 tests, graphics provided later (apps and statistics)

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.

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 3.0 Beta2. Source and documentation are available on the internal XtreemOS SVN.

4.11.1.3 Features to be Tested

The following features will be tested:

- XtreemOS 3.0 support in mobile phones (ARM architectures).
- VO support of XtreemOS-MD.
- Lightweight security support of XtreemOS-MD (considered also as a performance test).

- Performance of AEM and XtreemFS in XtreemOS-MD.
- Support of specific applications created inside XtreemOS project (the application in this case will be JobMA).
- XtreemOS-MD support of main services (XtreemFS, AEM, CDA).
- Resource sharing: 3G connection sharing and data sharing.

4.11.1.4 Overall Approach

The purpose of these tests is to evaluate the current version of Mobile-Device software (3.0), provide feedback to developers, detect bugs and improve the final 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 3.0 and the XtreemOS PC client version (3.0) and also to demonstrate the benefits of the use of Grid services natively from mobile devices.

The tests will be done on a Nokia N900 and Nokia N800.

The test preparation requires the following tasks:

1. Install the software XtreemOS-MD on the Nokia N900 and N800 terminals.
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 3.0 could be downloaded from the project repository. 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 customary processor 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 N900 will be used for this test. No need of special software pre-installed in the Nokia N900. 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 3.0 could be downloaded from the project repository and 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 N900 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 N900 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 through VOLife GUI from the Nokia N900 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 3.0 could be downloaded from the project repository and the users guide and installation guide could also be found in the project repository. Also CDAProxy, included as part of XtreemOS release 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 N900 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 until obtaining a stable average measurement of 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. Actually, those commands are scripts using the time facilities of the underlying operating system.

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 N900 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 4.59 and 4.60 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 N900.

4.11.5 Test Unit 04: Performance comparison with XtreemOS PC flavor and no-Grid solutions

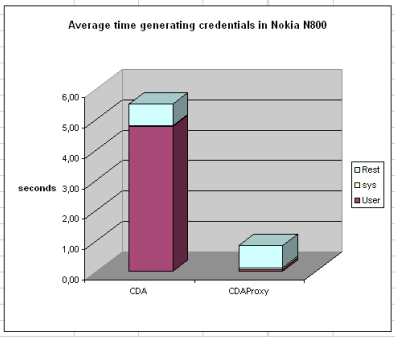
4.11.5.1 Responsibilities

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 3.0

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



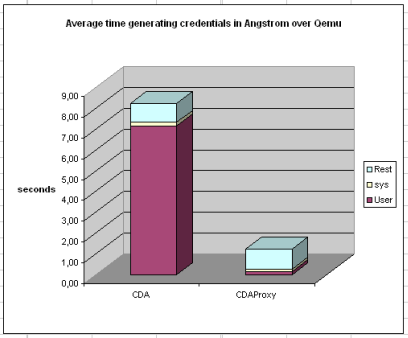


Figure 4.59: CDAProxy comparison

could be downloaded from the project repository and the users guide and installation guide could also be found in the project repository.
For this test, XtreemOS-MD will be installed in a mobile device: Nokia N900. Also, a XtreemOS PC client and a Core included as part of XtreemOS 3.0 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 (with and without XtreemOS-MD software installed on it) 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

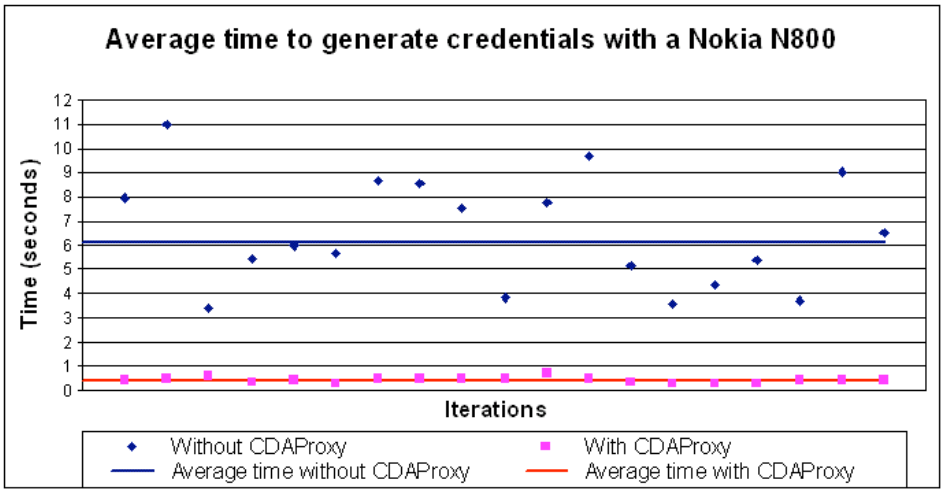


Figure 4.60: CDAProxy comparison results

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. The conversion process is optimized for the target device, a Nokia N800 terminal. Both videos, original and converted are stored in XtreemFS.
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 4.61, 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.

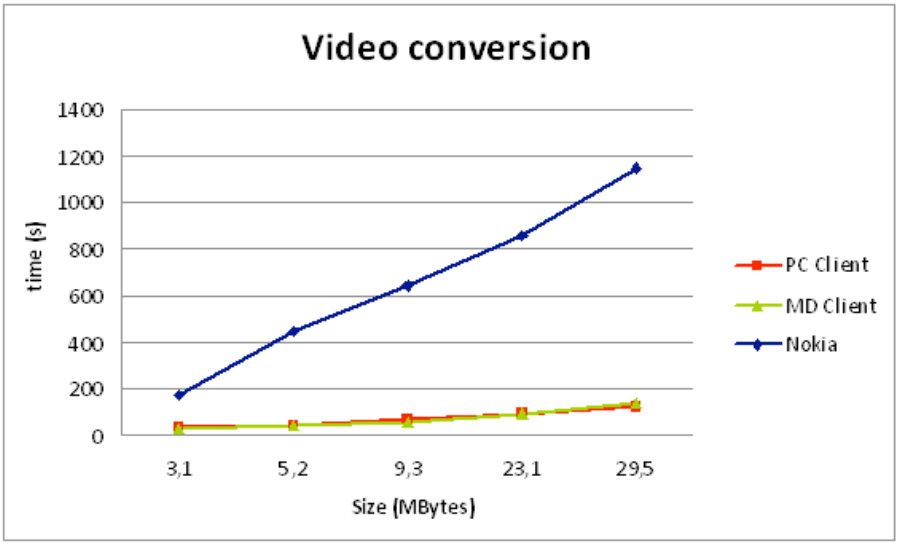


Figure 4.61: Video conversion performance

- `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.

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 4.62

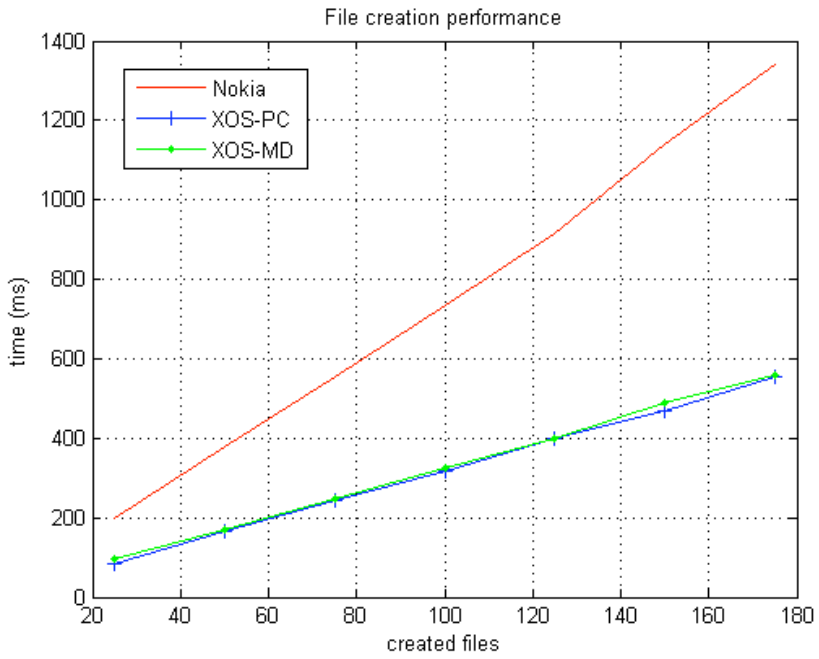


Figure 4.62: File creation performance

- `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 subtest) 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 4.63

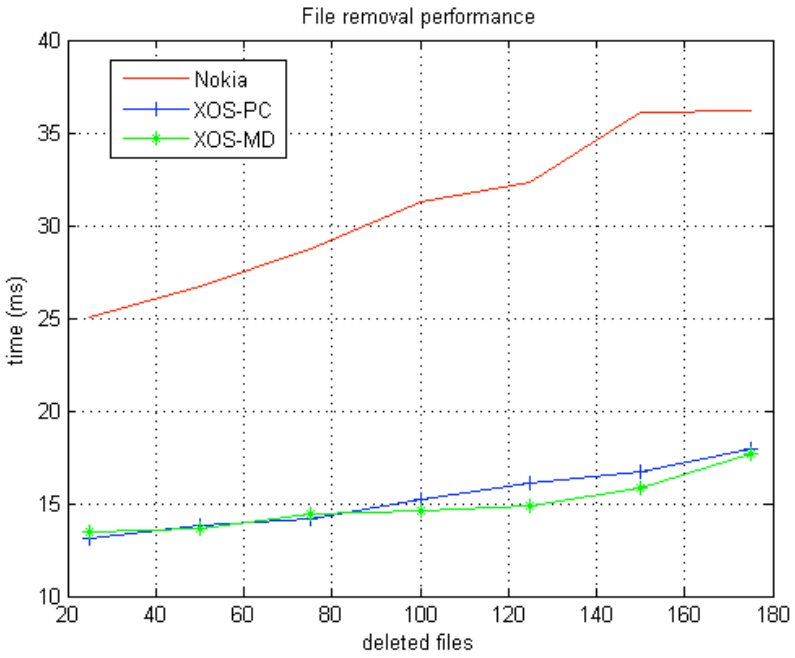


Figure 4.63: File removal performance

- `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 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 4.64

- `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.

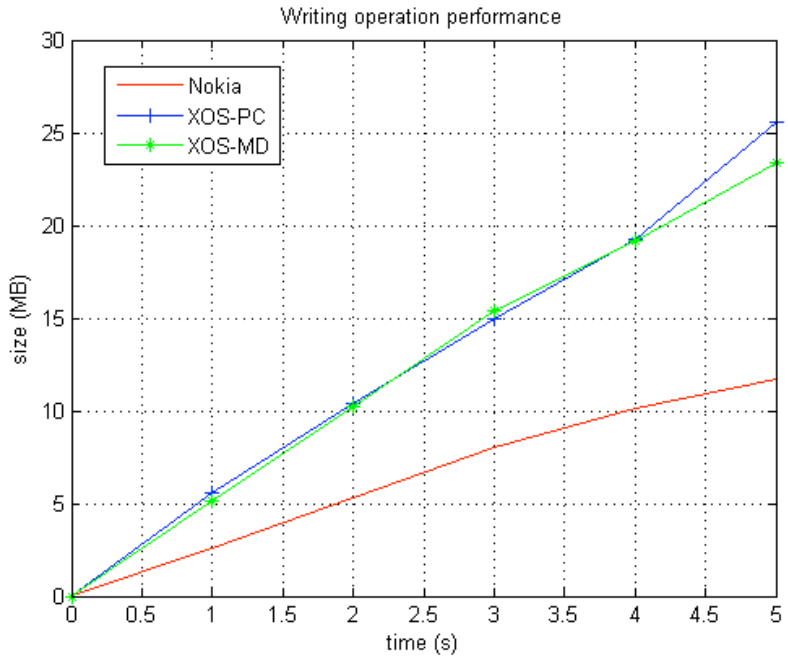


Figure 4.64: Writing operation performance

- `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 other words, the implementation of XtreemOS-MD does not penalize any Grid feature. 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.

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 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 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.

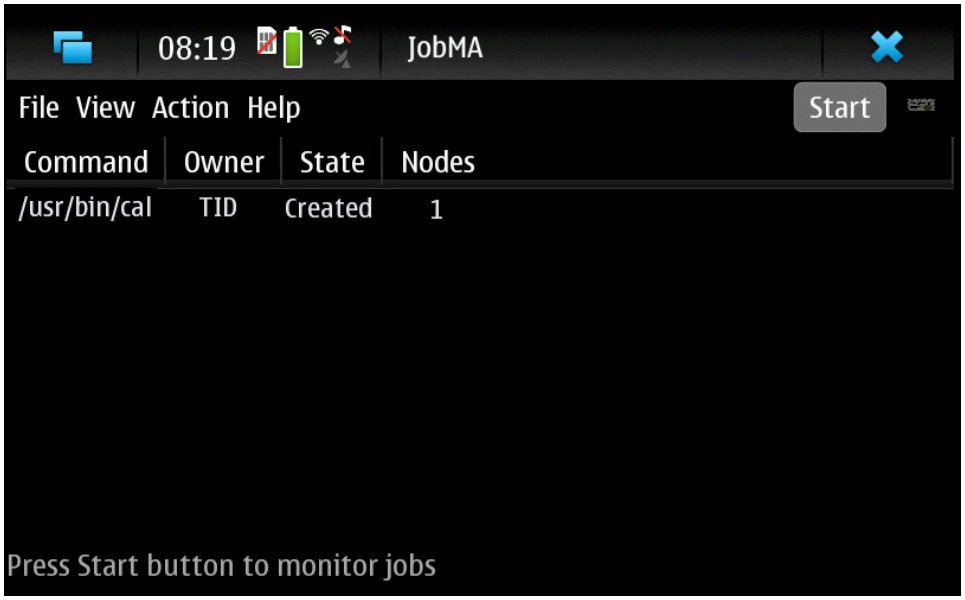
After introducing the password, the JobMA user interface will appear on the X-terminal (not needing to load manually the certificate in this version)

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.



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.

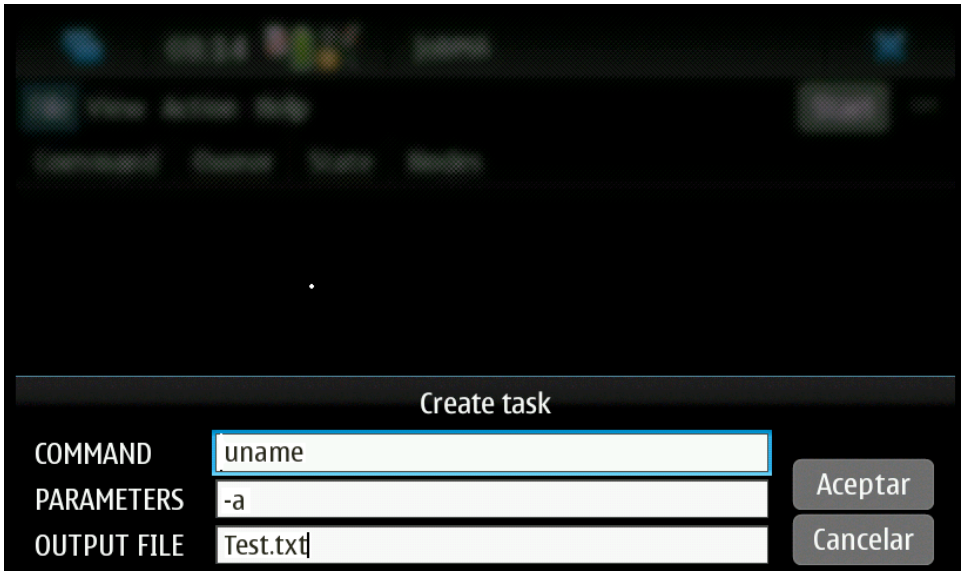
Set Up No special configuration after the installation is needed.

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: "uname" is the command to show name and information about current kernel.
- Parameters: is the specification that we want to add to the job. E.g: "-a" in the previous command will show basic information currently available from the system.
- 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 successfully.

Procedure Results The execution of this test was successful. The JobMA GUI was correctly presented and a JSDL file was loaded using it.



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 N900 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 jobs 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.

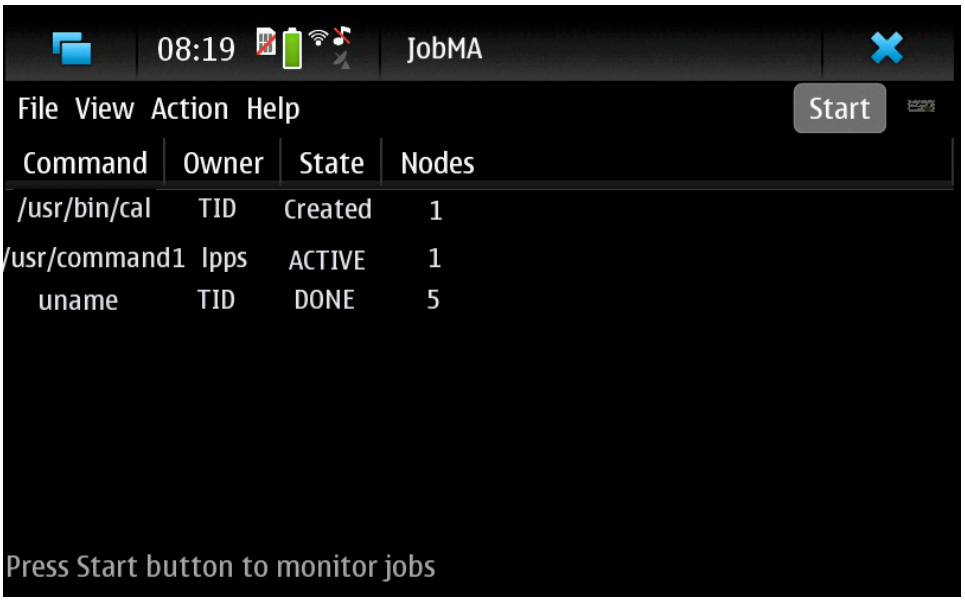
Set Up No special configuration after the installation is needed.

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 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.



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 N900 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:

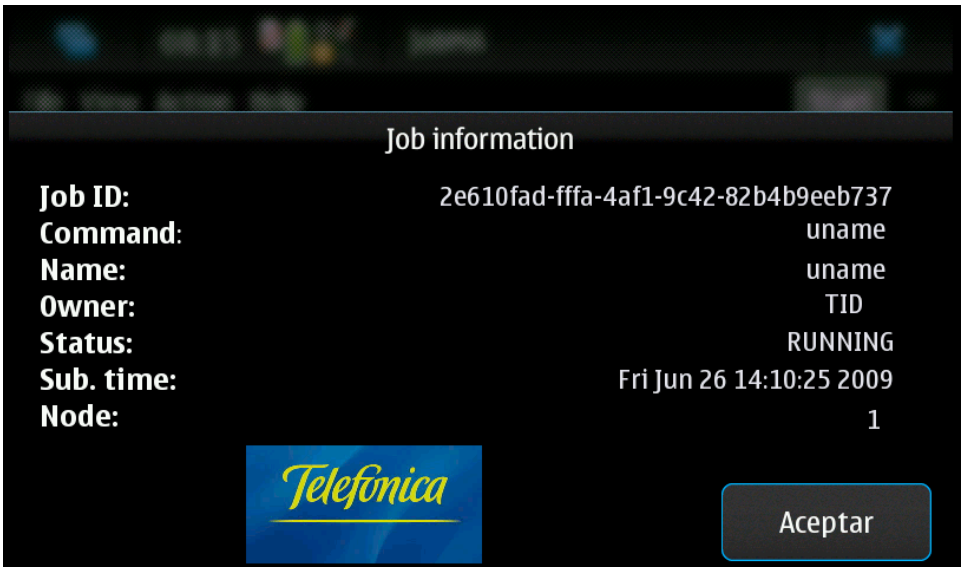
Installation Once installed XtreemOS-MD on the mobile device used, JobMA application should also be installed in the terminal following the installation and configuration instructions.

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.



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 N900 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.

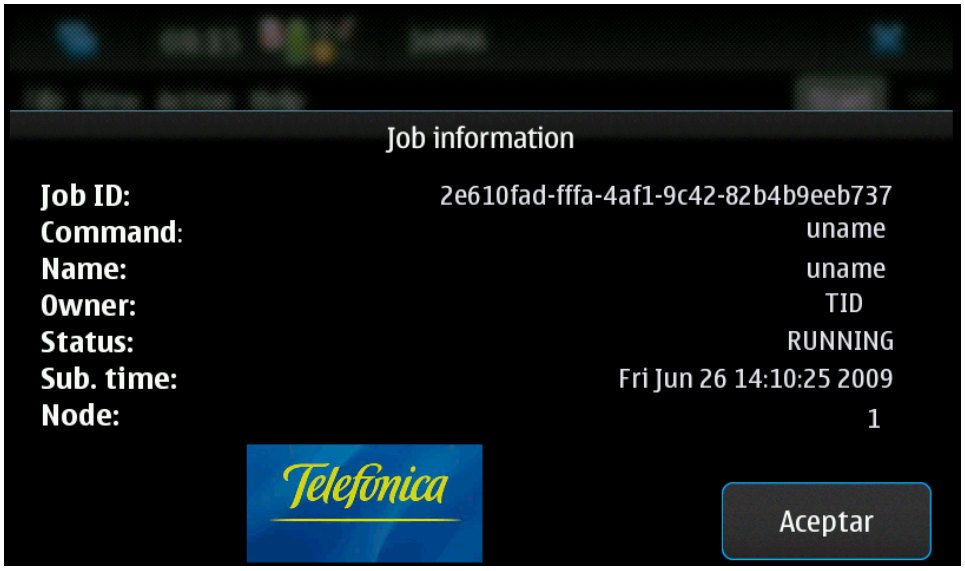
Set Up No special configuration after the installation is needed.

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.



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 N900 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 N900 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 N900 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: IMA and XtreemFS integration

4.11.14.1 Responsibilities

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

4.11.14.2 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.

4.11.14.3 Test Results

The test was run following this procedure:

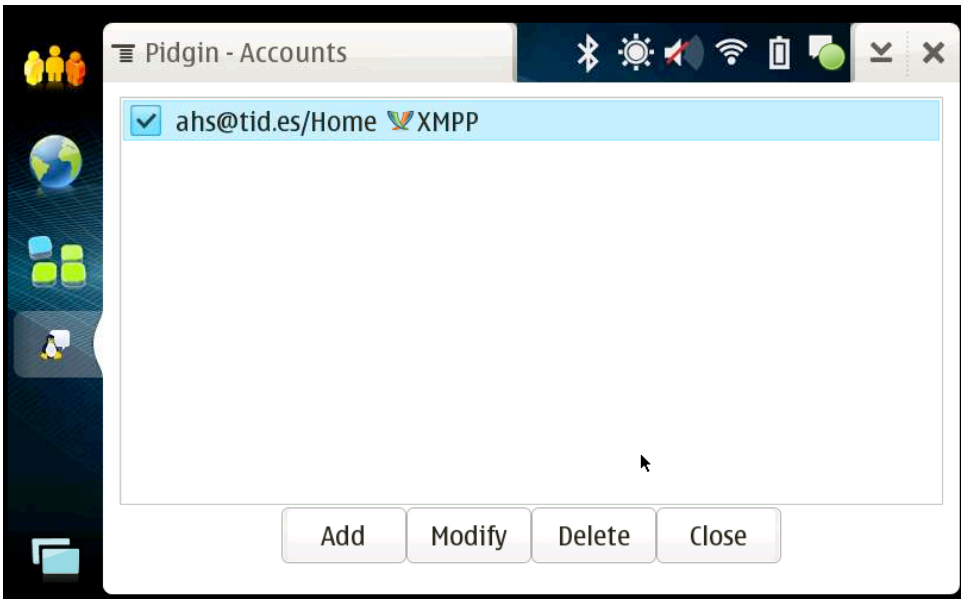
Installation Once installed XtreemOS-MD on the mobile device used, IMA application should also be installed in the terminal following the installation and configuration instructions.

Set Up Login in the Grid with a user/password valid.

Start The user will start the IMA application and will configure a new jabber account. Then, he will connect using the new account and will launch a conversation with one of his contacts. Finally he will log out.

Then he will use a different mobile device, with XtreemOS-MD and IMA application installed on it. After login in the Grid with the same credentials 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 successfully. After moving to a different device, 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.



4.11.15 Test Unit 14: Using XtreemOS-MD for sharing 3G connection

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

Test Items
3G sharing feature of XtreemOS Mobile device.

Features to be Tested
This test case tests the connection between a XtreemOS-PC and a XtreemOS-MD to share the Mobile 3G connection.

Approach Refinements
The goal of this test is to demonstrate that it's possible to share the 3G connection of mobile devices with others XtreemOS clients. A Nokia N900 with XtreemOS-MD and PC with XtreemOS-PC pre-installed will be used for this test.

4.11.15.3 Test Results

The test was run following this procedure:

Installation XtreemOS-MD must be installed on the mobile device. In addition for this test, it must be used a PC client that requests a 3G connection to the Mobile device. XtreemOS-PC must be installed on the PC and later the `sharing3gclient` must be installed on the PC following these steps:

- download the package from xtreemOS-MD web
- urpmi sharing3gclient-1.0-2.i386.rpm

Set Up No special configuration after the installation is needed on PC or mobile device.

Start The first step is to enable the 3G sharing feature from RS-Monitor application on the Nokia terminal. This is done bye clicking on the *Network sharing* tab of the RS-Monitor and then modify the Current status by clicking on the *modify*. After doing this, status d̈isabledc̈hanges from the previous state to ënabled,̈ so the 3G connection sharing is now started from this mobile device.

The second step is to execute, from XtremOS-PC, the `sharing3gclient` in a standard shell as root: `root@xos-pc1:#sharing3gclient`

If everything goes OK, a request from the PC client will be sent to the mobile device and will generate a VPN connection between the PC and the Mobile Device. All web traffic of PC client will be forwarded to the mobile device (Nokia N900).

Procedure Results The execution of this test was successful. All web traffic of the PC is routed by the Mobile device 3G connection.

4.11.16 Test Unit 15: Using XtreemOS-MD for data sharing

4.11.16.1 Responsibilities

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

4.11.16.2 Test Specification

Test Items
Data sharing feature of XtreemOS Mobile Device.

Features to be Tested
This test case tests the sharing of data from a mobile device between XtreemOS clients.

Approach Refinements
The goal of this test is to demonstrate that it's possible to share data from mobile devices within the XtreemOS file system. Two Nokia, N900 and N800, with XtreemOS-MD will be used for this test.

4.11.16.3 Test Results

The test was run following this procedure:

Installation Install XtreemOS-MD on the mobile devices used. Install camera application on the Nokia N800: *Camera for Nokia N800/N810*.

Set Up Using the Resource Sharing Configuration tool, *ConfigRS*, on the Nokia N800, enable the Data Sharing feature and set the shared folder path with the camera application's working directory, *user/multimedia/camera*.

Start The first step is to take a photo with the Nokia N800. A new file is created in the working directory of the camera application and it is shared through the XtreemOS file system.

The second step is to execute the *filemanager* from Nokia N900, open the share folder and find the new file.

Finally, the third step is to open this file from Nokia N900 and the content is uploaded from Nokia N800 to Nokia N900 through the Xtreemos file system. In Nokia N800 the *RSmonitor* icon changes its state showing that data transfer was executed.

Procedure Results The execution of this test was successful.

4.11.17 Test Summary Report

4.11.17.1 Summary of Tests and Results

The tests executed have been passed without major issues. The functionalities developed in XtreemOS-MD and the applications have been verified. The tests

show that XtreemOS-MD performance is comparable to the XtreemOS PC flavour, in terms of protocols behaviour.

4.11.17.2 Conclusion and Directions for Future Work

This is the final test suite performed on XtreemOS-MD. The general conclusion is that XtreemOS-MD brings Grid functionality to Linux-based mobile devices. The software developed may be exploited in any Linux-based terminal running on top of a XtreemOS core Grid.

Chapter 5

Evaluation of XtreemOS as Foundation for Cloud Computing

This chapter is devoted to evaluating the benefits of using XtreemOS technology as a basis for Cloud Computing. Whereas deliverable D4.2.6 [10] put emphasis on Grid Computing aspects (comparing XtreemOS with competitive Grid Middleware solutions), the deliverable at hand tries to assess the capability of XtreemOS for enabling (or supporting) the management of cloud environments and the development of cloud services.

5.1 Evaluation of Cloud System Management Automation and Recoverability Enhancement with XtreemOS

The purpose of this evaluation is to assess the contributions that XtreemOS technologies can make to automation and recoverability[1] of infrastructure and general IT systems management . A research prototype for Rule-based and Unified Systems Management has been developed by SAP Research during the XtreemOS project. The motivation for this research prototype is relevant to XtreemOS, as it addresses the challenge of maintaining heterogeneous hardware, operating systems, data bases and software. This heterogeneity results in a collection of various scripts, tools and models for management . These increase the likelihood of error, conflict and of loss of know-how when administrators leave an organisation. Moreover, systems that contain many nodes, software instances and files can become particularly challenging to maintain without automation. Although cloud computing offers organisations the possibility to out-source the hosting of machines, software and data, the sprawl of machine instances and cloud providers can still become unmanageable. Figure 5.1 shows the overall architecture of the Rule-based System Management (RBSM) solution used in this evaluation.

[1] There is no assumption of self-healing, self-balancing, and other self-* features of autonomic computing

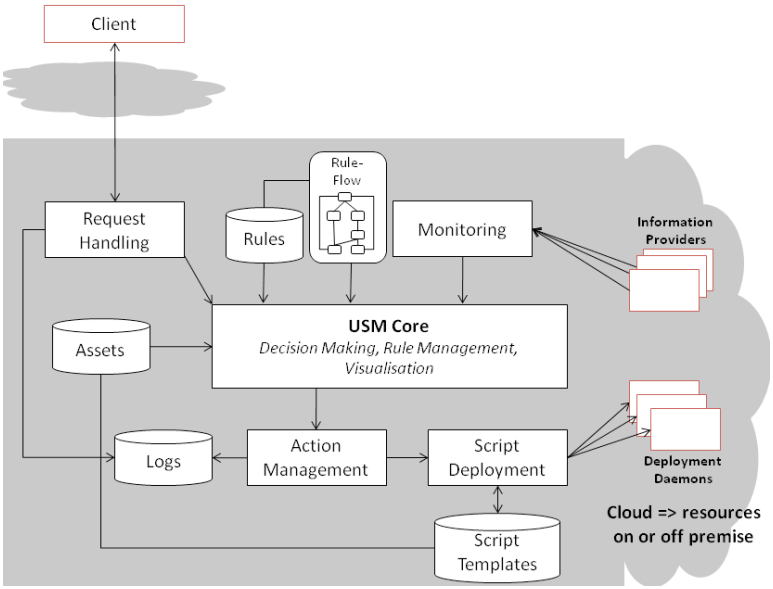


Figure 5.1: High-level component architecture of RBSM used for management of on-premise and cloud-based hardware, software and database instances, implementing a Unified System Management (USM) solution

In this evaluation, the concept of "Cloud" includes on-premise (private) and off-premise (publicly shared) resources . An organization's IT administration would use RBSM to support deployment decisions, based on factors such as cost, performance and security, towards determining which resources should be used for specific scenarios . For this reason it is also referred to as a Unified System Management (USM) solution. In cases of off-premise clouds there is no assumption made that XtreemOS is used for the management. However, as the organisation's data-centre can itself be deployed and managed using a cloud model, the advantages of operating aspects of the systems management solution using XtreemOS technologies is reasonable.

Clients include customers, users of the organisation or remote administrators. Clients will not tolerate extended downtimes and slow responses when requesting resources (e.g. virtual machines) to be provisioned and deployed. The *Request-Handling* component is responsible for receiving and interpreting requests from Clients. It also performs scheduling of passing these requests to the USM Core for handling . The USM Core is the central engine of the management solution. It uses the rules and various information sources in order to automatically decide how to best handle requests, or supports decision making through generation of visualizations[2] .

Rules used for decision-making are of the form `IF <conditions> THEN`

[2]Visualisation is often a requirement from IT administrators even with automation, as they prefer to have some oversight of the system's state. The ability to visualise the system is in any event also a subset of computer supported system management.

<actions>. These are processed by a rule engine, such that they can be updated offline. The concept of a *rule-flow* is used to separate concerns of systems management. For example, configuration of an element cannot occur unless the element has been installed. The configuration data for elements are stored in the collection of *Assets*. This includes the properties of the elements, their commands that can be executed on the elements and how they can be manipulated. This data is important for configuring the *Monitoring* subsystem and the *Information Providers* that run natively to monitor specific elements.

The *Action Management* component is used to coordinate the mapping of actions to executable scripts in the repository of *Script Templates*. This repository also includes VM images and other resource bundles required for installation and configuration of specific types of elements. The *Script Deployment* mechanism is responsible for transmitting a concrete script instance to the respective *Deployment Daemon*, which is a native process on an element that executes installation and configuration scripts. All actions and their feedback are captured in the system's *Logs*. The goal of the evaluation is to determine how XtreemOS' features can be used to enhance the automation and recoverability of management tasks.

5.1.1 Responsibilities

The responsibilities for this evaluation have been assigned to Philip Robinson and Barry McLarnon from SAP Research in Belfast. They are the principal designers and developers of the systems management solution. Barry has been the main programmer of the project and has implemented and setup various XtreemOS demonstrators surrounding RBSM. Philip designed the original concept and continues to guide the architecture and roadmap of the RBSM project. The project is currently creating transfers of its object models and automation concepts within SAP. Given that the idea for developing the solution was inspired by developments within XtreemOS, this indicates a success for exploitation.

5.1.2 Evaluation Setup

Three scenarios were used to carry out this evaluation. These scenarios follow from previous XtreemOS demonstrations developed with RBSM for XtreemOS. The capability of RBSM has been enhanced since the previous demonstrations but the basic features and architecture are still valid. Figure 5.2 shows the setup of RBSM for the evaluation.

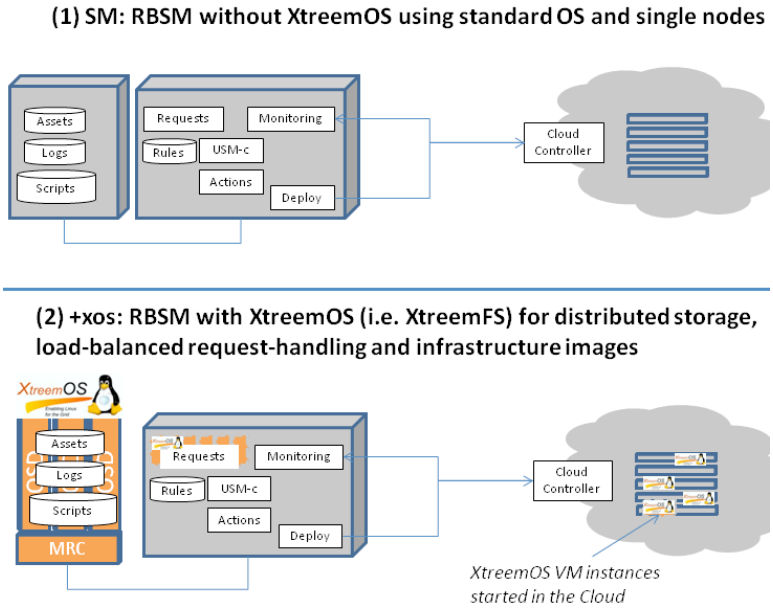


Figure 5.2: Technical set up of RBSM Evaluation

The initial configuration of RBSM is shown in Figure 5.2 (1) shows the typical setup of RBSM for managing deployment on a Cloud infrastructure. The data such as Assets, Logs and Scripts are persisted on a standard filesystem. The main components of RBSM are then installed on a single machine. The *Cloud Controller* is in our local data centre and is based on the Open Nebula [31] framework. The physical infrastructure in the data centre is a collection of blade servers, but the details of these are not important for the evaluation.

Figure 5.2 (2) shows the set up of RBSM using XtreemOS features. In addition, the scenario was a deployment of multiple XtreemOS VM images, in order to have an isolated XtreemOS grid of virtual machines running within the cloud infrastructure. The persisted elements of RBSM were placed on the XtreemFS filesystem with the motive of improving the resilience of accessing critical management data. Loss of access to this knowledge would cause a complete shutdown of management operations. The results of this evaluation can help to inform developers and administrators of systems management solutions.

5.1.2.1 Methodology and Metrics

The evaluation methodology was a mix of hands-on usage development of scenarios for systems management, as well as architectural analysis. With the hands-on development and execution of systems management scenarios, it was possible to practically identify where the challenges are and gain experience with the critical needs for automation and resilience in systems management. The approach to use architectural analysis was inspired by our work on analysing faults in automated

system management [22]. The fundamentals of the methodology are in four stages:

1. Identify scenarios that capture the core features of the system under analysis.
2. Identify and classify locations where information and state are maintained.
3. Identify and classify management tasks that cause information to be transferred and state to be changed.
4. Assess the impact on information transfer and state change if tasks are not completed based on selected metrics or indicators.

Three scenarios have been selected in order to assess the core features of RBSM as well as the relevant features of XtreemOS that have been identified as having potential impact on the systems management problem. The selected indicators are the Level of Automation (LoA) and Level of Recoverability (LoR) for systems management tasks. The LoA indicators are adopted from the taxonomy of automation levels defined by Parasuraman, Sheridan and Wickens [33], defined as the level of assistance a computer or automated solution (i.e. RBSM) offers to a human (i.e. administrator):

1. (Lowest) no assistance from computer — purely manual
2. set of decision/action alternatives offered
3. computer narrows skeleton down
4. computer suggests one alternative
5. execute suggestion if approved by human
6. allow time before automatic execution
7. execute automatically then informs human
8. informs on request
9. informs if determined that it is necessary
10. (Highest) complete execution — fully automated

There is however no existing taxonomy for LoR indicators, such that we developed our own. The indicators consider the capability of a solution to detect faults and to then respond to them. These are as follows:

Capability to detect:

0 Manual timeout

1 Implicit/indirect alert (i.e. by observing other tasks or events)

2 Implicit/indirect prediction

3 Explicit timeout (i.e. refers to specific task)

4 Explicit alert

5 Explicit prediction

Capability to respond:

0 Interrupted restart (i.e. other tasks and managed elements are affected)

1 Interrupted rollback

2 Interrupted redirect (i.e. requires redundancy)

3 Isolated restart

4 Isolated rollback

5 Isolated redirect

The summation of detection and response capability gives a rating for LoR, where 0 (i.e. Manual timeout and interrupted restart) means that the automation solution offers no support for recovering from faults. The human administrator has to manually identify that the task is not reaching completion and just re-executes that task. The scenarios are now defined, showing how the set of management tasks analysed were derived.

5.1.2.2 Scenario 1: Basic Cloud Deployment

The first scenario considers the simplest use case for RBSM in a cloud environment, where the automation of deploying a single machine image on a cloud infrastructure is implemented. This basic scenario is fundamental for administrators of XtreemOS deployments, which might be composed of internal and external instances. In this particular case the image deployed to the cloud was an XtreemOS core node image. This also demonstrates setting up XtreemOS in different types of operational environments.

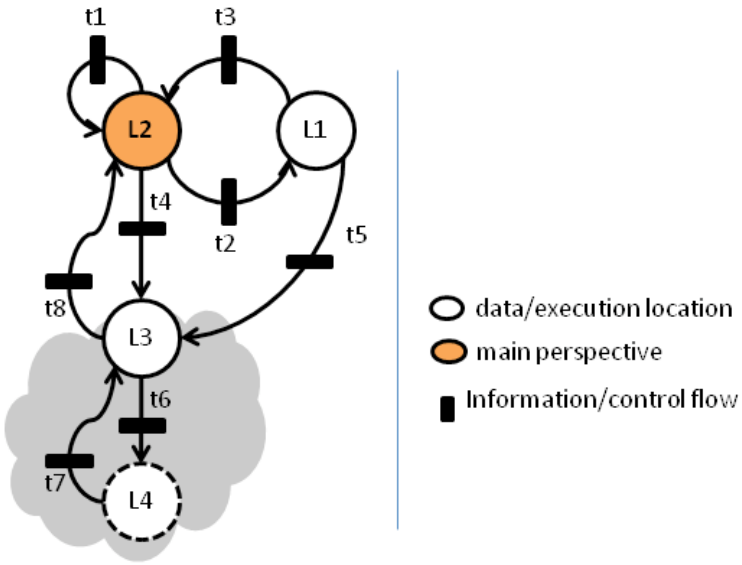


Figure 5.3: Information and control flow for basic cloud deployment

Locations L1 – L4:

L1: storage location for asset DB with images and templates

L2: management core

L3: cloud controller

L4: virtualization manager and VM host in cloud

Management tasks t1 – t8:

t1: specification of VM creation command at management core

t2: search and selection of appropriate image in asset DB

t3: transfer of image description data to the management core

t4: invocation of create-vm API on cloud controller API

t5: transfer of image to Cloud controller

t6: creation of VM on selected host and selected/specified VM manager

t7: feedback and query response of VM state to controller that VM creation is complete

t8: feedback to deployment management that VM creation is complete

It is assumed here that the management core and cloud controller are on different hosts. They can also be in different domains. The management core may be managing on-premise and cloud-based machines.

5.1.2.3 Scenario 2: Checkpointing and Migration

This second scenario involves automation of checkpointing and migration of selected processes or VMs. For the checkpointing of process instances the node-level BLCR integrated in XtreemOS was used.

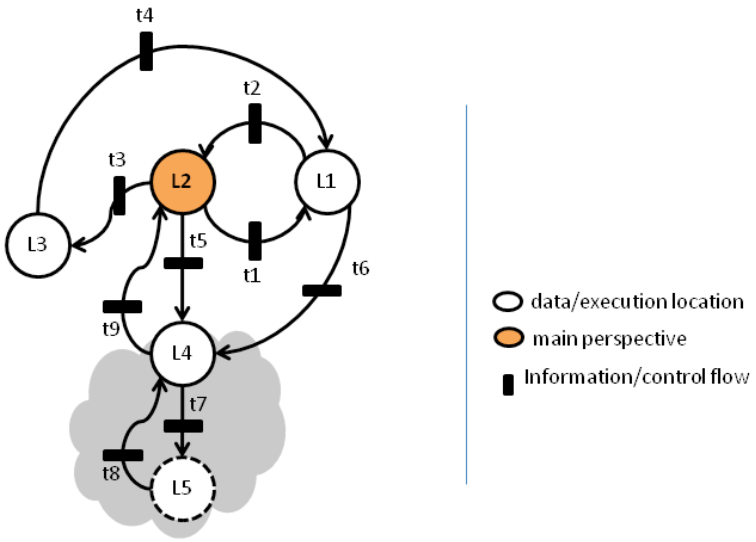


Figure 5.4: Information and control flow for checkpointing and migration

Locations L1 – L5:

L1: storage location for checkpoints

L2: management core

L3: current host of processes or images to be checkpointed

L4: target cloud controller

L5: host in cloud where checkpointed image is restarted

Management tasks t1 – t9:

t1: query/selection and configuration of volume for checkpointing

t2: query/update storage volume with checkpoint meta-data

t3: invocation of checkpoint action

t4: execution of checkpoint and transfer to checkpoint volume location

t5: issue migration and restart of checkpoint to cloud controller

t6: retrieve checkpoint data from designated volume

t7: restart checkpoint instance on a selected endpoint

t8: return checkpoint status to controller

t9: get checkpoint status from controller

However, note that the node-level checkpoint and restart can only work if there host architectures are identical, such that these prerequisites would have to be issued to the cloud controller. For that reason the above scenario refers to a private cloud.

5.1.2.4 Scenario 3: Deployment of Distributed Virtual Infrastructure with Dependencies

In this scenario the management core of RBSM acts as a cloud broker. It receives management requests from external clients and can select to handle these using internal or remote resources. The location of resources is however abstracted for this evaluation. Secondly, this extends scenario 1 as the emphasis is now on deployment of multiple, dependent elements, forming a virtual infrastructure. This means that it is necessary to deploy elements in a specific order, as dependencies of one on another will cause installation and configuration errors to propagate. XtreemOS was designed for setting up distributed systems such as Clusters, Grids and Clouds. This scenario also investigates the additional tooling that administrators could employ when managing large-scale XtreemOs deployments.

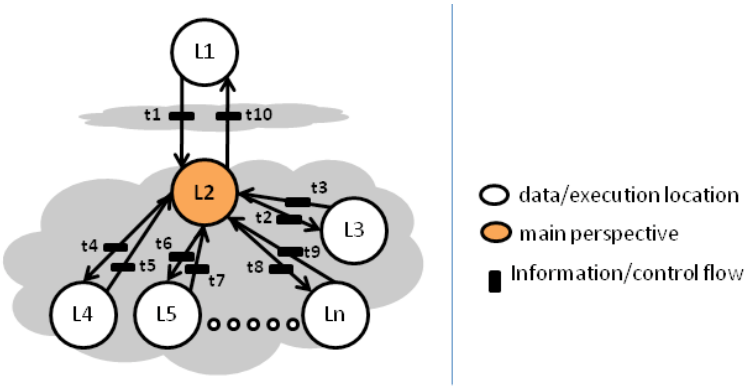


Figure 5.5: Information and control flow for deployment of distributed virtual infrastructure

Locations L1 – Ln:

L1: remote client sending request

L2: management core

L3: internal storage for images, templates and scripts

L4: host of first instance in virtual infrastructure

L5: host of intermediate instance in virtual infrastructure

Ln: final node to be deployed in the virtual infrastructure

Management tasks t1 – t8:

t1: specification of VM creation command at management core

t2: search and selection of appropriate image in asset DB

t3: transfer of image description data to the management core

t4: invocation of create-vm API on cloud controller API

t5: transfer of image to Cloud controller

t6: creation of VM on selected host and selected/specified VM manager

t7: feedback and query response of VM state to controller that VM creation is complete

t8: feedback to deployment management that VM creation is complete

t9: feedback and query response of VM state to controller that VM creation is complete

t10: feedback to deployment management that VM creation is complete

The overall specification of the virtual infrastructure was done as a batch of requests. This works by sending them as a linked list. There is no assumption of parallel tasks for batches, but this could also be inferred in the scheduling, if the dependencies are explicitly defined.

5.1.3 Evaluation Results

The results for the evaluation are now presented per scenario.

5.1.3.1 Evaluation of Scenario 1

Figure 5.6 shows the results of comparing automation and recoverability for RBSM with and without XtreemOS. The results in Figure 5.6 (A) show that there are no advantages for additional automation achieved with XtreemOS, as much of the configuration management features that exist in XtreemOS are achieved using other mechanisms in RBSM. There are however advantages for recoverability detected, as shown in Figure 5.6 (B).

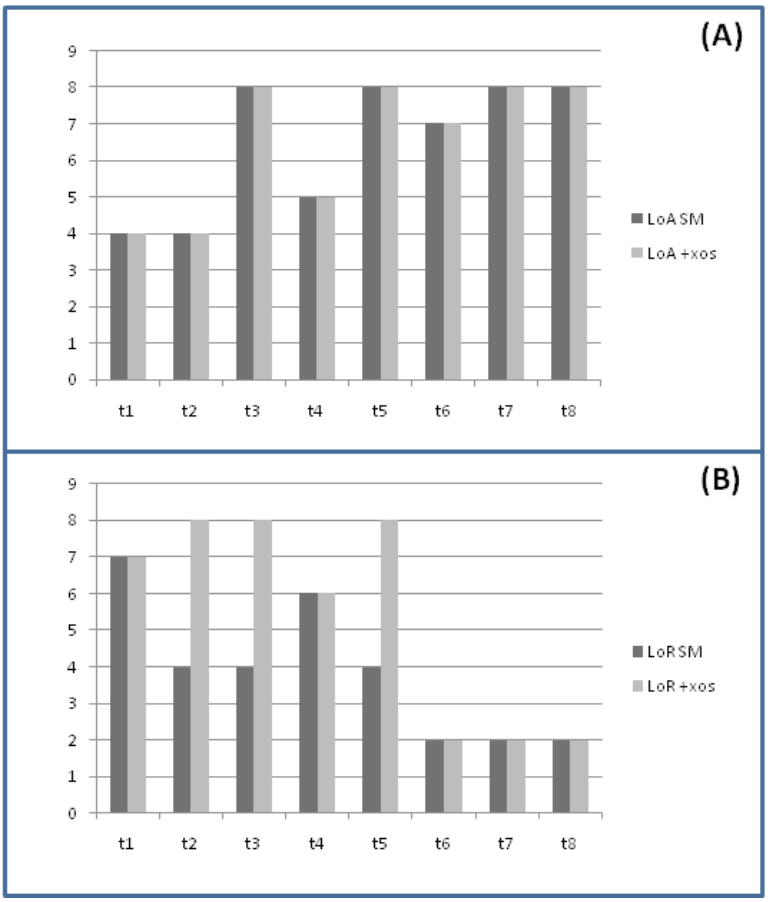


Figure 5.6: Evaluation of (A) Level of Automation (LoA) and (B) Level of Recoverability (LoR) of RBSM for basic delpoyment tasks t1 to t8. The comparison is between RBSM without XtreemOS (SM) and with XtreemOS (+xos)

Tasks t1 and t2 cannot be automated beyond level 4 (suggests one alternative), as these explicitly require human input in order to specify the need to create a VM. This could be changed if the same VM needs to be installed several times, such that his request could be captured in a script. Tasks t3, t5, t7 and t8 were however capable of being supported with LoA 8, where RBSM informs the administrator about how the request is being handled if the administrator requests this to be the case.

From the perspective of LoR, task t2, t3 and t5 show significant advantages for XtreemOS underlying RBSM. These tasks are each storage-dependent tasks, such that the unavailability of the centralized storage node in (A) will block these tasks. It is therefore suggested to employ a distributed storage architecture like XtreemFS in order to have better guarantees of correctness and timeliness for automated management tasks.

5.1.3.2 Evaluation of Scenario 2

Figure 5.7 shows the results of comparing automation and recoverability for RBSM with and without XtreemOS for the checkpointing and migration. The results in Figure 5.7 (A) show that there are some advantages for additional automation achieved with XtreemOS, unlike in scenario 1. Advantages for recoverability were also observed when using XtreemOS, as shown in Figure 5.7 (B).

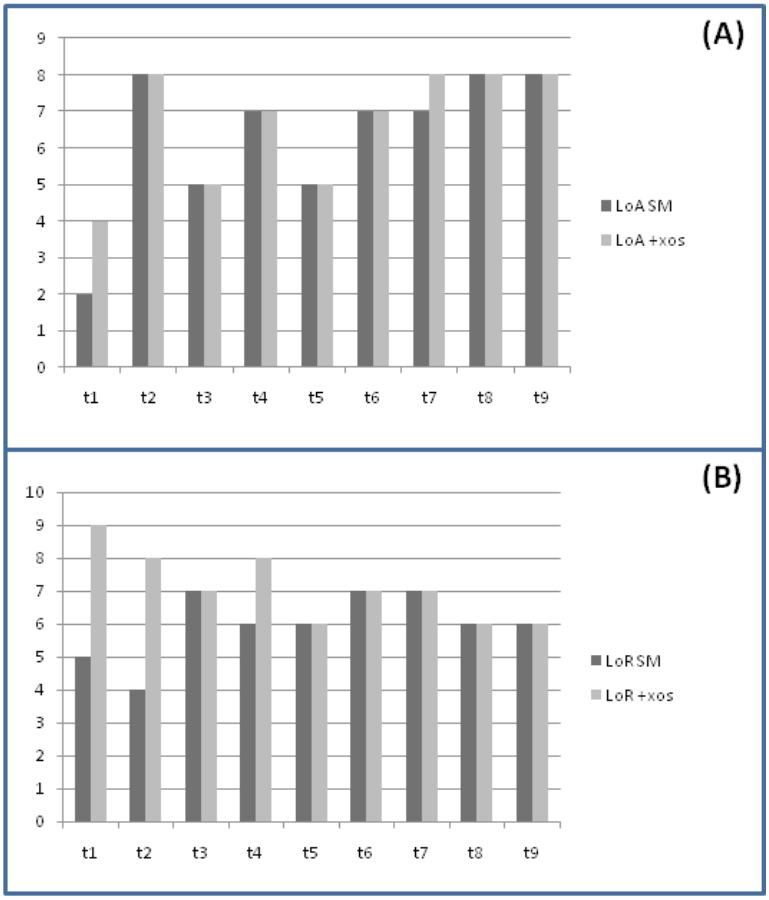


Figure 5.7: Evaluation of (A) Level of Automation (LoA) and (B) Level of Recoverability (LoR) of RBSM for checkpointing and migration, tasks t1 to t9. The comparison is between RBSM without XtreemOS (SM) and with XtreemOS (+xos)

Using XtreemOS features increases the automation of t1 and t7, given the usage of the distributed file system for storage of checkpoint data. Given that an automatically-scalable volume is mounted, there is then no need for the administrator to explicitly select the storage location and check that it has the required capacity. Tasks t1, t2 and t4 are enhanced with respect to their recoverability given the improved resilience brought by XtreemFS.

5.1.3.3 Evaluation of Scenario 3

Figure 5.8 shows the results of comparing automation and recoverability for RBSM with and without XtreemOS for the scenario of deploying distributed virtual infrastructure. The results in Figure 5.8 (A) show that there are minor advantages for additional automation achieved with XtreemOS. Two tasks benefit from enhanced recoverability, as shown in Figure 5.8 (B).

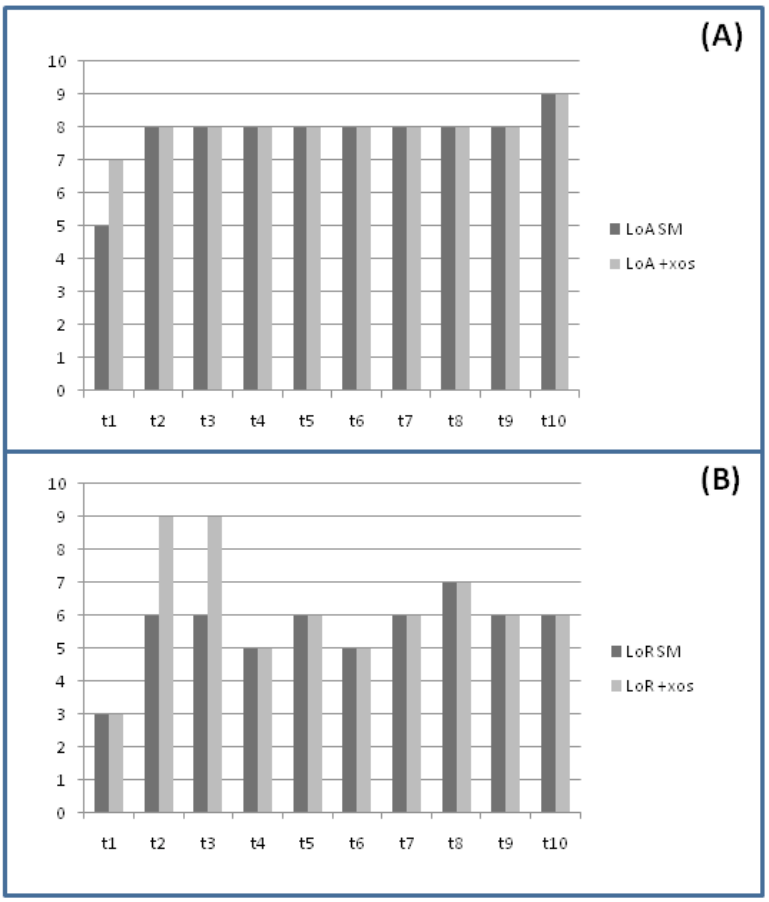


Figure 5.8: Evaluation of (A) Level of Automation (LoA) and (B) Level of Recoverability (LoR) of RBSM for deployment of distributed virtual infrastructure with tasks t1 to t10. The comparison is between RBSM without XtreemOS (SM) and with XtreemOS (+xos)

Tasks t2 and t3 are again storage operations, such that the usage of XtreemFS as our underlying filesystem yielded an increase in the overall recoverability.

5.1.4 Summary

Automated systems management can support data centres in becoming more efficient. Labor costs reduction leads to significant reduction in overall costs of ownership. However, without resilience and recoverability implemented in the management system, or built as a feature of the operating environment, the benefits of automation are degraded.

5.2 Evaluation of an Online Photo Archive as Cloud Deployment using XtreemOS

5.2.1 Test Plan

5.2.1.1 Responsibilities

WP4.2, Maik Jorra (ZIB), Björn Kolbeck (ZIB)

5.2.1.2 Test Items

The main purpose of the *Online Photo Archive* demonstrator (referred to as Zmile) is to proof that XtreemOS can be utilized as platform for a state of the art web application. In addition to just host Zmile on an XtreemOS environment, we also use XtreemOS features to improve our web-application.

Using XtreemOS yields two gains for our application: When using XOSAGA and XtreemFS in the same application one has the possibility to perform all calls to XtreemFS in a system-independent way through the XOSAGA-API or simply to mount a XtreemFS volume via a fuse client, and access it as was part of the local file system.

XOSAGA would mount the XtreemFS volume when a file is requested, read the file from the volume and unmount it. It is very comfortable if one would read a few files and does not want to bother where the files are located. However, in Zmile reading and storing of files is the major purpose of the application and additionally we want the image processing tools to be able to have direct access to the images files on „disk“. For these reasons, we decided to mount our XtreemFS volume manually and have Zmile access it as if it was a folder. This also has the advantage that it shows, how a web-application which was not designed for XtreemOS can profit from XtreemFS, thanks to its transparent nature and POSIX conformity.

The second advantage is that we can use the XOSAGA-Job-adapter to execute required jobs in a platform independent way. See Section 5.2.1.4 for more information.

5.2.1.3 Features to be Tested

As said Zmile is a proof of concept. To show that Zmile works, it is sufficient to demonstrate that users can store and upload images.

Additionally, we show that the memory can be increased by adding a new OSD to the XtreemFS pool which registers itself to our directory service, leads to a higher amount of memory Zmile can use, while the node running Zmile needs not be touched.

We also show that configuring the XOSAGAs SSH adapter we can run jobs on different hosts, which have the Zmile public key installed and must have access to our volume.

5.2.1.4 Features not to be Tested

The AEM job service. In the first draft of Zmile it was desired to use AEM to submit jobs. Jobs in this case are image transformations like scaling, rotating and thumbnail-creation.

These tasks can be accomplished in two ways: The first would be to use a Java image-processing-library like *Jai* which would execute this operation directly in the server process, in another task. Hence this approach will not scale well, we used the second option, which is using an external application for image manipulation. We choose the ImageMagick program suite, because it is available on most Linux boxes and is easy to install on XtreemOS.

The following workflow triggers the execution of ImageMagick commands: Every time an image is displayed to a user it is checked if the image has the desired size, e.g. when the user opens an overview of album contents, images need to be displayed as thumbnail, for each image Zmile checks if a thumbnail is available, if this is the case its send to the browser, if not, it is converted into a thumbnail by an ImageMagick command and then again the result is returned immediately to the user.[3]

The fact that the result is required as soon as possible does not leave much room to schedule the conversion job. Using the AEM service to submit jobs yielded an simple problem: On the one hand AEM is used to schedule jobs which are expected to have a long runtime and may require more than one CPU-core to run. On the other hand ImageMagick commands have a very short runtime, need very few resources and the result must be available ASAP. In other words Zmile does not provide the kind of job you want to send to an job-scheduler no matter its AEM, PBS or Condor.

So we abandoned the idea of using AEM instead we are using the XOSAGA-SSH binding and gain the possibility to either run ssh commands on the localhost or send jobs to arbitrary compute servers which need not necessarily run XtreemOS, but need access to the OSDs.

[3] After a thumbnail is created it will be stored on XtreemFS along with the original picture

5.2.1.5 Overall Approach

XtreemOS 2.1 is used for all of our tests. We run XtreemOS with Zmile on an out-dated workstation with the following specs:[4]

CPU 2x Intel Xeon 2.20GHz 32-bit

RAM 1GB

HDD-Storage approx 60GB

We ran another XtreemOS instance in an virtual machine on our ZIB internal Cloud setup [13], which is hosted on a dedicated cluster consisting of 32 nodes called *cumulus*. The nodes have access to our storage pool. The virtual XtreemOS node is called *vmhost01* and runs an OSD which is connected to the directory-service running on the Zmile-machine. The OSD provides additional 115 GB which zmile is using to store images.

There is an another more powerful workstation in our setup which accepts the XOSAGA-SSH commands and executes image conversions.

CPU 2x Intel Core2 2.66GHz 64-bit

RAM 4GB

We choose this setup to show how a low-cost server with little compute power is extended by storage from a cloud environment and other nodes which offer additional compute power. Instead of the second workstation we could have sent the SSH requests to another *vmhost* node.

The fist test run will show an extension of the cloud nodes, the second test will show the benefit from the additional compute node.

This is the setup for the tests, all further details about the testing methodology can be found in the following Test Unit sections.

5.2.2 Test Unit 01: Storage Extension

5.2.2.1 Responsibilities

See 5.2.1.1.

5.2.2.2 Test Specification

Test Items

This test aims at the file-system of XtreemOS. If XtreemOS is not used one could download XtreemFS from `http://www.xtreemfs.org`, this site provides the latest release as well as the required documentation.

The following components on XtreemFS were used:

[4] We'll call this the Zmile-machine in the following.

- xtreemfs-backend-1.2.0-3.1.i586.rpm
- xtreemfs-client-1.2.0-3.1.i586.rpm
- xtreemfs-server-1.2.0-3.1.i586.rpm
- xtreemfs-tools-1.2.0-3.1.i586.rpm

Features to be Tested
Scalability of XtreemFS in cloud scneario.

Approach Refinements
We used the test set-up described in Section 5.2.1.5 with a running Zmile-server. The test consists of the following steps:

1. we use the `df` command to acquire the available size of the zmile image directory.
2. Then we launch a new instance of the XtreemOS-VM running on *vmhost01*.
3. When the VM is on-line we call `df` on the zmile-server again.

We expect a doubling of the available storage space, since the image has the same configuration and provides additional 115GB of space.

This test will show how easy and without reconfiguration the storage can be extended. The instantiation of the new VM could be executed automatically by the cloud management software.

5.2.2.3 Test Results

Before we show the results it should be mentioned that the tests were executed correctly without any hurdle. This includes the set up of the test sytem from the installation of XtreemOS to the upgrade of the XtreemFS version.

Here is the first output of the `df` command, the Zmile volume is mounted on /var/zmile:

```
1030 zmile@csr-pc28 ~ > df
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda1             5,8G  2,6G  3,0G  48% /
/dev/sda3              66G  9,6G   53G  16% /home
xtreemfs              116G   27M  116G   1% /var/zmile
```

Now after the second machine is up:

```
1030 zmile@csr-pc28 ~ > df
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda1             5,8G  2,6G  3,0G  48% /
/dev/sda3              66G  9,6G   53G  16% /home
xtreemfs              232G   27M  232G   1% /var/zmile
```

As we expected the available storage is doubled see Figure 5.9.

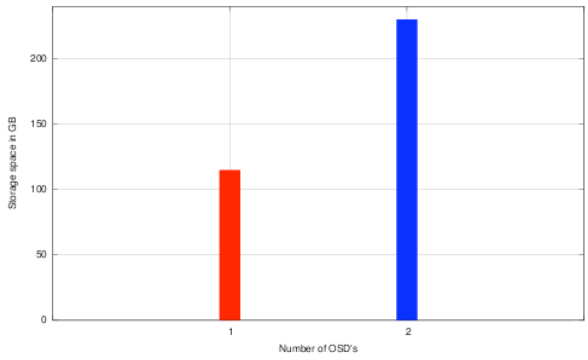


Figure 5.9: Zmile with replicated OSDs.

5.2.3 Test Unit 02: XOSAGA-SSH benefit

5.2.3.1 Responsibilities

See Section 5.2.1.1.

5.2.3.2 Test Specification

Test Items

This will show how useful the XOSAGA-SSH adapter is. The XOSAGA version was pulled from the SVN, it is trunk revision r6868.

Features to be Tested

The machine running Zmile is quite old but serves well as a web-server, however converting images can take quite some time. This we configured XOSAGA to submit jobs through the SSH-adapter to a different machine.

Approach Refinements

The set-up is the same as described in 5.2.1.5. The major image transformation Zmile performs is resizing, so we upload an image via the web interface, add it to an album and open the album view. This causes zmile to resize the image to the size of 30x30 pixel.

As the source image we use a picture which has the standard format of a normal digital camera with the following properties: R0012033.JPG JPEG 2048x1536 2048x1536+0+0 8-bit DirectClass 1.254MB

Internally, Zmile generates the following command line to resize the image:
`convert R0012033.JPG -thumbnail 30x30 r0012033_thumb.jpeg`
The `thumbnail` argument causes convert to generate an image of lower quality and runs pretty fast.

When an image is viewed it is resized to fit into a box of 600x600 pixel. Here the command
`convert R0012033.JPG -resize 600x600 r0012033_small.jpeg`
is used. The `resize` argument causes convert to create an image of good quality which takes longer than the thumbnail creation. The runtime of the command increases when the size decreases.

In order to compare execution of the jobs on the machine running the Zmile service with the execution using XOSAGAs SSH-adapter, we wrote an XOSAGA mock-up which executes the command using a `System.exec`-call on the server machine. Then we replaced our mock-up with the real XOSAGA-SSH adapter, to send the command to a faster machine. The time the execution command needs to return is measured in the Java code.

The crucial point here is that the local runtime of the command must be bigger then the remote runtime plus the time for the communication overhead. This is way we compare the thumbnail creation with the normal resize action. It is possible that only the latter can profit from the remote execution while the first fails.

The test shows how versatile XtreemOS is. We need not implement the SSH submission by ourselves, and are able to switch to another job-scheduler if this gives some advantage.

5.2.3.3 Test Results

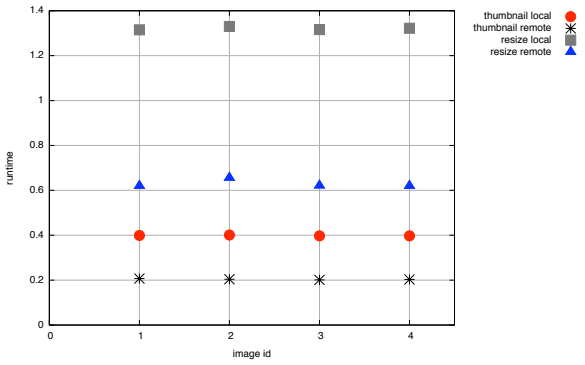


Figure 5.10: Zmile remote and local job execution.

Figure 5.10 shows that in every case the remote execution is faster than the local one. This would not necessarily be true for small images, but one can see that the usage of the XOSAGA-adapter has only a small overhead, and the big images

are the ones causing problems.

5.2.4 Test Summary Report

5.2.4.1 Summary of Tests and Results

Both tests show that its not only possible to create a web-application running on XtreemOS, it was even faster to use the provided features than developing them on our own. We are not aware of a any distributed file system (DFS) which could replace XtreemFS in our scenario, other DFS having either local boundaries like Lustre, which is for LAN use only, or doesn't scale so well like CIFS of WEBDAV. Because of the lack of alternatives one would have to create his one storage pool solution or store the images into a normal data-base. However the first solution would be time consuming and the second one would scale well, since standard OpenSource DBMS can hardly be distributed.

Second using the XOSAGA-API provides a comfortable way to generate jobs, which are executed in other processes even on different machines and again thanks to XtreemFS no additional work is needed to get the pictures to the machine which runs the ImageMagick commands.

5.2.4.2 Conclusion and Directions for Future Work

As developer of the Zmile application I have to admit that I would use XtreemFS and XOSAGA in future web-applications as well.

The possibility to increase the storage size dynamically is a huge advantage. In other solutions, which could use CFIS or some other remote file systems, however, you always have to register it to the server. Also the placement of the data does not require any influence from the server which would not be the case for example when you have different mount points.

Other distributed file systems have a locality constraint, in our case we can host OSDs even on different clouds and static machines around the world.

It was also shown in the test that it basically does not matter if you run XtreemOS on stationary machines or in a cloud environment. It works in both cases and does not give you problems when you prepare it as cloud image.

As stated in Section 5.2.1.4, the AEM service was not suitable for our work. XOSAGA-SSH, however, gives us some flexibility to choose the host we want to run our compute jobs on. Here, we see the possibility for some improvement in the form of load-balancing for lightweight jobs: Compute-nodes could register themselves to the server running Zmile, like the XtreemFS OSDs are doing. Currently the Zmile-machine uses a list of hosts it can send jobs to.

Chapter 6

Conclusion

In this deliverable, we presented the extended set of applications and the test documentation of the XtreemOS evaluation. The spectrum of previous applications in WP4.2 has been extended by further applications for experimental usage during the project extension phase. We briefly introduced the applications, reported on application development and porting activities done and indicated how the applications can be used for dissemination and exploitation purposes.

The evaluation was carried out in three different test categories: evaluation of installation and configuration, evaluation of XtreemOS components, and evaluation of XtreemOS as foundation for Cloud Computing. 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 tests gathered the experience of end users with seven XtreemOS releases from XtreemOS 1.0 (released end of 2008) until XtreemOS v3.0 beta 2 (tested September 2010). This allowed for tracing the evolution of user satisfaction and for transferring feedback to developers in a continuous manner. In all categories examined, one could detect an remarkable upward trend. The ratings improved almost 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 and completeness, it also corrected many errors. The separation into a user and an admin guide is highly appreciated.

Recommendations given in previous surveys were given to developers which tried to deal with them to a far extent. Also the comments and recommendations collected for the latest two releases covered by the survey were communicated. After four years of research, XtreemOS end-users acknowledged the progress made

in software installation, configuration, basic usage and documentation quality. This was expressed by ever increasing ratings in all categories. It is highly appreciated that so many recommendations and comments have been addressed which leveraged user experience.

The second category of tests consists of the evaluation of XtreemOS components including Node-level VO support, checkpointing and restart, Linux SSI, DIXI message bus, XtreemOS API, Resource Selection Service, Application Execution Management, Data Management, Security Services and the Mobile device Flavor. In the following, the results per component shall be summarized.

The tests with node-level VO support show that account mapping performs as expected in normal usage, except in cases when logged into the client node as root. We can conclude that the components for node-level VO support in XtreemOS release 2.1 are adequate, with all major functional requirements met but certain bugs still to be fixed.

The experimentation completed for checkpoint/restart functionality provided good coverage of the features provided by XtreemOS. Tests included checkpointing of openVZ containers, the Grid Checkpointing Service (GCS) and the Linux SSI checkpointing. With respect to container checkpointing the tests have shown that the checkpointing mechanisms could be executed successfully. The time taken to submit a job to an openVZ container is around 5.5 times greater than native submissions. Still the execution in a container may be profitable for applications demanding a high degree of isolation. The scalability test of openVZ checkpointing integrated into XtreemOS reveal practically no difference to the openVZ checkpointing on native Linux. Considering the test with GCS, the overhead of GCS seems to be negligible. Only in one case there are differences in the range of seconds caused by dynamic network traffic behavior and non-linear disk access for these large checkpoint file images and also seek times of the disk to read this large scattered checkpoint-images. Long restart times could be measured for restarting large images which could be explained by the bottleneck wrt. disk read and write bandwidth which is responsible for the tremendous overhead. Also the average duration of the pre- and post-checkpoint phase was measured. The pre-checkpointing phase takes longer than the post-checkpointing phase. Pre checkpointing covers the most of the work associated with channel flushing, such as channel control threads coordination, the actual channel flushing and the channel removal. Post checkpoint has a minor workload, it merely covers socket recovery and input of buffered messages which requires less coordination with control threads. No significant correlation could be found between the message length the pre-checkpoint phase time. The final tests with checkpointing on the current LinuxSSI failed because the setup of the appropriate testbed could not be prepared. Bugs were reported accordingly and it is planned to repeat the tests when fixes have been released.

The tests with the Dixi message bus were done with the Grid5000 platform. In the tests, we progressed from first making simple calls with single client on a single server. We gradually increased the number of the clients, adding to the stress level of the service. All along we noted the slow linear increase in the average

response times. We concluded that the number of requests issued by an individual client does not influence the delays, but the number of clients concurrently making service calls has the expected impact on the individual request's response time. In a special test case, we explored the influence of putting a lot of requests quickly into the message bus. Here we found that the length of the queue of messages to be processed has a noticeable influence in terms of the initial request's response time. The purpose of the DIXI was to host a number of services, connected both internally on each node and throughout the network by the built-in message bus. Ideally, the staging framework would host the front-ends of the services which process incoming service calls quickly, but moves all the lengthy and complex computation "off-line". For these purposes our tests show that the services can be hosted by DIXI efficiently. Of course it is not always practical to optimize for fast service call processing, particularly when prototyping new services, which is another strong point of the DIXI. On the other hand, the growing grids will provide an increasing service stress. In either case it is difficult to construct a framework to alleviate the slow-downs, but in this case it is better to provide a higher number of replicated services to distribute the load of the clients. We see a possibility of further enhancements and improvements for DIXI in the ability to distribute the actual load on the distributed services when the service request does not call for a particular host's service. The already built-in service call redirection could take decisions on the service message traffic to use the less-visited parts of the grid.

Tests with the XtreemOS API included tests with MPI, XOSAGA and AEM. The comparison of AEM and XOSAGA showed that the performance regarding job submission, resource management and job monitoring is quite similar for both AEM and XOSAGA, although XOSAGA introduces some additional overhead. Further tests indicated that the failure rate of jobs is increasing with increase in number of reserved nodes. Hence, one can deduce that increasing the size of grid is inducing overheads on parallel job execution.

The evaluation of Resource Selection Service (RSS) demonstrated that the self-adaptation algorithms can effectively adjust internal RSS parameters to maintain optimal performance across a wide range of fluctuations in node properties and query workloads. Reconfigurations had a limited impact on query delivery. The overall protocol was very inexpensive in terms of memory and bandwidth requirements.

We performed further tests to validate AEM functional requirements, evaluate its performance and scalability. Respective results are described in [29] in greater detail. AEM in XtreemOS has obtained good results on scalability terms, using only one job manager and a hundred resource nodes. Scalability is also maintained when asking for job status information. Performance in this part is better than Globus. This shows the benefits of having interaction with the kernel and our architecture.

With respect to XtreemOS data management, we performed tests to evaluate the performance of XtreemFS and compared it to the performance of NFS and CEPH. Tests were also performed to assess functional requirements and perfor-

mance under stress of Object Sharing Service provided by XtreemOS. We found that network latency is crucial for transaction processing and transaction throughput drops if latency is too high. Although, the maximum theoretical throughput could be reached using local commits. The incrementation of the shared variable offers contrary results of both token passing algorithms. Due to many transaction aborts, incrementing the shared variable scales poorly. We also found that transactional conflicts depend on the granularity of object access detection. In the POSIX compliance tests, XtreemFS passed 90.91% out of the adequate tests. Our experiments with large-scale XtreemFS deployment show the potential of XtreemFS to support transactional load. However more work should be done to improve synchronous write latency, stabilize the replication feature in XtreemFS and further prove its ability to support business application transactional load. Our comparative performance tests revealed that for read operations CEPH outperforms XtreemFS for up to 4 IO streams, while for higher number of IO streams the opposite is true; for write operations XtreemFS outperforms CEPH, excluding the case with only one OSD server. The experiments with XtreemFS's replication show that XtreemFS provides consuming business applications with fault tolerance. However the replication feature in XtreemFS should be further stabilized in order to bring performance benefits for a business application. 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 using normal hard drives, it enables to reach the throughput of the baseline filer technology that uses solid-state drives (SSD). Based on the tests we conclude that POSIX Compliance leaves the room for improvement. Asynchronous write latency under the transactional load is still relatively high as compared to the baseline filer technology. And most probably this shortcoming may be overcome only by means of using solid-state drives (SSD) at OSD nodes. Our experimental results under transactional load look very promising and suggest that XtreemFS may support transactional load. However more experiments need to be performed to support this conclusion. We can conclude that XtreemFS and Object Sharing Service in XtreemOS Release 2 are adequate, but performance leaving some room for improvement. Further testing is required for comparative performance analysis of XtreemFS with additional advanced file systems such as Lustre. Another set of tests is required to assess the performance benefits that a business application can get from the replication feature of XtreemFS. An additional set of experiments should include a usage of read/write replication to overcome failures such as network partitioning and to improve performance of applications in WAN conditions.

Also the main security features of XtreemOS have been evaluated. These are consistent with the features expected from any Grid platform but enhance the state of the art by integrating them with the operating system. For example the integration of OS-level isolation features is an improvement to existing practice. The security architecture is comparable to the Security Architecture for Open Grid Services proposed by Nagaratnam et al.[28], but does not include features such as

intrusion detection, anti-virus management and secure conversations, as these are not fundamental to the Grid security problem. It is possible to implement Grid security mechanisms that are integrated with OS-level mechanisms without introducing significant overheads. Future work is to apply these security principles in the context of Cloud computing, where there is more emphasis on virtualization and collocation of services. In this case the isolation features become even more important for management and not just security.

The tests executed have been passed without major issues. The functionalities developed in XtreemOS-MD and the applications have been verified. The tests show that XtreemOS-MD performance is comparable to the XtreemOS PC flavour, in terms of protocols behaviour. The general conclusion is that XtreemOS-MD brings Grid functionality to Linux-based mobile devices. The software developed may be exploited in any Linux-based terminal running on top of a XtreemOS core Grid.

A further category of tests was devoted to evaluating the benefits of using XtreemOS technology as a basis for Cloud Computing. Whereas the last deliverable put emphasis on Grid Computing aspects (comparing XtreemOS with competitive Grid Middleware solutions), deliverable D4.2.7 tries to assess the capability of XtreemOS for enabling (or supporting) the management of cloud environments and the development of cloud services. Considering the management of cloud environments the evaluation has shown that automated systems management can support data centres in becoming more efficient. Labor costs reduction leads to significant reduction in overall costs of ownership. However, without resilience and recoverability implemented in the management system, or built as a feature of the operating environment, the benefits of automation are degraded. Regarding the capability of XtreemOS to support the development of cloud services, the tests have shown that its not only possible to create a web-application running on XtreemOS, it was even faster to use the provided features than developing them on our own. We are not aware of a any distributed file system (DFS) which could replace XtreemFS in the given scenario, other DFS having either local boundaries like Lustre, which is for LAN use only, or doesn't scale so well like CIFS of WEBDAV. Because of the lack of alternatives one would have to create his one storage pool solution or store the images into a normal data-base. However the first solution would be time consuming and the second one would scale well, since standard OpenSource DBMS can hardly be distributed.Second using the XOSAGA-API provides a comfortable way to generate jobs, which are executed in other processes even on different machines and again thanks to XtreemFS no additional work is needed to get the pictures to the machine which runs the ImageMagick commands. The possibility to increase the storage size dynamically was a huge advantage. In other solutions, which could use CFIS or some other remote file systems, however, you always have to register it to the server. Also the placement of the data does not require any influence from the server which would not be the case for example when you have different mount points. Other distributed file systems have a locality constraint, in the case tested we could host OSDs even on different clouds and static machines

around the world. It was also shown in the test that it basically does not matter if you run XtreemOS on stationary machines or in a cloud environment. It works in both cases and does not give you problems when you prepare it as cloud image. XOSAGA-SSH gave some flexibility to choose the host we wanted to run our compute jobs on. Possible improvements may include load-balancing for lightweight jobs: Compute-nodes could register themselves to the server running Zmile, like the XtreemFS OSDs are doing.

Evaluation reports and bugs have been continuously reported to developers and to project management thereby guiding the project and contributing to software evolution. The project approached its end and the we can conclude that the software product became increasingly mature and complete.

Chapter 7

Acknowledgments

Many thanks go to 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.

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] David P. Anderson and Kevin Reed. Celebrating diversity in volunteer computing. In *Proc. HICSS*, pages 1–8, January 2009.

[3] SPECweb2005 benchmark. Specweb2005 benchmark. Website, 2010. http://www.spec.org/web2005/.

[4] SPECweb2009 benchmark. Specweb2009 benchmark. Website, 2010. http://www.spec.org/web2009/.

[5] Raphael Bolze, Franck Cappello, Eddy Caron, Michel Dayde, Frederic Desprez, Emmanuel Jeannot, Yvon Jegou, Stephane Lanteri, Julien Leduc, Noredine Melab, Guillaume Mornet, Raymond Namyst, Pascale Primet, Benjamin Quetier, Olivier Richard, El-Ghazali Talbi, and Irea Touche. Grid'5000: A large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494, November 2006.

[6] Barcelona Supercomputing Center. Marenostrum supercomputer. http://www.bsc.es.

[7] SPECweb2005 E commerce Workload Design Document. Specweb2005 e-commerce workload design document. Website, 2009. http://www.spec.org/web2005/docs/EcommerceDesign.html.

[8] The Kerrighed Community. Kerrighed. Website, 2010. http://www.kerrighed.org/.

[9] XtreemOS consortium. Design and implementation of node-level vo support. XtreemOS deliverable D2.1.2, 2007.

[10] XtreemOS consortium. Evaluation report. XtreemOS deliverable D4.2.6, 2009.

[11] Paolo Costa, Jeff Napper, Guillaume Pierre, and Maarten van Steen. Autonomous resource selection for decentralized utility computing. In *Proceedings of the 29th International Conference on Distributed Computing Systems (ICDCS)*, June 2009.

[12] SPECweb2005 Benchmark Design Document. Specweb2005 benchmark design document. Website, 2009. http://www.spec.org/web2005/docs/designdocument.html.

[13] Inc. Eucalyptus Systems. Eucalyptus, the open source cloud platform. Website, 2010. http://open.eucalyptus.com/.

[14] Open GATE. Simulations of preclinical and clinical scans in emission tomography. Website. http://www.opengatecollaboration.org/forewords.html.

[15] Hewlett-Packard. The netperf homepage. Website, 2010. http://www.netperf.org/.

[16] IEEE. IEEE standard for software test documentation, ieee 829-1998. IEEE Computer Society, 1998.

[17] Alexandru Iosup, Hui Li, Mathieu Jan, Shanny Anoep, Catalin Dumitrescu, Lex Wolters, and Dick H. J. Epema. The grid workloads archive. *Future Generation Computer Systems*, 24(7):672–686, 2008.

[18] S. Jan. Gate: a simulation toolkit for pet and spect. *Phys. Med. Biol.*, 49(3):4543–4561, July 2004.

[19] S. Kortas. Rï£¡olution haute prï£¡ision des ï£¡uations de navier-stokes sur machines parallï£¡es ï£¡mï£¡oire distribuï£¡. Phd thesis, Universit'e de Provence, Centre de Mathematiques et d'informatique.I, France, 1997. http://samuel.kortas.free.fr/DOCS/THESE/these_Samuel_Kortas.pdf.

[20] 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.

[21] Libvirt. Libvirt: Virtualization API. Website, 2010. http://www.libvirt.org.

[22] Barry McLarnon, Philip Robinson, Paul Sage, and Peter Milligan. Classification and impact analysis of faults in automated system management. In *Proc. of the Third International Conference on Dependability (DEPEND)*, pages 182–187, July 2010.

[23] John Mehnert-Spahn, Thomas Ropars, Michael Schoettner, and Christine Morin. The architecture of the xtreemos grid checkpointing service. In *Euro-Par*, pages 429–441, 2009.

[24] John Mehnert-Spahn and Michael Schoettner, editors. *Checkpointing and Migration of Communication Channels in Heterogeneous Environments, 11th International Conference on Algorithms and Architectures for Parallel Processing, ICA3PP, Busan, South Korea, Mai 21-23, 2010*, Lecture Notes in Computer Science. Springer, 2010.

[25] N. Miles. *Emission Tomography. The Fundamentals of PET and SPECT*. Elsevier Academic Press, USA., 2004.

[26] E. Milošev, 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.

[27] Marc-Florian Müller, Kim-Thomas Rehmann, and Michael Schöttner. Efficient commit ordering of speculative transactions. In *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 21–27, June 2010.

[28] Nataraj Nagaratnam, John Dayka, Anthony Nadalin, Frank Siebenlist, Von Welch, Ian Foster, and Steve Tuecke. Security architecture for open grid services. Global Grid Forum Draft, 2002. http://www.ggf.org/ogsa-sec-wg.

[29] R. Nou, J. Giralt, J. Corbalán, E. Tejedor, J. O. Fitó, J. M. Pérez, and T. Cortés. Xtreemos application execution management: A scalable approach. In *The 11th ACM/IEEE International Conference on Grid Computing (Grid 2010)*, 2010.

[30] University of Pisa. Information technology center of the university. Website. http://www.itc.unipi.it.

[31] OpenNebula. The open source toolkit for cloud computing. website, September 2010. http://www.opennebula.org/.

[32] OpenVZ. Openvz. Website, 2009. http://wiki.openvz.org/.

[33] Raja Parasuraman, Thomas B. Sheridan, and Christopher D. Wickens. A model for types and levels of human interaction with automation. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 30(3):286–297, 2000.

[34] Pidgin. Pidgin, a universal chat client. website, September 2010. http://www.pidgin.im/.

[35] Kim-Thomas Rehmann, Marc-Florian Müller, and Michael Schöttner. Adaptive conflict unit size for distributed optimistic synchronization. In *Euro-Par (1)*, pages 547–559, 2010.

[36] R. Reuillon, D. R. C Hill, C. Gouinaud, Z. El Bitar, V. Breton, and I. Buvat. Monte carlo simulation with the gate software using grid computing. In *NOTERE '08: Proc. of the 8th Int. Conf. on New Technologies in Distributed Systems*, pages 1–4, New York, NY, USA, 2008. ACM.

[37] Jan Sacha, Jeff Napper, Corina Stratan, and Guillaume Pierre. Adam2: Reliable distribution estimation in decentralised environments. In *Proceedings of the 30th IEEE International Conference on Distributed Computing Systems (ICDCS)*, June 2010.

[38] Dwight Schauer. Lxc howto. Website, 2010. http://lxc.teegra.net/.

[39] Corina Stratan, Jan Sacha, Jeff Napper, and Guillaume Pierre. Coordinated self-adaptation in large-scale peer-to-peer overlays. Technical Report IR-CS-60, Vrije Universiteit, Amsterdam, The Netherlands, September 2010. http://www.globule.org/publi/CSALSPTPO_ircs60.html.

[40] Inc. The Linux Kernel Organization. The linux kernel archives. Website, 2010. http://www.kernel.org/.

[41] Apache Tomcat. Apache tomcat. Website, 2010. http://tomcat.apache.org.

[42] Fatih Turkmen and Bruno Crispo. Performance Evaluation of XACML PDP Implementations. In *SWS 2008: ACM Workshop on Secure Web Services*, pages 37–44. ACM, 2008.

[43] XtreemOS. Advanced guide: Installation and administration. Technical report, August 2010. https://gforge.inria.fr/scm/viewvc.php/*checkout*/doc/trunk/AdminGuide.pdf?root=xtreemos.

[44] XtreemOS. Third prototype implementation of security and vo management services. Deliverable D3.5.16, May 2010.

[45] XtreemOS. User's guide. Technical report, August 2010. https://gforge.inria.fr/scm/viewvc.php/*checkout*/doc/trunk/UserGuide.pdf?root=xtreemos.

[46] XtreemOS. Xtreemos grid operating system component interfaces release v3.0. Technical document, July 2010. https://gforge.inria.fr/scm/viewvc.php/*checkout*/interfaces/Release3.0/Release3-interfaces.pdf?root=xtreemos-deliv.