D5.2– Report on Parallel and Distributed Filesystems on the BSC ARM Prototype Version 1.0

Document Information

Contract Number	288777
Project Website	www.montblanc-project.eu
Contractual Deadline	M12
Dissemintation Level	PU
Nature	Report
Coordinator	Alex Ramirez (BSC)
Contributors	Isaac Gelado (BSC), Jonathan Marti (BSC), Toni Cortes (BSC)
Reviewers	Chris Adeniyi-Jones (ARM)
Keywords	File System, HPC, Energy-efficiency

Notices: *The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 288777*

Change Log

Version	Description of Change
v0.1	Initial draft released to the WP5 contributors
v0.2	Draft sent for internal review
v1.0	Final version sent to EU

Contents

Executive Summary

Nowadays, topmost high performance computing (HPC) clusters use scalable distributed parallel file systems that are able to stripe data over multiple servers to achieve high performance also in I/O. From our experience in the Storage Systems Research Group and given the requirements of the project, we chose a parallel file system that is very common, open-source and POSIX compliant: Lustre; as the first candidate to provide high performance I/O on our ARM cluster.

Given that Lustre is open source we are able to access its code and adapt it to our Linux kernel (provided by SECO) for the ARM architecture. In the meantime, we focused on the client part since the server part is not expected to be executed in the ARM cluster. Thus, we started spending our efforts on adapting the code of the Lustre client modules to our specific kernel version. As expected, we got some important compilation errors due to kernel incompatibilities, since last maintenance release of Lustre is compatible with kernel versions up to 2.6.32 whereas our current version is 2.6.36 (based on an Ubuntu Maverick distribution).

However, we lately got a first patched version of the Lustre client that can do mostly all of the most common and important POSIX operations. The problem is that due to circumstances we still do not control, when executing some specific deletion operations causes the client to hang.

From this deliverable on we will more efforts to try to understand what is really happening, whether it is an issue related with the architecture or the changes we performed that still need to be further reviewed.

1 Introduction

When building a high-performance computing (HPC) cluster, we can choose among three main categories of file systems: the Network File System (NFS) [NFS]; storage area network (SAN) [SAN] file systems, and parallel file systems.

NFS is generally considered easy to use and most system administrators and working with Linux are familiar with it. However, NFS does not scale well across large clusters and it lacks parallel I/O so that achieving good performance results in a large cluster is not feasible. SAN file systems can provide extremely high data throughput but they are usually implemented with Fibre Channel storage. This is a drawback not only because it makes the architecture to become very costly, but also because every node connected to the SAN must have a Fibre Channel host bus adapter (HBA) to connect to the Fibre Channel network (which is not the case of our Tegra boards). On the other hand, parallel file systems only require a few nodes connected to the storage, known as I/O nodes, which serve data to the rest of the cluster nodes, and mostly all of them offer the advantages commented in the following section.

1.1 Parallel file systems

The main advantages a parallel file system can provide include a global name space, scalability, the capability to distribute large files across multiple nodes and the capability of accessing multiple files or a single split one in parallel.

A global name space provides a unique vision of the filesystem, so that all the client nodes see the same directory tree and file paths and also offering a hardware-agnostic access to data since they no longer need to know where it physically resides. Thus, the filesystem becomes potentially more flexible to changes and makes the data organization easier to the administrator and facilitates the data access specification to the programmers.

Scalability is a must if we want our cluster to be able to grow regarding either the number of client nodes, the storage itself (i.e. increasing storage capacity) or I/O bandwidth. As mostly every HPC supercomputer, our ARM cluster will surely grow regarding the number of nodes from time to time, thus providing providing more computational resources that hopefully will serve to a continuously increasing number of applications running on them. The more applications are executing in the cluster, the more I/O accesses will be performed. Thus the file system must be able to scale not only regarding the number of parallel accesses that can attend, but also be flexible to add more storage resources and keeping I/O bandwidth accordingly.

The capability to distribute files across multiple nodes is also important since it is one of the key features for providing efficient and balanced parallel access to them. If a big file is split in blocks or objects that are spread among several servers, a multi-threaded application can access them from several nodes simultaneously with real parallelism (or from different applications at the same time in parallel). Consequently, offering parallel access is obviously mandatory in order to benefit from such file distribution.

Finally, besides these previous commented features and taking context of Mont-Blanc project into account, we also have to consider two extra features: POSIX compliance and Open Source. We want our file system to be Open Source due to the Mont-Blanc global requirements, and we also want a POSIX compliant filesystem since most of the users that will use the ARM cluster expect their applications to work as they were executed on their own desktops, i.e. they expect the file system to behave as common ones: EXT3/EXT4, ReiserFS, etc. which are implemented under the POSIX standards.

In section 2 we will compare the main three file systems we know from the HPC context: GPFS [GPF, Wika], PVFS [PVF, Wikc] and Lustre [Wha, Wikb]); and we will explain which

one we chose for the project taking all the previous requirements into account.

1.2 Our main goal

Once we choose a filesystem to start with, our main goal is to achieve a functional filesystem client for the ARM architecture in order to provide storage access to every node of the ARM cluster/supercomputer. The filesystem services of the server side are considered to be out of the scope of the project, i.e. they are not supposed to be executed on the ARM architecture since they are not part of the ARM cluster itself. Therefore, we can focus on the client side where, as the ARM architecture representative, we have a NVIDIA Tegra2 [teg] board with Ethernet network connectivity and a kernel 2.6.36 provided by SECO [SEC] that can be booted on it.

In section 3 we explain how we configured our test environment to achieve our first goal, and in section 4 we summarize how we are dealing with the incompatibilities between the filesystem and our current Linux kernel.

2 Choosing a parallel filesystem

The three topmost common parallel filesystems for HPC are GPFS, PVFS and Lustre.

In table 1 you can see the main features we are considering (commented in section 1) and for each of the parallel file systems whether it covers them or not.

Table 1: Parallel file system comparison

Feature	*GPFS*	*PVFS*	*Lustre*
Global name space	✓	✓	✓
Scalability	✓	✓	✓
File distribution	✓	✓	✓
Parallel access	✓	✓	✓
POSIX compliance	✓		✓
Open Source		✓	✓

As you can see, PVFS2 would be a good option but we discarded it because it is not fully POSIX compliant so that it could become a potential drawback for users used with common Linux filesystems or NFS. On the other hand, GPFS covers all the common requirements but we have discarded it because of its proprietary license.

Consequently, we chose Lustre [BS02, SS05] given that its key features are:

Scalability: Lustre scales up or down with respect to the number of client nodes, disk storage and bandwidth. Currently, Lustre is running in production environments with up to 26,000 client nodes [Sch03], with many clusters in the 10,000-20,000 client range. Other Lustre installations provide aggregated disk storage and bandwidth of up to 1,000 OSTs running on more than 450 OSSs. Several Lustre file systems with a capacity of 1 PB or more [EMWB12] (allowing storage of up to 2 billion files) have been in use since 2006.

Performance: Lustre deployments in production environments currently offer performance of up to 100 GB/s. In a test environment, a performance of 130 GB/s and 13,000 creates/s has been sustained. Lustre single client node throughput has been measured at 2 GB/s (max) and OSS throughput at 2.5 GB/s (max). Lustre has been run at 240 GB/sec on the Spider file system at Oak Ridge National Laboratories.

Open source: Lustre is open source, licensed under the GNU GPL [GPL].

POSIX compliance: The full POSIX test suite passes on Lustre clients. In a cluster, POSIX compliance means that most operations are atomic and clients never see stale data or metadata.

3 Environment

Our current scenario (figure 1) is quite simple since we are not interested in installing the Lustre server part over an ARM architecture, and our first goal is to achieve a functional Lustre client (i.e. we do not have to concern about performance until we have a functional version).

We use a laptop for the basic services, a NVIDIA Tegra2 board as the ARM client side, and a x86_64 node for the server side.

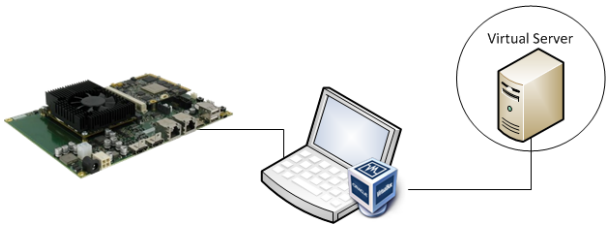

Figure 1

3.1 Basic services

A laptop is enough to configure the basic services (VirtualBox [Vir], NFS, DHCP and TFTP) that allows us to boot and OS in the Tegra2 board and to communicate the serve as a router to communicate both server and client.

3.2 Server

The Lustre server side is currently deployed using VirtualBox. The guest OS of the Virtual Machine is a Linux Ubuntu Lucid [Ubu].

The guest OS forms a subnet with the laptop by setting its IP address via the VirtualBox DHCP service (using a virtual network interface).

Regarding the Lustre services, for our first purposes we started with the simplest scenario with one MetaData Server (MDS) in charge of managing the global namespace of the filesystem and the metadata operations, and one Object Storage Server (OSS), which is the responsible of handling the stored data and I/O operations.

3.3 Client

The Lustre client must be executed in a Tegra2 board as one of the nodes of the ARM cluster. To achieve this, the Tegra2 network interface forms a subnet with the laptop by setting its IP address via the DHCP service. Then, TFTP is used to boot the Linux Ubuntu Maverick provided by SECO (with a patched kernel 2.6.36). Finally, the basic OS filesystem is exported to the Tegra2 board via NFS from the laptop as well.

3.4 Connecting Lustre client and server

In order to communicate the Tegra2 board subnet (client side) with the Virtual Machine (server side) subnet, we configured simple IP forwarding rules in the laptop that allows to do that.

4 Porting Lustre to ARM

The main difficulty to face this task is the kernel compatibility. Nowadays, current versions of Lustre only support kernels up to version 2.6.32. Therefore, when we try to compile the Lustre client modules with our kernel 2.6.36 we are getting lots of errors. Moreover, we started knowing nothing about the sources of Lustre itself, which means an extra effort when trying to understand the code so we can change it correctly.

We started with Lustre 1.8.5 in our first efforts to make Lustre client modules work on the Tegra2 board with the given kernel. During the configuration process (before compiling) we disabled some parts (libcfs-trace, debug, assert) that were not necessary neither for the compilation of the client modules nor for the functional tests. During the first round of the compilation (without modifying the sources) we got 19 major errors and some warnings, but we finally got a patched version of the Lustre 1.8.5 sources. With this beta version we were able to perform metadata operations such as create or remove a file/directory, but we were not able to perform write operations on created files. However, we stop our progress on that because another Lustre version was agreed to be used: version 2.1, which is the current maintenance release from WhamCloud.

With this newer version of Lustre, which theoretically fits better with our kernel version, we actually got less errors in the first attempt of compiling the code. In fact, some of them are shared with those we obtained with previous Lustre version. The problem is that there are very tricky and sometimes require understanding both Lustre source code and kernel code.

Below there are some examples of errors we encountered and how we are dealing with them, and afterwards we present the current status of our first tests.

4.1 Error Examples

4.1.1 VFS

Linux filesystem structures change from time to time. Sometimes the operations are updated with new headers and implementations. For instance, fsync function in kernel 2.6.32 had this header:

```
int vfs_fsync(struct file *file, struct dentry *dentry, int datasync);
```

Whereas in kernel 2.6.36 has this one:

```
int vfs_fsync(struct file *file, int datasync);
```

As you can see, the dentry structure is not needed to be passed anymore (it is accessed from the file structure itself). This change in the kernel implied modifying several related files from Lustre source code, such as *lustre/llite/file.c*, or *lustre/lvfs/lvfs_linux.c*.

4.1.2 Sysctl

The *ctl_name* and strategy fields are unused, now that *sys_sysctl* is a compatibility wrapper around */proc/sys*. Thus, in *lnet/lnet/router_proc.c* or *lustre/obdclass/linux/linux-sysctl.c*, we had to comment all its appearances.

4.1.3 Locks

In kernel 2.6.36, the locks used in the *fs_struct* where modified from *rwlocks* to *spinlocks*. From official kernel GIT repository [Ker]: *struct fs_struct.lock is an rwlock with the read-side used to protect root and pwd members while taking references to them. Taking a reference to a path typically requires just 2 atomic ops, so the critical section is very small. Parallel read-side operations would have cacheline contention on the lock, the dentry, and the vfsmount cachelines, so the rwlock is unlikely to ever give a real parallelism increase. Replace it with a spinlock to avoid one or two atomic operations in typical path lookup fastpath.*

Therefore, *write_lock* and *write_unlock* were not used so in file *lustre/include/linux/lustre_compat25.h* we had to change them to use *spin_lock* and *spin_unlock* instead.

4.2 Results and current status

Finally, we have just developed a patched Lustre client version that can be compiled and loaded in our kernel 2.6.36 in the ARM Tegra2 board, and we successfully mount a Lustre filesystem from the Lustre server (the one in the Virtual Machine).

Our first tests have been focused on executing operations on files and directories, omitting operations related with links. In the meantime we can create, read and write files, and create directories as well; but we are encountering some issues in deletion operation which are propagated to the server but cause the client to hang.

5 Conclusions and future work

Although latest versions of Lustre are not compatible with our current kernel version for the Tegra2 board, we strongly believe that we can solve the compilation errors of the Lustre client modules and make a patched functional version (at least with the very basic scenario we start with). We did a first patched version of the Lustre 1.8.5 client and we could perform metadata operations with a common basic scenario.

Lustre version was agreed to be changed and we chose version 2.1, that is the latest maintenance release from WhamCloud. With this version we obtained less errors but still some of the previous ones reappeared, but we finally have just developed a first patched version of the Lustre 2.1 client part and we have started executing basic tests to see what it is able to do. We omitted advanced operations about links and focused on the common ones related with files and directories. In the meantime we can create, read and write files, and create directories as well; but the client still presents some bugs in deletion operations, which cause the client to hang although they reach the server and produce the expected changes in the filesystem.

If we cannot get a solution, we probably could try to compile a kernel 2.6.32 compatible with the Tegra2 board and check whether we can directly compile Lustre sources on it with more insignificant or minor changes.

Once we get a proved fully functional Lustre client for the given ARM architecture with the SECO kernel, we will be able to proceed with the performance tests both on our current basic scenario and on the ARM cluster as well.

References

[BS02] P.J. Braam and P. Schwan. Lustre: The intergalactic file system. In *Ottawa Linux Symposium*, page 50, 2002.

[EMWB12] M.A. Ezell, R. Mohr, J. Wynkoop, and R. Braby. Lustre at petascale: Experiences in troubleshooting and upgrading. Technical report, Oak Ridge National Laboratory (ORNL); Center for Computational Sciences, 2012.

[GPF] Gpfs. `www.ibm.com/systems/software/gpfs`.

[GPL] Gpl license. `www.gnu.org/copyleft/gpl.html`.

[Ker] Kernel git. `git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git`.

[NFS] Network file system (nfs) version 4 minor version 1 protocol. `tools.ietf.org/html/rfc5661`.

[PVF] Pvfs. `www.pvfs.org`.

[SAN] Storage area network. `en.wikipedia.org/wiki/Storage\_area\_network`.

[Sch03] P. Schwan. Lustre: Building a file system for 1000-node clusters. In *Proceedings of the 2003 Linux Symposium*, pages 400–407, 2003.

[SEC] SECO. `www.seco.com`.

[SS05] R.S. Studham and R. Subramaniyan. Lustre: A future standard for parallel file systems. In *Invited presentation at International Supercomputer Conference. Heidelberg, Germany*, 2005.

[teg] Nvidia tegra2 specification. `www.nvidia.com/object/tegra-superchip.html`.

[Ubu] Ubuntu. `www.ubuntu.com`.

[Vir] VirtualBox. `www.virtualbox.org`.

[Wha] WhamCloud. Lustre documentation. `wiki.whamcloud.com/display/PUB/Documentation`.

[Wika] Wikipedia. Gpfs. `en.wikipedia.org/wiki/IBM\_General\_Parallel\_File\_System`.

[Wikb] Wikipedia. Lustre. `en.wikipedia.org/wiki/Lustre\_%28file\_system%29`.

[Wikc] Wikipedia. Pvfs. `en.wikipedia.org/wiki/Parallel\_Virtual\_File\_System`.