ELASTICITY THANKS TO KERRIGHED SSI AND XTREEMFS
Elastic Computing and Storage Infrastructure using Kerrighed SSI and XtreemFS

Alexandre Lissy[1], Stéphane Laurière[2]
[1]*Laboratoire d'Informatique, Université de Tours, 64 avenue Jean Portalis, Tours, France*
[2]*Mandriva, 8 rue de la Michodière, Paris, France*

Julien Hadba
Laboratoire d'Informatique, Université de Tours, 64 avenue Jean Portalis, Tours, France

Keywords: Kerrighed, Cloud, Infrastructure, Elastic, XtreemFS, Storage.

Abstract: One of the major feature of Cloud Computing is its elasticity, thus allowing one to have a moving infrastructure at a lower cost. Achieving this elasticity is the job of the cloud provider, whether it is IaaS, PaaS or SaaS. On the other hand, Single System Image has a hotplug capability. It allows to "transparently" dispatch, from a user perspective, the workload on the whole cluster. In this paper, we study whether and how we could build a Cloud infrastructure, leveraging tools from the SSI field. Our goal is to provide, thanks to Kerrighed for the computing power aggregation and unified system view, some IaaS and to exploit XtreemFS capabilities to provide performant and resilient storage for the associated Cloud infrastructure.

1 INTRODUCTION

When talking about Cloud, John McCarthy's (Garfinkel, 1999) quotation at the MIT's 100^{th} year celebration is often used: "computation may someday be organized as a public utility". This idea, which dates back to 1961, came back after the Dot-Com bubble, pushed by Amazon: like many, the company had datacenters full of servers, running mostly at 10% of their full capacity and occasionally running at their full capacity in the periods of potential peaks as the ones occurring at Christmas for example. Amazon decided to stop wasting this power, and created *Amazon Web Services (AWS)*, in July 2002. This date might be refered to as the Year Zero of Cloud Computing. Since then, all big IT companies have been involved in Cloud.

From a user perspective, the main advantage of AWS is the ease of use that it provides. It became possible to easily access huge computing power at low cost. With Amazon Elastic Cloud Computing (Amazon EC2), the idea went further and Xen virtual machines now are available via the service. Elastic Computing is now becoming a major feature that is tightly linked to Cloud, especially for its Infrastructure part.

Another interesting field is the Single System Image: SSIs are distributed systems running on top of a network of computers, that appear as one big unique computer to the user. Several projects aimed at providing such a system. Among them, we can cite MOSIX (Barak and Shiloh, 1977), openMosix (Bar, 2008), OpenSSI (Walker, 2001) and Kerrighed (Kerlabs, 2006). The first one is not a free software, but is still actively developed. The openMosix project died in 2008, but the LinuxPMI (Longtin, 2008) project is trying to continue the work initiated. OpenSSI is not dead but still based on a rather old kernel (2.6.14 for the most recent kernel used, 2.6.12 in the latest released version). Kerrighed is the only project still active and working with a kernel 2.6.30.

XtreemFS is a part of the European Founded project "XtreemOS". Its goal is to provide a distributed and replicated filesystem for the Internet. It has now reached some useable state as demonstrated in (Lissy, 2009a).

Our goal is to leverage these tools and gather them together.

The rest of this paper is structured as follows: in section 2 we present some related works on SSI and Cloud, we also introduce the involved tools as well as our goal regarding what has already been done by others. Then, in section 3 we present our goal and mo-

Lissy A., Laurière S. and Hadba J..
ELASTICITY THANKS TO KERRIGHED SSI AND XTREEMFS - Elastic Computing and Storage Infrastructure using Kerrighed SSI and XtreemFS.
DOI: 10.5220/0003391504310438
In *Proceedings of the 1st International Conference on Cloud Computing and Services Science* (CLOSER-2011), pages 431-438
ISBN: 978-989-8425-52-2
Copyright © 2011 SCITEPRESS (Science and Technology Publications, Lda.)

tivations. In section 4 we present our solution from a technical and implementation point of view, and in section 5 we propose a critical analysis of the proposed solution as well as a test plan and criterions to validate or not the viability of this implementation. Then we conclude in section 6 whether our solution is viable or not.

2 RELATED WORKS

In this section, we present a couple of papers related to our topic: Single System Image, Cloud, and/or XtreemFS. It can be noticed that there are only few papers dealing with the use of SSI to provide Cloud.

2.1 A View of Cloud Computing

Definition. Based on view of cloud computing (Armbrust et al., 2010), we define a cloud as a system which respect three properties

- impression of infinite capacity: cloud cannot be limited by the system capacity. User must have impression that the capacity of the cloud is unbound by hardware (storage, computing, ...).
- short term usage: usage of cloud computing is free constraint. User must use cloud just for his needed.
- no upfront cost: utilization of cloud avoid the initial cost of his own equipment. This must not be visible for the user.

Important Cloud Features. We note important features which appear as essential in cloud computing.

- **Elasticity.** To respect the three properties above, a cloud must have the capacity to scale quickly. Plus in practice, we note peaks of utilization. For instance, reactivity is an important user requirement. We can explain the failure of services by a too long response time. Generally capacity of cloud computing is design on maximum need reached on peaks. The rest of the time (most of the time), the cloud is under used.
- **Scalability.** If demand evolution of resources is unknown, need of resource can be assumed by another one. That is a typical use case of cloud and that's why capacity of cloud have to be scalable (unbound).
- **Development Framework.** Utilization of cloud implies a new model for application design. The model is dependent on the cloud. A general rule can be picked up: the more developers have freedom, the more they have to consider specific mechanisms of the cloud. Inversely, more mechanisms of cloud is automatic, more the model of development is restrictive. To ensure the portability of application under more than one cloud, framework have to tend toward standard. A smooth solution is to reuse existing standard of programming.
- **Virtualization Solution.** Quality of service and bug detection is difficult to manage by direct utilization of cloud. Bugs are hardly repeatable because it is difficult to recreate environment. Classic solution is to partition the user space. That's why we used virtualization.

2.2 Kerrighed

Kerrighed (Kerlabs, 2006) is a project that was initiated in 1999, at INRIA. It was developed until 2006 at IRISA, in Rennes, with people from INRIA, EDF and the University Rennes 1. In 2006, it was judged that it was time to move from a research prototype to an industrial solution, and thus Kerlabs was founded by members of the research team, to continue the development. Kerrighed allows a set of interconnected (LAN) computers to function as a so-called "Virtual-SMP", aiming at making a Single System Image: single process space, single IO space, single root, etc. Single File system root is provided through a common NFS server for all nodes. One major feature of Kerrighed is the ability to handle processes on several nodes. There are two ways to deal with it: migration or distant fork. The goal of those two is to balance the load on the whole cluster. Migration might still suffer from some hiccups, even if these are improving incrementally each time. The idea is to freeze the process, transfer it to another node, and continue the execution. Distant fork is much simpler: when `fork()`ing, Kerrighed intercepts the system call, and manages to execute it on another node.
These features are controlled, since the Kerrighed 2.4 version, by a userspace-configurable kernel scheduler framework, called SchedConfig. The "legacy" scheduler provided with Kerrighed behaves like MOSIX' one. As part of (Lissy, 2009b), another scheduler was implemented, based on the work by (Pérotin, 2008), called "RBT". This scheduler is able to control migration and/or distant fork, i.e. to decide on which node the execution of a process should be done, either by migrating the process, or when `fork()` occurs.
As part of the 3.0 release, major work was done to support Hotplug node. Currently, the addition of nodes while running the cluster is fully function-

nal. The removal of nodes, however, remains complicated. The developers have patches that works relatively well to allow the administrator to get a node out of the cluster. However these patches are not quite ready yet. A crashing node is still a big issue, but it should be addressed eventually (patches allowing safe removal of a node from the cluster are already available and working). This hotplug feature is very interesting, since it allows very easy elasticity: basically, you take a node, you rack it, plug the network, power up and your cluster has more power.

2.3 XtreemFS

XtreemFS (XtreemFS Team, 2006) is a distributed and replicated Internet filesystem, that was developed as part of the XtreemOS (XtreemOS Consortium, 2006) project. The objective of the project is to provide a solution to have a distributed, parallel and replicated filesystem for sharing data inside the XtreemOS project, but not just limited to it. It consists of several components: a directory service (DIR), a metadata and replication service (MRC) and a storage service (OSD). The directory service allows all the other services to register themselves and be able to know which is available. MRC is responsible for the metadata of the stored data and replication. The OSD service is the one that runs on each node and effectively stores data. The administrator will be able to create several volumes on the MRC, which will determine the features of the volume: access control policy (POSIX by default), a stripping policy (only RAID0 for the moment) a stripe size (128kB by default) and a stripe width (1 by default). This stripping feature allows chunks of data to be stored onto different OSD. When the client will mount the filesystem, it will enable him to perform parallel reads from all the OSD that have the data, thus improving throughoutput.
Replication works but might still need some work in order to be better and easier to deal with. The filesystem provides an extension mechanism, so specific features might be added easily. This stripping enables elasticity too, even though data will not, currently, be replicated automatically onto the new nodes, at least the volume size will increase when adding new OSD to the system. Thanks to the DIR service, this is kind of automagic.

2.4 Apache Hadoop

Apache Hadoop, which some consider to be a PaaS component of Cloud, is a framework designed to treat very large datasets. It is a free implementation of Google's MapReduce, implemented thanks to their publications. It is distributed and fault-tolerant, but requires using its specific API to be used. It is not POSIX-compliant, but there are projects aiming at providing a FUSE interface to Hadoop (Foundation, 2009).

2.5 Ceph Project

This project, started as a PhD thesis, aims at providing a distributed and scalable free file system for Linux. Scaling is achieved simply, in the same way as XtreemFS is, by adding new Object Storage Device (OSD). The filesystem replicates data accross several OSDs inside the cluster, allowing both resilient behavior when losing some nodes, and parallel reading while retrieving data. One feature that is very interesting is the ability to rebalance the load when extending the cluster with new OSDs. Both an advantage and an issue is the fact that Ceph is now mainstream Linux kernel. It means that it works well enough to be included in the official vanilla sources since 2.6.34, which is very good. It also means that if we want to have it in Kerrighed, we will have some work to do to backport it to 2.6.30 used by the SSI. As the kernel evolves very rapidly, this might be an issue. Moreover, dealing with FUSE might be much more valuable: there are other interesting filesystems based on this tool that might benefit from our work.

2.6 XtreemOS and Cloud Computing

In (Morin et al., 2009), the authors analyse the "impact" of the recent Cloud Computing idea and how it interacts with XtreemOS. Two approaches are used: leveraging clouds for XtreemOS usage, and using XtreemOS technologies to manage cloud computing infrastructures.

In the first case, the idea is to install XtreemOS in several Clouds to be able to federate themselves. XtreemFS is an interesting piece of software for this case, due to its POSIX interface for easy integration with legacy application, contrary to "classical" Cloud storage. Also, the use of an SSI (LinuxSSI precisely in XtreemOS) allows to provide much larger virtual machines than Amazon EC2 does. The processor transparency provided by SSI is an advantage here.

In the second case, XtreemOS is used as an infrastructure for cloud providers. The major concept of Virtual Organization, at the heart of XtreemOS, can be leveraged here to define and apply policies and access control for the resources used. XtreemOS will be running Virtual Machines, and these will be managed as an application. Any operating system might run inside the VM. Migration and hotplug allow for ser-

ver consolidation depending on the workload.

2.7 SSI on the Cloud

In (Modzelewski et al., 2009), the authors describe how they intend to ease the use of cloud and massively parallel computers (1000+ cores, called "many-cores"), by designing a cloud-aware operating system: *fos*. The main idea is to have a Single System Image running on top of an *Infrastructure as a Service* cloud. To design their OS, they introduce four challenges that *fos* must address: scalability, elasticity of demand, faults and programming large systems.

Scalability. Algorithms inside the OS, such as locking for example, must scale beyond one hundred or more cores. They reuse their work analyzing the limitations towards this goal in current OS design.

Elasticity of Demand. It is defined as this: workload and resources to treat demand will change dynamically.

Faults. Obviously, large systems are more prone to failures, and there are several references about these issues, such as (Shang et al., 2007; Huysmans et al., 2006; Hacker et al., 2009). So the Operating System must be designed with fault tolerance in mind.

Programming Large Systems. Currently, tools to efficiently use large systems easily are not available: no uniform programming model for communication (threads for local resources, sockets for remote resources), cloud complexity embedded inside the application whereas for multiprocessor systems it is embedded inside the OS.

The authors propose a solution, "a factored OS". The main design principles are:

- Space multiplexing instead of time multiplexing.
- Factorization into specific services.
- Faults detection and handling.

Even though it does not correspond exactly to our research, those challenges are interesting in the way that they demonstrate the complexity of large systems.

2.8 SuperComputers and Cloud Computing

In "Providing a Cloud Network infrastructure on a Supercomputer" (Appavoo et al., 2010), the authors present a similar idea: they state that, due to cost and scale condiserations, cloud hardware will evolve to something similar to supercomputer hardware. More precisely, they are interested in showing that it is possible to use supercomputers as cloud infrastructure. Scientific computing, and thus supercomputers aim at delivering the highest performances achievable. Nowadays, however, they tend to use standard general-purpose software environments rather than highly-specialized and dedicaded ones, to allow easy portability. Some might then consider using Cloud for High Performance Computing (HPC). For them, a central feature of Cloud is "elastic resource provisioning of standardized resources", to easily scale according to users needs. Elasticity can also be used in high-performance systems, and the authors introduces two hypotheses that they intend to verify by adding dynamic network virtualization to a supercomputing system:

1. Elasticity can be achieved along with the predictability associated with high performance computing;
2. General-purpose elastic computing can be supported with direct access to specialized networks.

This elasticity is not part of the supercomputer architecture. The authors' idea is thus to adapt HPC, by emulating Ethernet over their BlueGene/P's specific networking, RDMA, to allow the use of classical Cloud tools. They demonstrate its performances, with a patched version of memcached and benchmark.

They conclude that by providing commodity software layer, they can run a cloud computing application on top of their BlueGene/P and benefit from its computational power. One of their future works will be to backfill the supercomputer with cloud in idle periods.

2.9 Using XtreemOS for NBX Grid

In the technical report (Mason et al., 2010), the authors study the use of XtreemOS components inside NBX (NuGO Black Box, a project that aims at providing a "lab-scale" bioinformatics servers. As part of an upgrade of the solution, the authors study the possibility to leverage XtreemOS for their needs: mainly, Kerrighed and XtreemFS. They justify the Kerrighed choice by the status of all others FLOSS[1] SSI: openMosix is dead since march 2008, OpenSSI is rather old (latest release was in 2010, 18th Feb., and with kernel 2.6.12). Only Kerrighed is still developed actively. Even though they had a couple of hiccups using this piece of software, they finally validated its use. Then, they studied the filesystem

[1]Free and Libre Open Source Software.

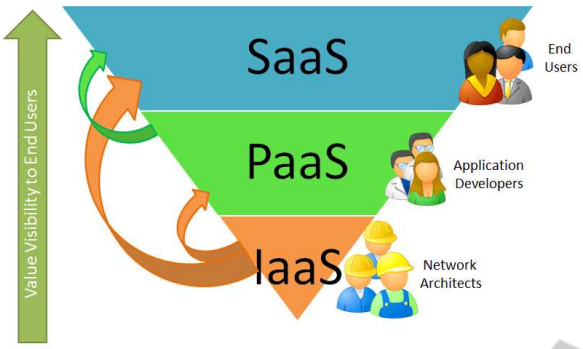

Figure 1: The Cloud Computing Stack (Clubic.com, 2010).

part: replacing the use of "sshfs"[2] to share data, with XtreemFS, which is *considerably more robust than "sshfs"*. However, they managed to achieve good results in local network (stating that XtreemFS *uses too many ports for the firewall configuration* which made them unable to test at WAN level). Finally, as a conclusion, they validate the use of both XtreemFS and Kerrighed inside their NBX platform.

3 AN ELASTIC AND TRANSPARENT INFRASTRUCTURE

In the previous sections, we have shown how elasticity is one of the biggest advantages provided by Cloud Computing. Before introducing the idea, it is important to remind our readers of the layers (figure 1) involved in Cloud, in a bottom-up approach:

1. Infrastructure as a Service (IaaS): Amazon EC2
2. Platform as a Service (PaaS): Google App Engine
3. Software as a Service (SaaS): ERP5 SaaS

As we have also seen previously, both Kerrighed and XtreemFS are *elastic-aware*, meaning that they can be extended at will and without any downtime. So the idea is pretty simple in fact: following the same kind of logic as in (Morin et al., 2009) and in (Appavoo et al., 2010), we would like to benefit from these capabilities to easily provide the basis for an IaaS, effectively leveraging Kerrighed and XtreemFS. Even though both might not be production ready for this kind of usage, it is worth checking the feasibility of this kind of construction. One usage of this solution can be imagined as part of the CompatibleOne (Team, 2010) project to provide IaaS, either as a final basis for the solution, or more likely as a way to have an elastic infrastructure as a prototype.

[2]A FUSE software to mount SSH folders.

The process that sustains the production of a Linux distribution relies on infrastructure services that are both CPU and storage intensive: a build service, a code analysis service, and a system integration testing service. For example, in Mandriva's case, it consists of approximately 10,000 source packages and 20,000 binary ones, which evolve at a rapid pace: during activity peaks, the amount of new packages produced every day can amount to more than 1,000, as a consequence of the availability of new component versions, bug fixes or security updates, and of the complex dependency graph among packages. The volumes at stake and the need to meet highly demanding industry requirements in terms of speed, test coverage and traceability call for an innovative scalable infrastructure. The elasticity and the robustness offered by Kerrighed and XtreemFS are appealing for the creation of such a Linux engineering platform. The use of both as an infrastructure for the Mandriva distribution build and analysis system will be experimented and measured in 2011-2012. The key success criteria will hinge on the following aspects: efficient elasticity, performance, robustness, and simplicity of the set up, administration and replication processes.

4 PROPOSED SOLUTION

Now that a description of the context as well as the main idea have been set, we will focus on delivering a description of the technical aspects of the solution. But first, we will summarize the qualities of Kerrighed and XtreemFS for this topic.

4.1 Components of the Solution

Kerrighed. Kerrighed is a Single System Image for a cluster of computers. A kerrighed cluster can be viewed as a big computer which is an agregation of all nodes. Following the same idea than for the "fOS" project, the idea is to benefit from the kernel-side single view inherent to the SSI aspect of the project.

XtreemFS. Other distributed filesystems exist, such as Ceph, but XtreemFS is more interesting mainly because it is less intrusive than Ceph. The former is a userspace filesystem, implemented with FUSE, while the latter is in-kernel, which would require patching Kerrighed to add it. XtreemFS also provides elasticity even though it does not support rebalancing the load when adding new storage devices.

Association of Kerrighed and XtreemFS. So now the idea is to associate both Kerrighed and XtreemFS

in one node, so that we can plug new computers and extend both storage and computing power.

4.2 Implemented Solution

To set up the solution, we reused our Kerrighed development cluster. The basic setup involves a NFS server, serving the root filesystem (later referred as "NFSROOT"), and client nodes booting the kerrighed kernel over PXE. Inside the NFSROOT, we simply install XtreemFS server package, to provide OSDs for each node. We installed XtreemFS directory and metadata services on the NFS server. In the end, the setup can be summarized as in figure 2.

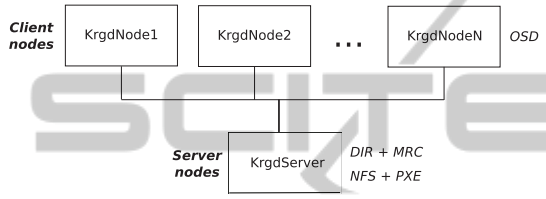

Figure 2: Implemented solution.

Now, in order to mount the filesystem, we can envisage two ways:

- Mount it on the NFS server, then share via NFS.
- Mount it on each Kerrighed node.

It is important to notice that when the client, that is the kerrighed booted nodes, will mount the filesystem, they will request to directory and metadata server. DIR will be in charge of providing the information about the presence of MRC and OSD. When manipulating data, the XtreemFS client will have direct access to the OSD. So the first solution will be bottlenecked by the NFS server.

For our test bed, we deployed two OSDs per Kerrighed node. This requires a couple of operations, as the default installation only allows for one OSD to be launched, but nothing serious (duplicating the configuration file for each node and each instance, setting the appropriate UUID). These had to be done in order to be able to use the two storage devices we added to each node. For the test, they are simply Kingston 4GB USB key, providing a total of 42GB (some reports as 4.01GB, others at 3.8GB). We could have setup some Software RAID0 to allow the two keys to be exposed "as one". But in production this would have meant risking loosing the data of the two disks if loosing one. Two (or more) OSDs provide a better solution.

One interesting feature of Kerrighed is, as we previously stated, distant fork. Thus, we want to exploit distant fork "over" XtreemFS, that is having a distributed storage volume which is used by applications running inside the cluster. At first, it was impossible due to some implementation issues. After notification to the Kerrighed's developer, they came with an idea that we implemented and, as of now, validated. Some work is still needed to polish the code, but the main goal is achieved. Further investigations exposed another issue, related to XtreemFS, on which we are working.

While uncompressing the sources, we also ran into some bug lying in XtreemFS and related to management of symbolic links, which we helped to get fixed.

4.3 Towards PaaS Solution

The goal of the proposed solution is to include kerrighed and XtreemFS as foundation for IaaS as part of a Cloud stack (figure 3). The IaaS layer can be ensured by the management of virtual machines. This management is controled by an intelligent migration policy (section 2.2) in kerrighed and by a control of data in XtreemFS (section 2.3).

5 ANALYSIS OF THE PROPOSED SOLUTION

In this part, we will first characterize the properties that we want to achieve, then we will show how the proposed solution allows to comply with them.

5.1 Properties to Achieve

Elasticity. We define elasticity as the capacity of the system which we want to build to serve for Infrastructure as a Service to be resized, while running, without any downtime. This capacity is needed, for several reasons. First of all, it will allow for easy scaling: need more computational power? Add more servers. Secondly, if we can shrink the system, we can take nodes down when the load is too low. It also allows to remove parts of the system for maintenance for example.

Robustness. We define robustess as the capacity for the system to remain accessible and (even not) fully functionnal when critical events occur. Since it will have many components, it is very likely that some will fail. We don't want the whole system to be down in that case. What would be better would be that nothing got lost: data or process. Even if it bad to be at the cost of computational power.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

Transparency. We define transparency as the capacity of the system to manage both these properties in a transparent way for the applications using it. That is, unlike other cloud applications, as stated in (Modzelewski et al., 2009), we do not want to manage elasticity and robustness in applications.

5.2 Validation of Properties

5.2.1 Elasticity

Kerrighed. As defined previously, elasticity in Kerrighed is the ability to add and/or remove nodes while keeping a running computer accross the nodes. This is precisely the behavior of Kerrighed. Since the latest release, 3.0.0, Kerrighed supports hotplug: we can add new nodes to an already living cluster. Since November, 22nd, Kerrighed has support for hot removal of nodes. It works, even though there are still issues to be solved.

XtreemFS. Elasticity in XtreemFS is natural and linked with the directory service. Each OSD registers to DIR and then can be used for every volume that is present in the system. Removing an OSD, for example by shutting its node down, will reduce the available size of the volume, and if the file chunks it was hosting have not been replicated, corresponding files will remain unavailable until the OSD comes back online.

5.2.2 Robustness

Table 1 summarizes the issues that might arise for both Kerrighed and XtreemFS.

Kerrighed. The question of robustness for Kerrighed can be seen as a two-way question: is it able to handle the loss of one or more several nodes? Can we remove nodes by hand and keep the cluster usable? As of today, only the second part has been addressed in an interesting way. Even if node removal is very recent, publicly available in git tree since November 22nd, it works relatively well. However, we have been able to trigger bugs in some cases. When removing a node, the system tries to migrate all the processes that are running on this node onto other nodes. If a process is not migrateable, then it will be killed. Otherwise, it will remain running. As far as the first question is concerned, the current answer is simply “no”. It is not a limitation, it is a bug not yet addressed. Loosing one node will still crash the whole cluster. Work is being done on failure prediction to avoid this kind of damages.

XtreemFS. Regarding XtreemFS, we can ask ourselves the same questions than for Kerrighed: how does it behave when loosing one node? Can we remove one node safely? While in Kerrighed there is a tool managing node hotplugging, *krgadm*, there is no such thing for XtreemFS, thus both questions are the same. XtreemFS can afford loosing one or several OSDs, as long as there are other providing replicas of the chunks involved. If not, it will issue an error to the requesting application. Once the missing OSDs are back, the data is available again. Also, there is currently a single point of failure for the directory and metadata services. However, this should be addressed in the 1.3 release of XtreemFS (currently 1.2.3). One must also be aware that replication is not yet automatic: the file must be set as “read-only” for replication to be able to start.

Table 1: List of potential problems.

	safety stop	crash
XtreemFS	*full disk space* & recovery	*full disk space* & recovery
Kerrighed	loss of available power	*system crash*

5.2.3 Transparency

Kerrighed. As a SSI, it is totally transparent to the applications that are run inside the cluster. Technically, Kerrighed is implemented as a Linux Container, a lightweight virtualization technique. Once the cluster environment is started, the user can connect to it (through a *sshd* started inside the container) and see a SMP machine that aggregates all the power and memory of all the nodes participating. Transparency is complete. If one wants to benefit from process migration or distant fork, however, running processes must be “configured” (setting capability) to allow those actions. This can be easily circumvented.

XtreemFS. As a FUSE and POSIX-compatible filesystem, XtreemFS is rather transparent. One just need to mount it and then it is possible to use it as any storage device that exists.

5.3 Efficiency

Once we have checked that the requirements are globally met, it seems important to us to check that the performances are not bad. Kerrighed and XtreemFS are not dedicated to Cloud IaaS, so highest performance levels are not expected. However, we want to be able to get an honest average performance. One

possible way to exploit the IaaS that we have created with Kerrighed and XtreemFS would be to run virtual machines inside, and to store virtual hard disks on XtreemFS. Doing so, we could be able to benchmark against Amazon EC2.

For the record, we have "checked" this by doing a massive parallel build of the Linux Kernel (`make -j24`).

6 CONCLUSIONS

We finally set up the IaaS Layer of cloud stack. The described setup is functional. Thanks to kerrighed, we obtain an elastic and transparent computing solution. Thanks to XtreemFS, the proposed solution has an efficient elastic and transparent data storage solution. However we are aware of the current limitations of both, and efforts are made to improve the robustness. Thus we are hoping to reach an ideal elastic solution of cloud.

Our solution has been tested by the execution of a massive parallel computing application. The proposed solution has the advantage of the elasticity which is very interesting in many use cases like the one presented previously. Now that this achievement has been made, we can go further and expect to finally be able to run some benchmarks.

REFERENCES

Appavoo, J., Uhlig, V., Stoess, J., Waterland, A., Rosenburg, B., Wisniewski, R., Silva, D. D., Hensbergen, E. V., and Steinberg, U. (2010). Providing a Cloud Network Infrastructure on a Supercomputer. In *1st Workshop on Scientific Cloud Computing, Chicago, IL, USA, June 2010*, Chicago, Illinois.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. (2010). A view of cloud computing. *Communications of the ACM*, 53(4):50–58.

Bar, M. (2002–2008). The openMosix Project. http://openmosix.sourceforge.net/.

Barak, A. and Shiloh, A. (1977). MOSIX – Cluster and Multi-Cluster Management. http://www.mosix.org.

Clubic.com (2010). Cloud stack. http://img.clubic.com/photo/02883120.jpg.

Foundation, A. (2009). Hadoop hdfs. http://wiki.apache.org/hadoop/MountableHDFS.

Garfinkel, S. L. (1999). *Architects of the Information Society: Thirty-Five Years of the Laboratory for Computer Science at MIT*. The MIT Press.

Hacker, T. J., Romero, F., and Carothers, C. D. (2009). An analysis of clustered failures on large supercomputing systems. *J. Parallel Distrib. Comput.*, 69(7):652–665.

Huysmans, J., Baesens, B., Vanthienen, J., and Gestel, T. V. (2006). Failure prediction with self organizing maps. *Expert Syst. Appl.*, 30(3):479–487.

Kerlabs (2006). Kerrighed – Linux Clusters Made Easy. http://www.kerrighed.org.

Lissy, A. (2009a). Espace de stockage avec XtreemFS. Master's thesis, Département Informatique de l'École Polytechnique de l'Université de Tours.

Lissy, A. (2009b). Politiques hybrides d'ordonnancement dans un cluster Kerrighed. Master's thesis, Département Informatique de l'École Polytechnique de l'Université de Tours.

Longtin, J. (2008). The LinuxPMI Project. http://linuxpmi.org/trac/.

Mason, A., Clivio, L., and Travis, A. (2010). Application of XtreemOS for development of NBX Grid. Technical report.

Modzelewski, K., Miller, J., Belay, A., Beckmann, N., Gruenwald, Charles, I., Wentzlaff, D., Youseff, L., and Agarwal, A. (2009). A Unified Operating System for Clouds and Manycore: fos. Technical report, MIT Dspace [http://dspace.mit.edu/dspace-oai/request] (United States).

Morin, C., Gallard, J., Jégou, Y., and Riteau, P. (2009). Clouds: a New Playground for the XtreemOS Grid Operating System. Research Report RR-6824, INRIA.

Pérotin, M. (2008). *Calcul Haute Performance sur Matériel Générique*. PhD thesis, Laboratoire d'Informatique de l'École Polytechnique de l'Université de Tours.

Shang, Y., Jin, Y., and Wu, B. (2007). Fault-tolerant mechanism of the distributed cluster computers. *Tsinghua Science and Technology*, 12(Supplement 1):186 – 191.

Team, C. (2010). The compatibleone project. http://compatibleone.org.

Walker, B. (2001). OpenSSI Clusters for Linux. http://www.openssi.org.

XtreemFS Team (2006). XtreemFS – a cloud filesystem. http://www.xtreemfs.org.

XtreemOS Consortium (2006). XtreemOS. http://www.xtreemos.eu.