Testing Object-based Storage Device Model for OpenAFS

Michal Švamberg
Luboš Kejzlar

University of West Bohemia in Pilsen
Center for Information Technology
e-mail: [svamberg, kejzlar]@civ.zcu.cz

2 June 2010

Contents

This Technical Report is a product of Project No. 293/2009 funded by the CESNET Development Fund. The Document gives a description of the testing environment, methodology and results.

1 Goals of the Experiment

Tests were intended to confirm the functionality of all Rx-OSD extension components, their stability and suitability for use in a production environment of the University.

2 Introduction to the Rx-OSD

Rx-OSD is the technical successor of MR-AFS, which was developed at the Pittsburgh Supercomputing Center. It is now developed by Hartmut Reuter from Rechenzentrum Garching (RZG) of the Max Planck Society and the IPP. Basically Rx-OSD is adding a new infrastructure how files are stored on the server-side of OpenAFS. Beside on the classical fileserver, files can now be stored on so-called OSD-servers. This feature can be used to have RW-copies of files, place files on tiered disk-system or to automatically migrate files from disk to tape. The files can be split internally and spread over many OSD-Servers, thus we talk about objects rather than files. Within the OSDs, an object may have more than one copy.[3]

3 Testing Environment

The testing environment included servers as well as clients. The purchase of server-side equipment was funded by CESNET Development Fund Project No. 293/2009. The Client side consisted of one high-performance workstation and approximately 30 lab PCs with varying parameters.

3.1 Infrastructure Design

Given the limited funds available, the testing infrastructure had to rely on virtualization technologies. It was designed to reduce the occurrence of bottlenecks and prevent individual machines from influencing each other.

Performance of the disk subsystem is definitely the most significant factor. That is why low-capacity/high-performance harddrives were purchased for the needs of the experiment and a dedicated physical harddrive was assigned to each DomU.[1] Each disk was assigned to its own Volume Group (VG) in an LVM controlled by Dom0. Each VG was then partitioned into a root filesystem (20 GB), a swap partition (2 GB) and a partition reserved for an AFS fileserver, so called /vicepa (50 GB). Remaining capacity was left as spare for

[1] an unprivileged virtual machine—a counterpart of Dom0, which is allowed to control hardware access for other virtual machines.

possible additional tests. 45 GB of disk space were also reserved on drives sda and sdb for an MD-RAID mirror dedicated to the Dom0 system. Partitioning does not cause any performance problems since Dom0 is only used for virtual machine management and does not participate in the test alone.

4 GB of RAM were allocated for the Dom0 system to minimize disk use through maximum reliance on cache. An exactly opposite approach was applied to DomU systems which were only allocated 1 GB of RAM to prevent internal caching mechanisms from influencing test results.

The E5505 processor used in testing comprises four physical cores (no hyperthreading), and the system was configured to assign each core to a specific DomU. Dom0 shared all four cores to reduce load imposed on any single core and balance the load evenly.

There were only two 1-Gb network adapters available to all testing servers. DomU were assigned to individual adapters in twos through bridges. Dom0 was connected directly to a bridge on eth0. It was essential to make sure that the throughput was not lower than the throughput of two DomU working simultaneously.

3.2 Server Infrastructure

3.2.1 Technical Equipment

Two standard physical machines were purchased (see tab. 1), funded by the project grant. Relying on Xen technology,[2] each of them was used to set up four separate DomU servers providing all essential infrastructure services.

Server assignment and names:

chryso1, chryso2 are physical servers. They also provide Dom0 services to manage DomU virtual machines.

chryso1-1, chryso1-2, chryso1-3, chryso1-4 are virtual DomU machines hosted by chryso1.

chryso2-1, chryso2-2, chryso2-3, chryso2-4 are virtual DomU machines hosted by chryso2.

The overall design of the virtualized infrastructure is shown in Fig. 1 and network topology is documented by Fig. 2.

3.2.2 Operating System

The server-side architecture is x86-64 (amd64), running a standard installation of Debian GNU/Linux (Lenny) with a custom-built kernel version 2.6.31.8, compiled with support for project Xen's Dom0 version 3.4 and OpenAFS server with Rx-OSD support, version 1.4.12.

[2] `http://www.xen.org`

Physical machine parameters

RAM	8 GB, 1066 MHz
CPU	1× Intel Xeon E5504 (2.0 GHz, 4 M Cache, 4.86 GT/s QPI)
HDD	4× 300 GB SAS 15 k 3.5”
NET	2× 1 Gb/s Broadcom BCM5716

Virtual machine parameters—Dom0

RAM	4 GB
CPU	4 cores, each shared with a single DomU
HDD	2× dedicated partition on two harddrives mirrored by software RAID
NET	1× 1-Gb/s adapter shared with two DomUs

Virtual machine parameters—DomU

RAM	1 GB
CPU	1× dedicated core
HDD	1× dedicated partition, HDD not shared with any other DomU, LVM by Dom0
NET	1× 1 Gb/s shared with one other DomU

Table 1: Server-side hardware configuration

3.3 Client Side

Configurations of individual clients are shown in Table 2.

3.3.1 Operating System

The clients were running Debian GNU/Linux (x86 or amd64) with kernel version 2.6.30 and OpenAFS clients version 1.4.11 with Rx-OSD extensions.

3.4 OpenAFS Configuration

All tests took place in a dedicated AFS cell civ.zcu.cz, separated from the production infrastructure.

All servers chryso1-x and chryso2-x were running rxosd. Server chryso1-1 also had volserver, osddbserver, vlserver, ptserver and fileserver installed. The same machine was also providing kerberos authentication services.

A set of volumes implementing various policies was created for the purpose of testing:

stripe0 no policy. Used to test direct access to the file server with no Rx-OSD extensions.

stripe1,2,4,8 one, two, four or eight stripes for data managed by rxosd servers.

Stripe[3] size was always 12 ($2^{12} = 4096$ B). The number of stripes used is always in powers

[3]It's the number of stripes your file consists of. With 2 stripes and stripesize 12 the 1st 4k go into stripe 0 the next 4k into stripe 1 and the 3rd 4k again into stripe 0 and so on. Each stripe is an object on a different OSD.

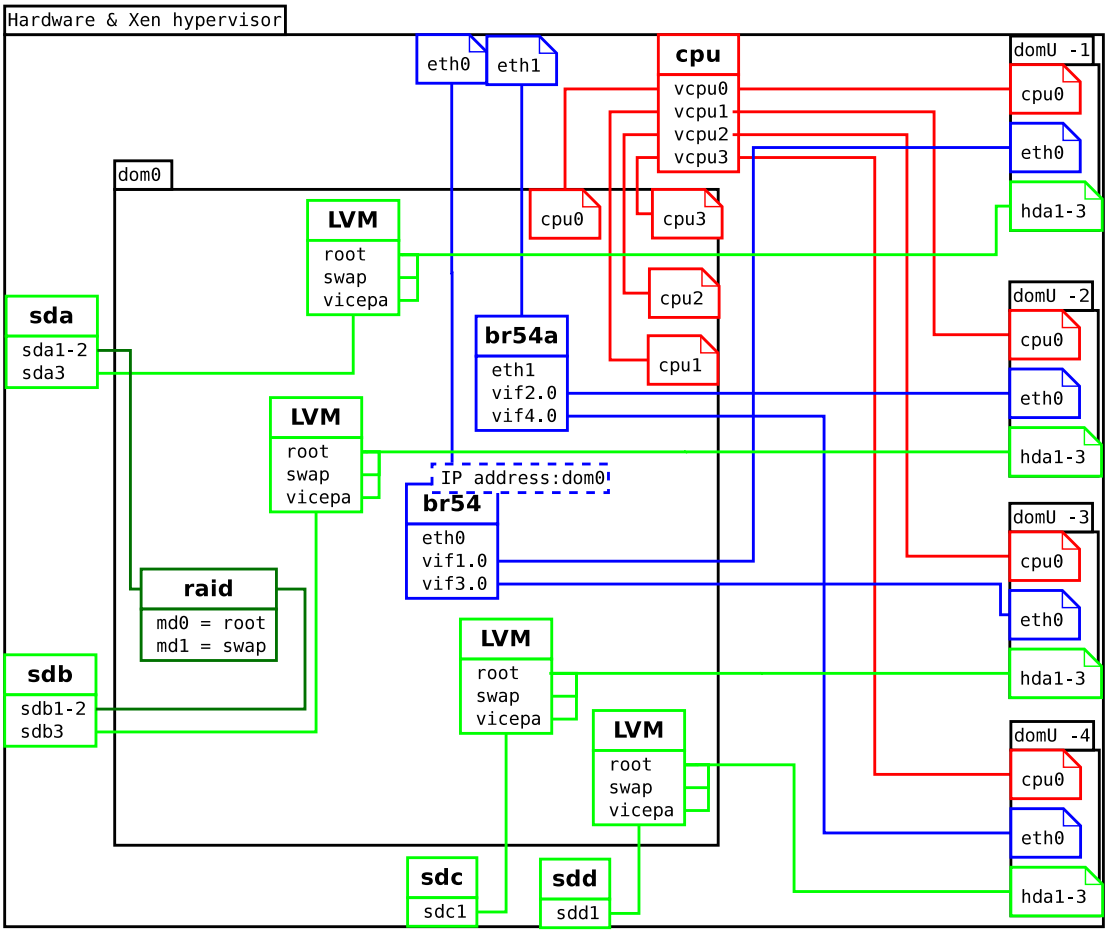

Figure 1: Configuration of the Virtual Infrastructure

of two and is limited by implementation to a maximum of eight. With options stripe1-8 the fileserver is only used to store file metadata, while the actual file management is provided by rxosd servers and the load is balanced on the client side. Fileservers can provide load balancing for clients not implementing Rx-OSD extensions.

Obviously, when using stripe0, data will be manipulated in a "classic" manner, imposing load solely on node chryso1-1 running the fileserver. Contrary to that, for RxOSD-enabled options stripe1-8, load will be spread among all available rxosd servers.

4 Testing Procedure

The testing procedure was rather time-consuming[4] and, as the laboratories were being used by students in daytime, experiments could only run overnight.

[4]Certain tests took several hours.

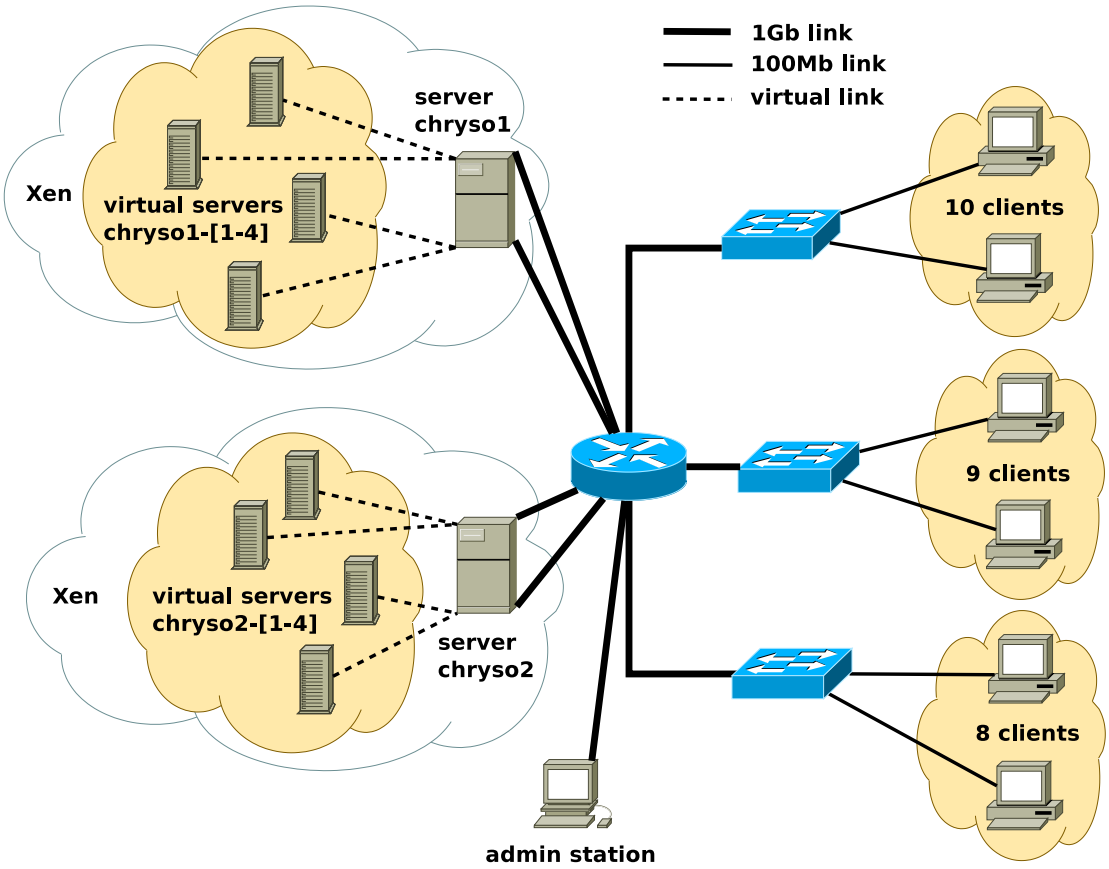

Figure 2: Configuration of the Testing Network

The iozone application was used to generate I/O data streams with the required properties. It was invoked on individual clients through parallel-ssh:

```
time parallel-ssh -h nodes.txt -p 100 -t 86400 -o result 'iozone_command'
```

The `iozone_command` argument took the following form:

```
iozone -s 1G -r 256k -c -t 1 -F ‘tempfile -d stripe2‘ -i 0 -i 1 -i 2
```

with individual parameters specifying the following:

-s 1GB testing file size 1 GB,

-r 256k record size is 256 kB. This value varied for different tests,

-c include time required by the close() function in the overall time,

-t 1 number of threads operating simultaneously.

Laboratory UI505b – 9 machines

RAM	1 GB
CPU	1× Intel Pentium 4, 3.40 GHz, single core
NET	1*times* 100 Mb/s, connected to a switch with 1-Gb/s uplink
ARCH	32-bit

Laboratory UI505 – 8 machines

RAM	2 GB
CPU	1× AMD Athlon 64 X2 Dual Core 4200+, 1 GHz, dual core
NET	1× 100 Mb/s, connected to a switch with 1-Gb/s uplink
ARCH	64-bit

Laboratory UI312 – 10 machines

RAM	768 MB
CPU	1× Intel Pentium 4 CPU, 2.60 GHz, single core
NET	1× 100 Mb/s, connected to a switch with 1-Gb/s uplink
ARCH	32-bit

Table 2: Client side hardware configuration

-F filename file name generated by calling tempfile in directory stripe2,

-i 0 run a write/rewrite test,

-i 1 then run a read/reread test,

-i 2 finally run random-read/write test.

All tests were run repeatedly and then evaluated:

Overall time returned by the time command. This value indicates time required to run the test, including operations on all nodes.

iozone *statistics* for all nodes participating in the test. This shows the dispersion in delays of various operations between nodes.

Dom0 *load at* chryso1 *and* chryso2 measured by dstat. Disk and network adapter load measured second by second.

5 Test Results

All results shown bellow are given relative to the overall time required to finish the test, i.e. all tasks on all nodes.

5.1 Block Size Impact

This test used various block sizes when calling iozone (argument -r, record size). Smaller blocks result in more operations and increased time consumption. The final graph shown in Fig. 3 demonstrates an obvious difference between Rx-OSD and classic OpenAFS.

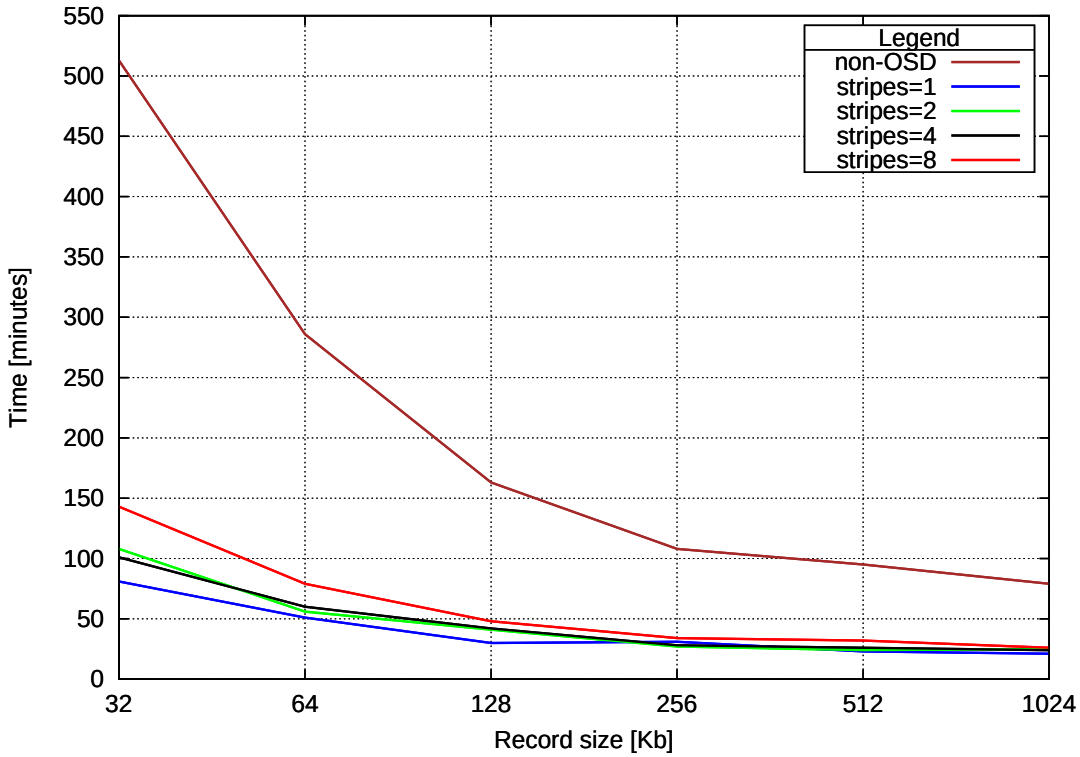

Figure 3: Dependence on block size for various stripe numbers

5.2 Impact of rxosd Server Failure

This experiment was designed to test the behavior of the system in case of an unexpected rxosd server failure.

Graphs shown in Figures 4 and 5 use the same underlying data. Following discussions with Rx-OSD developers, two extreme values were replaced with approximations in post-processing (original values are shown as data points). They were caused by measurement errors and by insufficient delays between failures of individual rxosd servers, which caused calls to unavailable servers to time out.

Graphs show that in no circumstances (with the exception of the two cases explained above) were the results worse than classic OpenAFS. Rx-OSD usually gives better or, in the worst case, equal throughput.

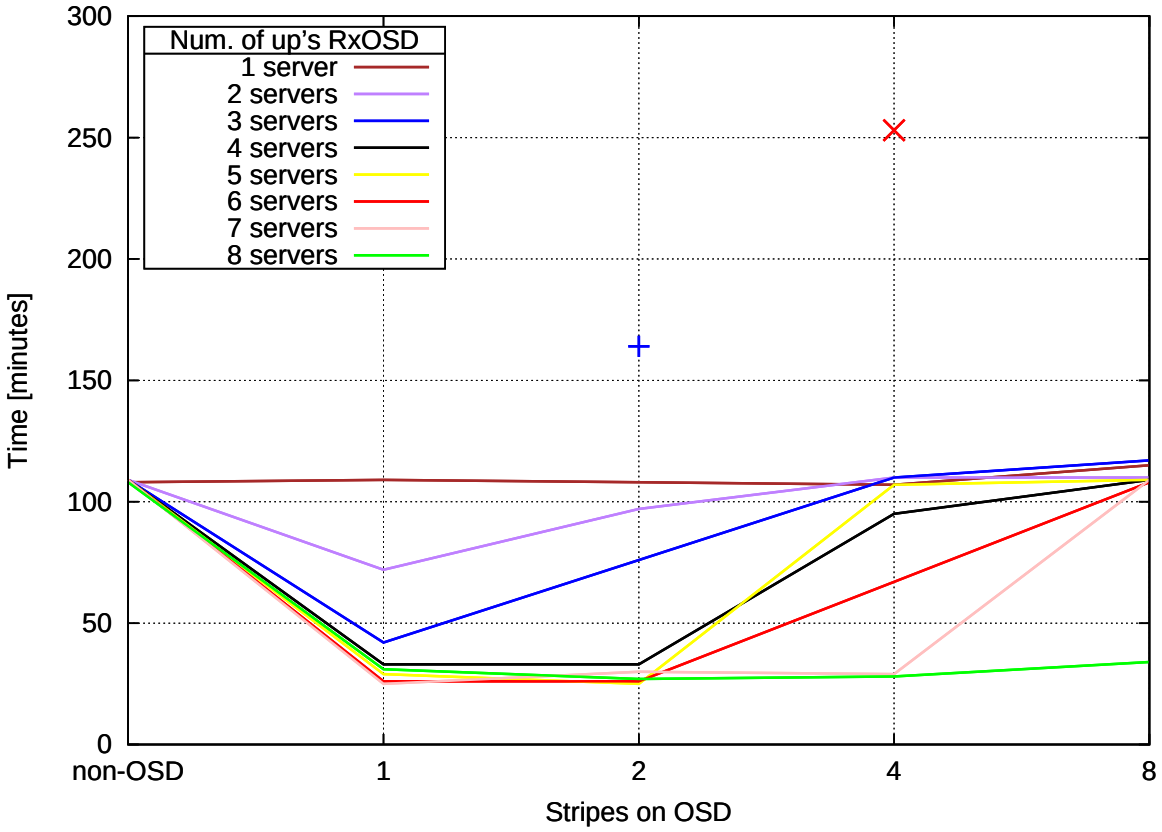

Figure 4: Stripe size impact on the number of running servers

5.3 Client Number Impact

The number of clients was growing throughout this test, starting with one and ending with 27. Rx-OSD was configured with two stripes and all eight rxosd servers were operational. Previous measurements have shown that those settings provided the best stability and were probably most suitable for the University's production environment.

Fig. 6 shows that clients act as bottlenecks at first, but a classic OpenAFS fileserver becomes the real bottleneck quite early as opposed to Rx-OSD, which gives almost constant throughput across the whole range.

5.4 Disk and Network Interface Load

Figures 8 and 7 show the development of disk and network interface load as measured at Dom0. Measurements were taken with Rx-OSD active, 27 clients and 8 rxosd servers.

Graphs prove that network interface throughput represented no real bottleneck in the testing environment as opposed to disk system performance, which had a significant impact on test results.

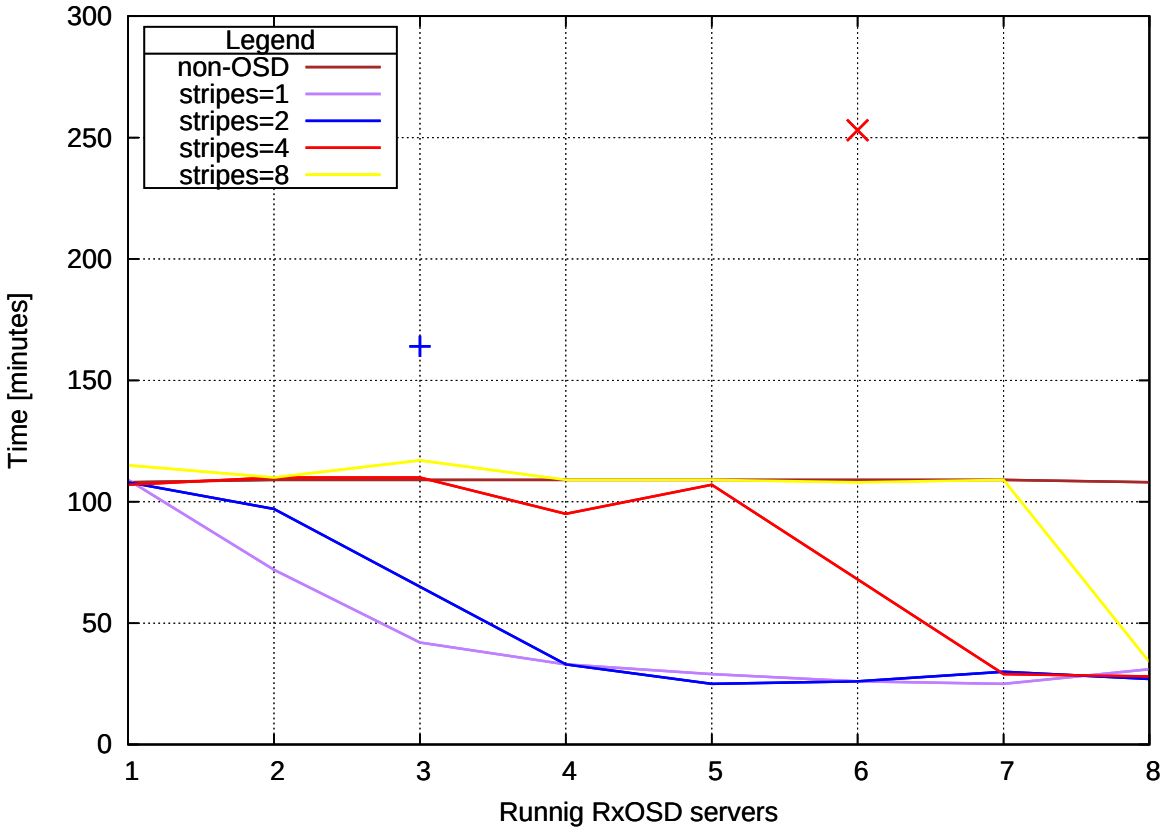

Figure 5: Impact of the number of running servers on stripe size

6 Conclusion

Note that virtualization of the fileserver infrastructure implies certain overhead and it is reasonable to expect that absolute throughput will be higher in a production environment. Still, for a relative comparison between Rx-OSD and classic OpenAFS files servers, the testing environment is adequate.

The main goal of the project consisted in getting acquainted with Rx-OSD technology and assessing possible benefits of its deployment across the infrastructure of the University of West Bohemia in Pilsen. Results show that there actually is great potential for increasing throughput and improving the used value of OpenAFS. On the other hand, it is necessary to note that Rx-OSD technology is rather complicated and there are additional operational and administrative requirements.

Rx-OSD extensions worked reliably throughout the test and there were no unexpected problems or outages. Rx-OSD extensions for AFS have already been deployed in several large AFS cells (RZG, DESY,...) and although it is still a development product, it is very stable. According to an announcement at the European OpenAFS Conference held at the

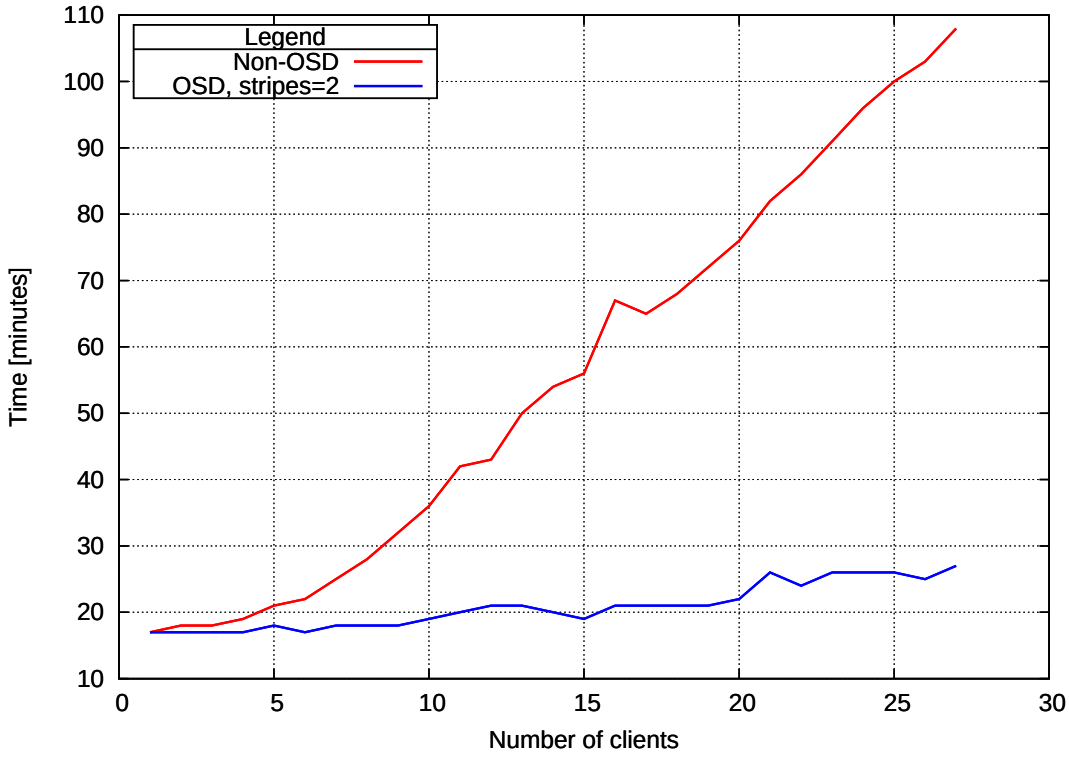

Figure 6: Number of clients impacting throughput

University of West Bohemia[5] Rx-OSD is going to be integrated in OpenAFS by version 1.10, planned for the first quarter of 2011.

[5]European AFS & Kerberos Conference 2010: `http://afs2010.civ.zcu.cz/`.

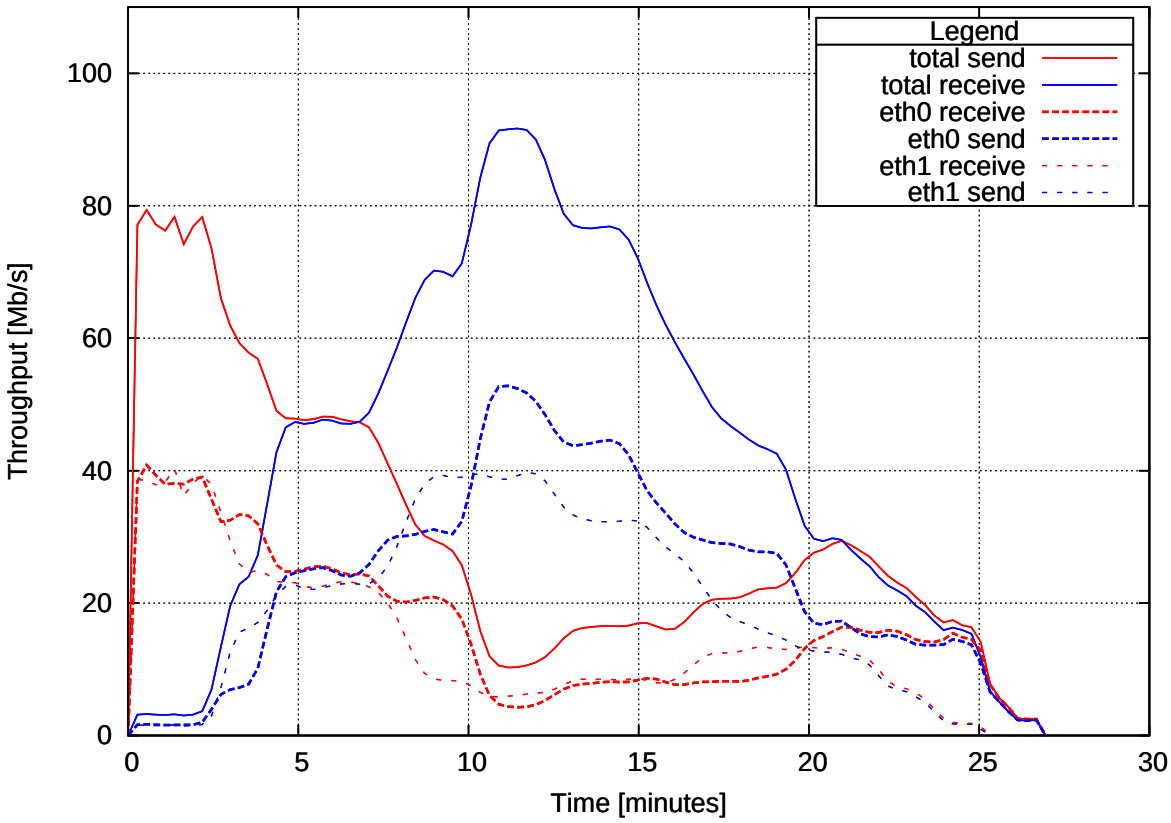

Figure 7: Network interface load

References

[1] Electronic resource page for the Project
`http://support.zcu.cz/index.php/CIV:Granty/Cesnet_293_2009`

[2] OpenAFS Project page
`http://www.openafs.org/`

[3] OpenAFS-OSD Project page
`http://pfanne.rzg.mpg.de/trac/openAFS-OSD`

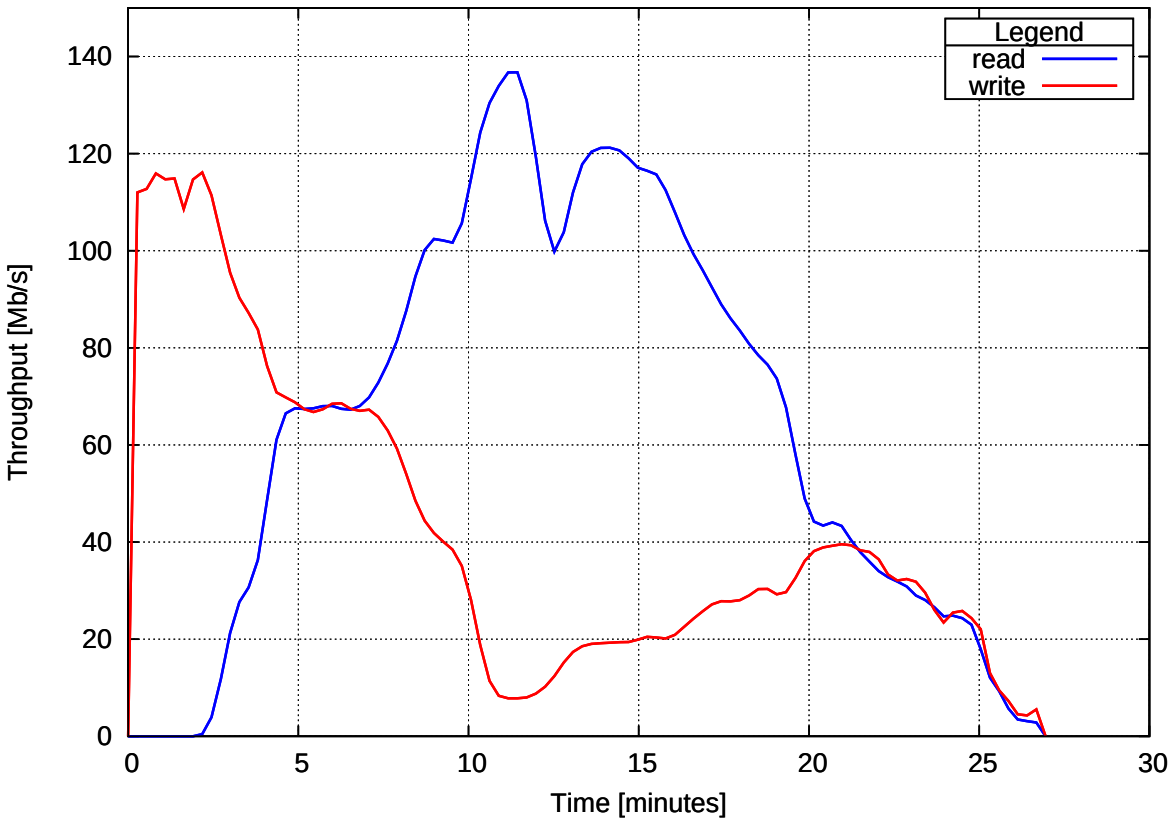

Figure 8: Disk load