Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Lessons Learned about Metadata Performance in the PRACE File System Prototype

Ernest Artiaga [a], Toni Cortes [a,b]

[a] *Barcelona Supercomputing Center, Barcelona, Spain*

[b] *Universitat Politècnica de Catalunya, Barcelona, Spain*

ernest.artiaga@bsc.es, toni@ac.upc.edu

Abstract

The purpose of this white paper is to document the measurements obtained in PRACE-2IP regarding file system metadata performance, and to assess mechanisms to further improve such performance. The final goal of the task is to identify the open issues related to file systems for multi-petascale and exascale facilities, and propose novel solutions that can be applied to Lustre, enabling it to manage a huge number of files on a system with many heterogeneous devices while efficiently delivering huge data bandwidth and low latency, minimizing the response time.

The performed tasks being reported included the observation, measurement and study of a large scale system currently in production, in order to identify the key metadata-related issues; the development of a prototype aimed to improve the metadata behaviour in such system and also to provide a framework to easily deploy novel metadata management techniques on top of other systems; the measurement and study of specially deployed Lustre and GPFS prototypes to validate the presence of the metadata issues observed in current in-production systems; and finally the porting of the framework prototype to test novel metadata management techniques on both a production system using GPFS and the PRACE Lustre prototype facility at CEA.

Executive Summary

This white paper documents the research and development carried on within PRACE-2IP. The main goal of the task is to identify and address some open issues in file systems for multi-petascale and exascale facilities, aiming to the development of solutions that can be applied to the Lustre file system.

The addressed issues focus on metadata management. Metadata handling relates to dealing with huge numbers of files and their hierarchical organization according the user’s view (including directory management and file attributes).

The performed tasks being reported included the observation, measurement and study of a large scale system currently in production, in order to identify the key metadata-related issues; the development of a prototype aimed to improve the metadata behaviour in such system and also to provide a framework to easily deploy novel metadata management techniques on top of other systems; the measurement and study of specially deployed Lustre and GPFS prototypes to validate the presence of the metadata issues observed in current in-production systems; and finally the porting of the framework prototype to test novel metadata management techniques on both a production system using GPFS and the PRACE Lustre prototype facility at CEA.

We have observed that in both Lustre and GPFS there are some scalability issues that reduce the performance of metadata operation when many files are used by the applications or when the number of accessing clients grows. The most important observation is that the number of files needed for the problem to appear is only a few hundreds and the number of clients a few dozens. This clearly shows that the problem needs to be addressed.

The study also shows that the scalability and behaviour of Lustre and GPFS differ in several aspects, that being a consequence of the different approaches they use for metadata handling (e.g. distributed consistency management in GPFS vs. centralized metadata handling in Lustre).

A portable intermediate layer, COFS (Composite File System) has been developed to provide a portable and convenient way to explore alternative metadata handling techniques on top of parallel file systems.

After our mechanism has been added to the GPFS system, we have observed that the decrease in performance that appears in the evaluated cases disappears making the system much more scalable with the number of files and clients. The main reason for this beneficial effect is that our middleware is able to convert not optimized cases (from GPFS point of view) into the optimized cases of GPFS.

The same mechanisms have been also tested on top of the Lustre file system. We have observed moderate improvements in the situations were Lustre suffered performance degradations, such as parallel creations in shared directories and parallel open operations.

The techniques and algorithms used by COFS to handle metadata and directory contents appear to be an effective way to manage the file system metadata in large scale systems. In situations where the overhead introduced by using an additional layer is not acceptable, such algorithms and techniques could be integrated inside the file system cores

.

1. Introduction

The purpose of this white paper is to document the measurements obtained in Task 12.4 of the FP7-funded project PRACE-2IP regarding file system metadata performance, and to assess mechanisms to further improve such performance. The final goal of the task is to identify the open issues related to file systems for multi-petascale and exascale facilities, and propose novel solutions that can be applied to Lustre, enabling it to manage a huge number of files on a system with many heterogeneous devices while efficiently delivering huge data bandwidth and low latency, minimizing the response time.

Parallel file systems usually keep pace with high performance computing clusters by incorporating optimizations targeted at specific workloads. Unfortunately, the growing number of large scale applications increases workload heterogeneity, generating a gap between how file systems work, and how users expect them to behave.

High performance computing is rapidly evolving into large aggregations of computing elements in the form of big clusters. In the last few years, the size of such distributed systems has increased from tens of nodes to thousands of nodes, and the number is still rising. Trying to keep pace with these developments, parallel file systems try to provide mechanisms for distributing data across a range of storage devices and making them readily available to the computing elements.

At the other end, the final user's view of the storage systems has not changed significantly: for the usual case, files are organized in a hierarchical name space, much in the same way as they were placed in a classical local file system using an attached disk in a single computer.

In this classical view, a directory was often tacitly used as a hint to indicate locality. Indeed, it is not surprising to find a directory containing files that are going to be used together, or in a very related way (for example, a directory containing a program's code, the corresponding executable, some configuration files and maybe the output of its execution).

Trying to exploit this affinity, file systems tended to group together the management information (the metadata) about directory contents. This was favoured by the fact that this information is relatively small, so that it is feasible to pack together information about access permissions, statistics, and also the physical location of the data, not only for a single file, but also for a set of related files (i.e. for files present in the same directory).

The approach of treating directory contents as a group of related objects with similar properties is still present in modern file systems. It is common to see parallel file systems to use directories as mount points to access different volumes or partitions with different functionalities or, in finer grain file systems, to be able to specify directory-wide rules that apply to directory contents.

The problem appears when, in large clusters with parallel file systems, directory contents are semantically related according user's view, without that meaning that they are going to be used together or in similar ways from the file system perspective. Reusing the example above, now the program code in a directory will be compiled and linked, probably in a single node, to generate a binary that will be read and executed simultaneously in 2,000 nodes, each of them generating an output file to be left in the same directory, which will be collected and read by a single post-processing tool generating some summary information. Files are indeed related, but patterns of use are totally dissimilar.

Parallel file systems trying to keep close the apparently related fragments of metadata will end up trying to keep consistent a relatively small pack of miscellaneous information (which is not easy to distribute and share) while it is being simultaneously used, and probably modified, by a large number of nodes. As a result, the pressure on metadata handling in large scale systems will rise up, producing delays comparable to the actual data transfer times from and to the storage systems, and jeopardizing the overall system performance.

The metadata issue has been confirmed by observations in actual systems, showing that lack of synergy between file systems and the multiplicity of applications running on them is increasing the pressure on metadata management, up to the level of becoming a significant performance-killer.

It is worth while mentioning that all modern file systems aiming to provide a storage solution for very large scale systems share similar goals and face related issues. For this reason, even if the final goal is to provide novel mechanisms to enrich the Lustre prototype, we have studied and taken advantage of the experience of large-scale production facilities also using different file systems (namely GPFS [1]). The results we have obtained could also be applied directly (or ported with minimum effort) to file systems other than Lustre or GPFS.

This white paper is structured into three main sections. Firstly, we describe the COFS (Composite File System) framework. COFS allows a detached handling of file system metadata, providing the applications with a conventional file system namespace hierarchy (according to user's view), while internally reorganizing the location of actual files in the underlying file system to avoid metadata conflicts caused by typical file usage patterns. Then, we summarize the results of applying the COFS technology to a production cluster using GPFS, showing how adequate metadata handling can improve the overall file system performance. Finally, we analyze the metadata performance of the last Lustre prototype in a dedicated PRACE test-bed cluster to see if metadata issues are present, and assess the feasibility of COFS as an adequate tool to mitigate them.

2. The COFS Framework

In general, file systems try to adapt to the new demands by specializing and targeting specific kinds of workloads. As a counterpart, there is a cost in terms of performance for non-optimized cases.

A usual way to deal with non-optimized cases is to add even more specific optimizations, either by means of file system specific modifications (e.g. parallel creations policies in PVFS2 [2]), or by means of a middleware targeted to fulfil the needs of specific classes of applications (e.g. MPI-IO implementations using hierarchical striping for Lustre [3]). Unfortunately, this approach does not deal with performance penalties caused by unforeseen or inadequate access patterns from arbitrary applications, which are likely to occur in heterogeneous workloads.

We have decided to take a novel approach to mitigate the effects of non-optimized situations. Our technique to handle the metadata issues consists of placing a layer on top of the file system for decoupling the user view from the actual low-level file system organization, allowing us to convert the application access patterns into something that can be dealt with by underlying file system without harming the performance (instead of trying to adapt the file system to any possible workload). In other words, the mechanism consists of improving the file system behaviour by avoiding bottlenecks caused by application patterns, instead of focusing only on optimizing the file system for a very specific workload.

Running on top of the file system (instead of integrating new code into it) makes development much easier and simplifies the integration of other state-of-the-art technologies which may help to address the issues under study. Additionally, it allows us to easily adapt the same solutions to multiple file systems and environments.

Given the observations on metadata behaviour in large scale production file systems, we have developed the Composite File System (COFS) as a prototype to validate the assumption that it is possible to boost the performance of a parallel file system by decoupling metadata, and name space handling, from the underlying directory layout. COFS acts as a metadata management layer on top of the file system, enabling the improvement of its behaviour under high pressure situations without harming the other aspects of file system performance.

In particular, COFS has been used to evaluate the performance benefits of separating the user view from the underlying file system structure on our systems under study. In this section we describe its main features, focusing on the implementation aspects that are relevant to explain the experimental results.

2.1.Principles

The main goal of the current COFS prototype is to be able to decouple the user view of the file hierarchy, the metadata handling and the physical placement of the files. In particular, this allows us to present the user with a virtual view of the file system directories, while the actual layout can be optimized for the underlying file system.

Regarding functionality, POSIX compliance was a strong design requirement. Apart from the fact that this is still the dominant model for most applications, our generic goal was to reduce the operational restrictions of underlying file systems; so, limiting the semantics was not an option: if POSIX semantics is required and the underlying file system supports it (as is the case with Lustre), then COFS should also be able to deal with it. The current implementation fully supports POSIX except for some functionality that was not relevant for our present work (specifically, named pipes).

Another important point to mention is that COFS is implemented as a user-level FUSE (Filesystem in USErspace [4]) daemon, and it is independent from the underlying file system. The reasons for this are two-fold. First, even if one of the original motivations of this work was mitigating potential performance drops on a specific file system, we believe that equivalent issues affect other file systems; so, the solving mechanism should be generic enough to be applied to any file system. Second, we plan to deploy our framework in production-grade clusters, and having a user-land drop-in package without hard requirements on kernel modifications, configurations or complex software packages makes it much easier to have access to such environments, as the potential impact on the rest of the system is minimal.

The COFS framework does not directly deal with low-level data storage. There is no explicit management of disks, blocks or storage objects: COFS simply forwards data requests to the underlying file systems and indicates an appropriate low level path when a file is created. Then it is up to the underlying file system to take the decisions on low-level data server selection, striping, block/object placement, etc. In this sense, COFS is not a complete file system, but a tool to leverage the capabilities of underlying file systems.

On the contrary, COFS does take the responsibility on metadata management. By metadata we specifically mean access control (owner, group and related access permissions), symbolic and hard link management, directory management (both the hierarchy and the individual entries) and size and time data for non-regular files (sizes and access time management for regular files rely on the underlying file system).

Regarding security aspects, COFS relies on the underlying infrastructure: local file system operations are already protected by FUSE's in-kernel support; communications with the metadata service make use of the authentication mechanisms provided by the Erlang/OTP environment [5].

2.2.Architecture

Figure 1 shows how the COFS framework integrates with an existing file system environment. An original parallel file system is served by 3 file system servers, and n clients contact such servers through the network. COFS introduces an extra layer on each file system client, providing a virtual view of the file system layout. Metadata information is handled by an additional server node. It is important to mention that even though we use a single metadata server in the current stage of implementation, this is not forced by design, and the framework also admits a distributed metadata service.

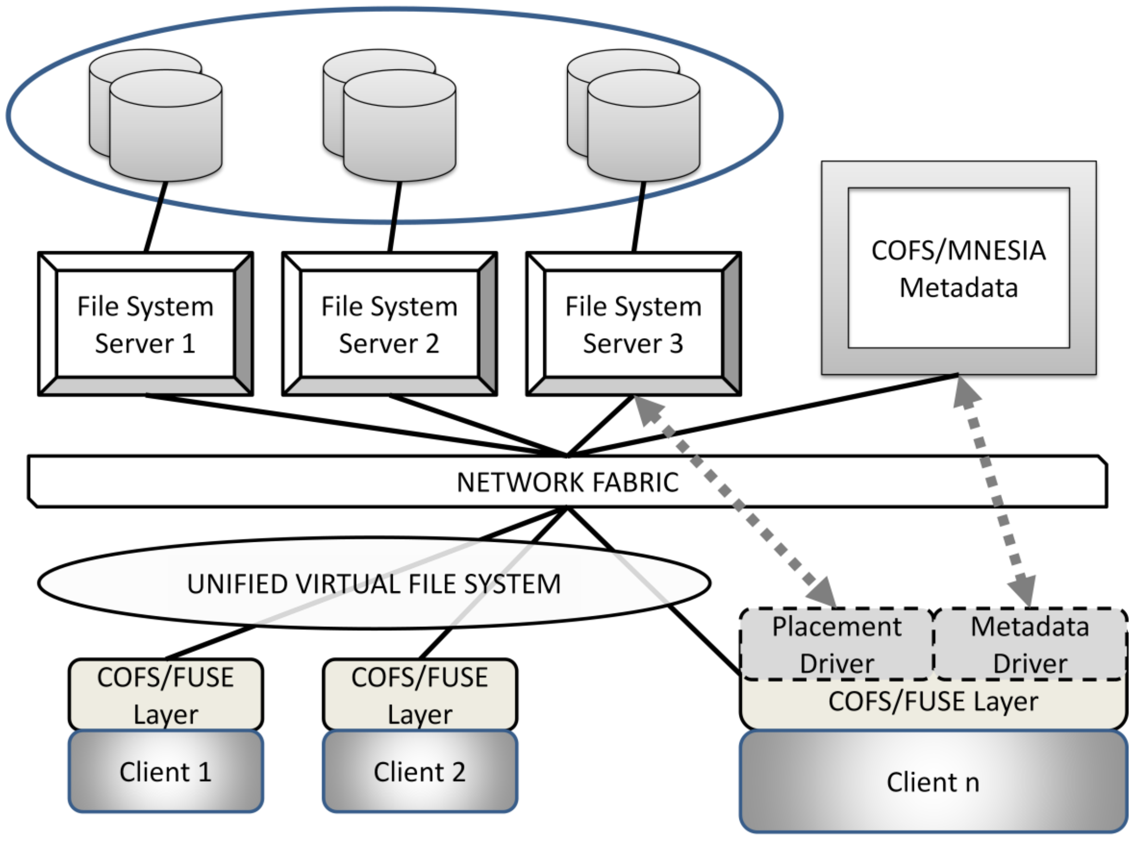

Figure 1 Parallel file system architecture augmented with COFS virtualization layer

The COFS layer on each node offers a file system interface, so it can be mounted as any other file system. The implementation is based on FUSE, which provides a kernel module that exports VFS-like callbacks to user-space applications. The decision to use FUSE was driven by both portability and level of support, as well as ease of development. FUSE is a standard component of current Linux kernels (also available for other operating systems) and provides a stable platform for implementing a fully functional file system in a user-space program. Considering our experimental goals, the downside of missing some kernel-level information that is not exported or forwarded to user level, and possibly minor efficiency losses, is largely compensated by having a drop-in environment that can be used in most Linux boxes without requiring specific kernel modifications.

Once intercepted by FUSE, file system requests are internally diverted by COFS into two different modules (the placement and metadata drivers) with well-defined interfaces. The placement driver is responsible for mapping the regular files into the underlying file system(s), while the metadata driver takes care of hard and symbolic links, directories, and generic attributes. Some operations need the collaboration of both drivers: for example, creating a file involves creating an actual regular file on a convenient location (a placement driver responsibility) and updating the proper entries in the directory (done by the metadata driver). So, an interface is also defined for communication between both drivers.

2.3.Metadata service details

How to handle metadata is an important factor that must be considered carefully when dealing with parallel and distributed file systems. This also applies to the COFS framework, as the separation of metadata and name space handling from the actual data layout is a key feature to obtain performance benefits.

Currently adopted solutions for metadata range from having a single centralized metadata server (like Lustre up to version 2.3) to fully distributing metadata (such as GPFS). While the first have simplicity on their side, centralized approaches are suspected to become a bottleneck and hinder scalability; on the other hand, the latter approach requires distributed locking/leasing techniques to keep the coherence and complex fault detection and recovery mechanisms. Other options lie in the middle, like explicitly dividing the system into smaller partitions (Panasas [6]) or relaxing consistency or caching to avoid synchronization needs (PVFS2 [7]). The distributed directory service for Farsite [8] also explores several techniques for partitioning file system metadata while mitigating synchronization hotspots.

In COFS, we have taken a conceptually centralized approach for metadata because of its simplicity. We dealt with scalability concerns by leveraging the well-know technology of distributed databases: metadata can be seen as small set of tables having information about the files and directories and, in case of need, it could be distributed into several servers by the database engine itself (without the need of explicit file system partitions or separate volumes).

To this end, we chose the Mnesia database, which is part of the Erlang/OTP environment. Mnesia provides a database environment optimized for simple queries in soft real time distributed environments (with built-in support for transactions and fault tolerance mechanisms). Additionally, the Erlang language has proven to be a good tool for fast prototyping of highly concurrent code (the language itself internally deals with thread synchronization and provides support for transparently distributing computations across several nodes).

The current COFS prototype uses a single metadata server running an Erlang node with an instance of the Mnesia database, and the COFS metadata driver on each client simply forwards the requests to the server and handles metadata leases.

The server also keeps the current working set of metadata information as a cache of active objects to reduce the pressure on the backend database engine, using a concurrent caching mechanism similar to the one described by Jay Nelson. Although our performance tests show that a single node is enough to handle the metadata, the used algorithm would be compatible with a distributed multi-node Erlang system, if needed.

3. Metadata Performance in GPFS

Parallel applications on large scale parallel systems expect to be able to perform I/O simultaneously from all the nodes as they would if it was a single node (i.e. efficiently, in parallel and keeping the consistency). Not many file systems are able to provide such support to applications in a reliable way. Together with Lustre, GPFS is another mature file system aimed to large distributed clusters which has been adopted at many high performance computing centres.

The metadata issue has been confirmed by observations in actual systems, showing that lack of synergy between file systems and the multiplicity of applications running on them is increasing the pressure on metadata management, up to the level of becoming a significant performance-killer.

As a matter of fact, the initial deployment of the COFS prototype used a GPFS system instead of Lustre as underlying file system, because the planned installation of the PRACE test bench was delayed. The initial measurements were performed in a GPFS-based production file system, and the lessons learned checked against the behaviour of the Lustre prototype once it was installed.

The following subsections summarize the observations of metadata behaviour in GPFS and the results of applying the COFS infrastructure on top of it.

3.1.Metadata Scalability Issues in GPFS

As Lustre, GPFS offers a standard POSIX interface, while having the possibility to use non-POSIX advanced features for increased performance (e.g. for MPI-IO). Nevertheless, their architectural approaches differ significantly. For this reason, studying its behaviour and comparing it with Lustre’s provides useful insights regarding the different techniques available to deal with large cluster file systems and their impact on performance.

GPFS uses block-based storage (contrary to Lustre’s Object Storage Devices [9]). A typical configuration consists of clients which access to a set of file servers connected to the storage devices via a Storage Area Network (SAN). Metadata is also distributed and consistency is guaranteed by distributed locking, with the possibility of delegating control to a particular client to increase performance in case of exclusive access.

We have been able to observe important performance drops in large production clusters using GPFS with heterogeneous workloads (using from one hundred nodes up to a few thousands), and we have been able to track the causes back to metadata management issues that are related to the way in which applications use the file system:

- Large parallel applications usually create per-node auxiliary files, and/or generate checkpoints by having each participating node dumping its relevant data into a different file; not unlikely, applications place these files in a common directory.
- On the other hand, smaller applications are typically launched in large bunches, and users configure them to write the different output files in a shared directory, creating something similar to a file-based results database; the overall access pattern is similar to that from a parallel application: lots of files are being created in parallel from a large number of nodes in a single shared directory.

In both cases, typical modus operandi ends up creating large amounts of files in the same directory; and very large directories, especially when populated in parallel, require GPFS to use a complex and costly locking mechanism to guarantee the consistency, resulting in far-from-optimal performance. For example, a parallel application spanning across a large number of nodes can use a substantial portion of its execution time creating and writing checkpoint files (significantly greater than what should be expected for the simple transfer of data.) To mention another example, trace collection software, used by computer science researchers to obtain per-node application execution traces for performance analysis, also suffers from this inadequate metadata handling.

As an additional concern, the overhead is not limited to the infringing applications, but affects the whole system, as file servers are busy with synchronization and all file system requests are delayed.

We have conducted a series of experiments in a GPFS cluster to confirm that metadata handling was a significant cause of performance drops. The situation we wanted to evaluate essentially involved parallel metadata operations, so we used Metarates [10] as the main benchmark. Metarates was developed by UCAR and the NCAR Scientific Computing Division, and measures the rate at which metadata transactions can be performed on a file system. It also measures aggregate transaction rates when multiple processes read or write metadata concurrently. We used this application to invoke *create*, *stat*, and *utime* on a number of files from the same directory in parallel. The issues identified in the production GPFS cluster have then been used to locate potential issues in the Lustre prototype.

The measurements for GPFS have been carried out in a cluster of 2,500 IBM JS21 blades, with 2 dual core processors PPC970MP at 2.3 GHz and 8 GB of RAM per blade. The interconnection network is a 1Gb Ethernet, and the file system was GPFS version 3.2.1.

The observations indicate that an important portion of the relatively large operation times is consumed not by actual information being transmitted from the server to the clients, but by consistency-related traffic (even when each process works on a different set of files). That assumption would give our virtualization layer enough room to obtain a good speed-up by reorganizing the file layout and reduce the synchronization needed among GPFS clients.

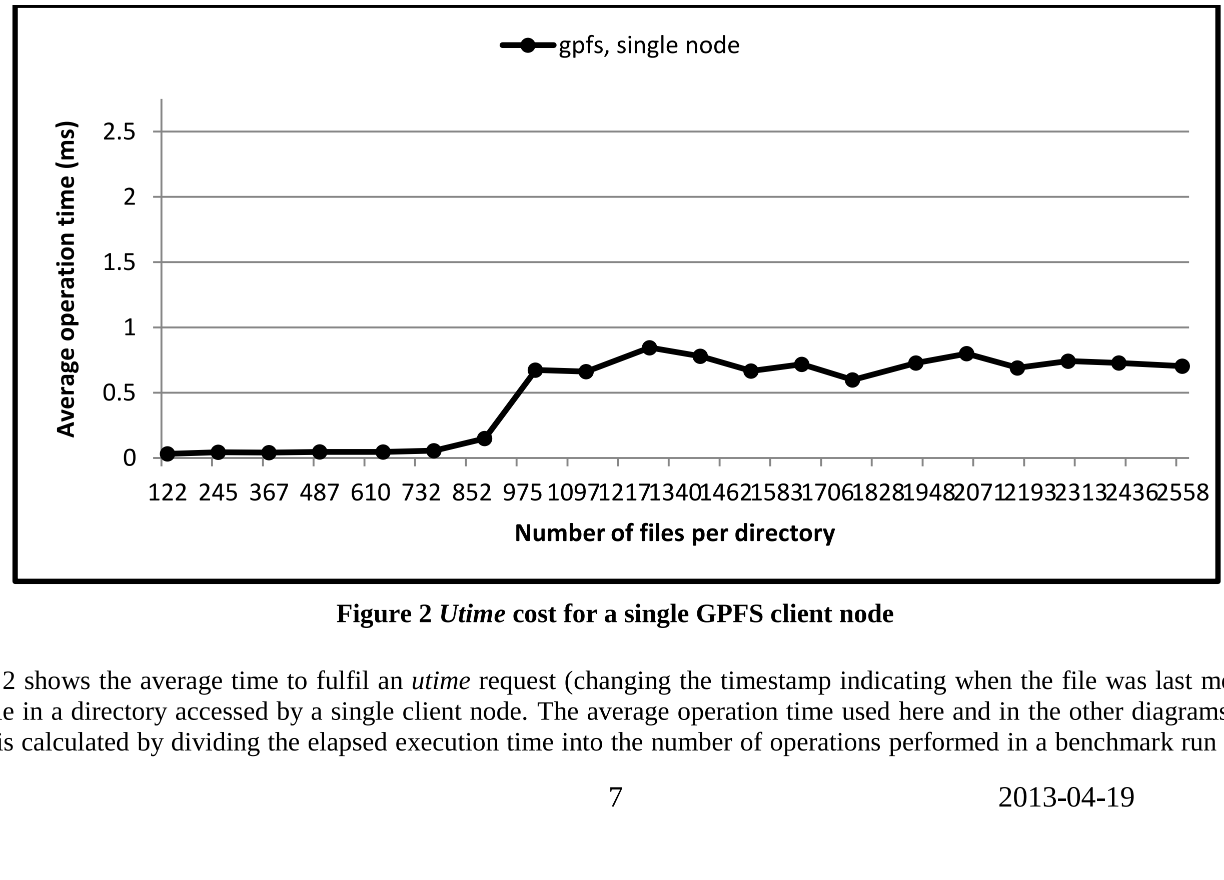

Figure 2 *Utime* cost for a single GPFS client node

Figure 2 shows the average time to fulfil an *utime* request (changing the timestamp indicating when the file was last modified) on a file in a directory accessed by a single client node. The average operation time used here and in the other diagrams of this report is calculated by dividing the elapsed execution time into the number of operations performed in a benchmark run (which, in the case of *metarates*, is the number of files). The low values for small directories (about 45 microseconds) are the cost of the request when the GPFS delegation mechanism is active (control is transferred to the client node, so that operations can be carried on locally, without server synchronization) and represent a lower boundary for the operation cost; beyond 1024 entries per directory, delegation is no longer active and the operation requires remote server intervention, resulting in larger operation times.

The operation average times when performing simultaneous *utime* calls from different nodes progressively decrease when we increase the parallelism, reaching an optimum value for 16 nodes. Beyond that point, the system does not scale anymore and increasing the number of nodes just increases the access conflicts, reducing the performance.

Another interesting observation from GPFS is that the average parallel creation time differs depending on the files being created on a single shared directory or on unique directories per processor. Large delays were observed when creating files from different nodes in the same directory. It is important to mention that each node creates a disjoint set of files; so, the only cause of poor behaviour is the management of a shared metadata management structure: the directory. By changing the way the metadata is organized and handled, it should be possible to reduce this cost to levels similar to using unique directories per processor.

The experiences and observations in the GPFS cluster served two purposes: on one side, they guided the analysis and study of potential metadata-related performance issues in the Lustre prototype for exascale systems (that will be commented in the last section of this document); on the other side, they have inspired the solutions implemented in the COFS framework (explained in Section 2).

The next subsection summarizes the main results obtained when using the COFS framework on top of GPFS.

3.2.Mitigating metadata issues with COFS

We deployed the COFS prototype (version 1.0.5) to verify that it could mitigate some of the metadata performance issues of the parallel file systems and that the benefits obtained where significant enough to compensate the cost of adding an extra layer on top of the file system. Initial experiments were run using GPFS as the underlying system in the same hardware used for the initial measurements (described in section 3.1). The reason behind the use of GPFS instead of Lustre was starting the evaluation of COFS before the PRACE Lustre prototype was ready. Evaluation results over Lustre were performed later and will be discussed in Section 4.2.

Metadata virtualization results

The measures of COFS over GPFS were obtained in the same system described in Section 3.1. Additionally, in this particular cluster, COFS uses IP over Myrinet for communicating metadata service and clients. The reason for using myrinet for COFS instead of the 1Gb Ethernet used by GPFS was to compensate for a network topology favouring the official file system servers. The forty GPFS servers had dedicated 1 Gb links to the main switch; on the contrary the COFS metadata server run on a standard computation node sharing 1 Gb link with other thirteen blades; as the tests were run while the whole cluster was in production executing other applications, we noticed that the results varied significantly depending on the blade being allocated for the metadata server and the activity of its neighbouring blades. On the contrary, the different topology of the myrinet network mostly prevented this variability. Though the myrinet protocols are theoretically able to provide less latency and greater bandwidth than the Gb network, using the IP stack on top of it adds enough overhead to compensate the differences: tests performed with the cluster in dedicated mode revealed that there were no substantial differences between the performance of the Gb network and the IP over myrinet network regarding metadata traffic.

Figure 3 shows the benefits of breaking the relationship between the virtual name space offered by COFS (exporting a single shared directory to the application level) and the actual layout of files in the underlying GPFS file system. By redistributing the entries into smaller low level directories, COFS allows GPFS to fully exploit its parallel capacity by converting a shared parallel workload into multiple local sections that do not require global synchronization.

The improvement translates into a reduction of the average *create* time from more than 10 ms in 16 nodes for GPFS to about 1 ms when using COFS over GPFS. The numbers for pure GPFS were limited to 64 nodes because beyond that point the system was suffering severe performance problems when running the benchmarks (that issue was not present when using COFS over GPFS).

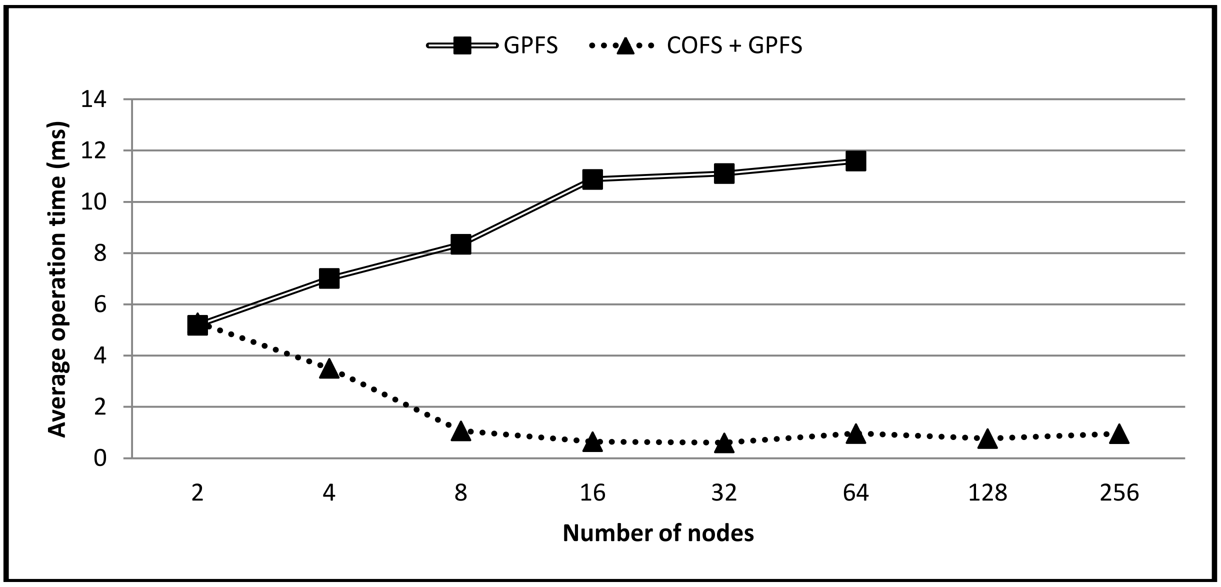

Figure 3 Parallel creation time improvements with COFS (1024 files per node)

Figure 4 shows the average time for *utime* requests. We can see that, in this case, the overhead introduced by COFS virtualization is noticeable for a small number of nodes, but it converges with pure GPFS results at 16 nodes. Beyond that point, we can observe that GPFS rapidly degrades its performance as we increase the number of participating nodes; on the contrary, COFS is able to compensate and eliminates the degradation, allowing the system to scale better to larger number of nodes.

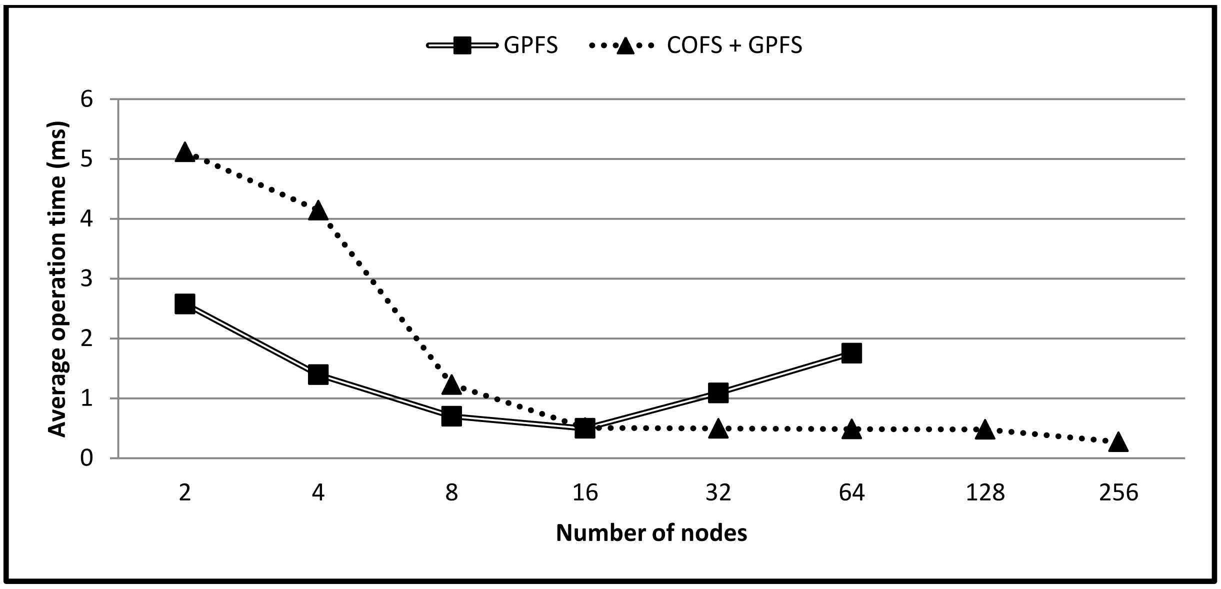

Figure 4 *Utime* request scalability (1024 files per node)

COFS takes advantage of the reduced operation time when GPFS delegation is active by redistributing the files in smaller directories and avoiding parallel accesses to such directories when possible, thus reducing the conflicts between different nodes and allowing GPFS to fully exploit the parallelism.

In summary, our measurements show that the virtualization of the name space provided by the COFS framework can drastically boost the base GPFS file system for file creations on shared parallel environments (with speed-up factors up to 10, as shown in Figure 3. For the rest of metadata operations, performance is also boosted for large numbers of nodes, and performance degradation due to conflicting parallel accesses is reduced.

Impact on data transfer bandwidth

After verifying that the benefits obtained by our prototype regarding metadata handling are promising, and that it effectively mitigates the issues motivating the present work, we also wanted to measure the possible impact of the virtualization environment on read/write operations on file contents.

Altering the file hierarchy could lead the underlying file system to modify the actual location of data, impacting negatively on read/write bandwidth; additionally, we wanted to be sure that COFS infrastructure was not adding an unacceptable overhead to data transfer operations. Possible causes would be FUSE's double buffer copying, round-trips to the metadata service or caching issues.

We have used the IOR (Interleaved Or Random [11]) benchmark to measure data I/O performance for GPFS with and without the COFS virtual layer. Even if COFS does not deal with data I/O, we wanted to verify that the hierarchy re-organization had no negative impact in this aspect. IOR v2 was developed at LLNL and provides aggregate I/O data rates for both parallel and sequential read/write operations to shared and separate files in a parallel file system. The benchmark was executed using the POSIX interface with aggregate data sizes of 256MB, 1GB and 4GB (the individual file size when using separate files is the aggregated data size divided into the number of participating processes.

Table 1 summarizes the results obtained with the IOR I/O benchmark in a small GPFS cluster. Overall, COFS over GPFS is usually able to obtain a data transfer performance similar to native GPFS. By ‘comparable performance’ we mean that the differences in performance between COFS and GPFS are not substantially bigger than the ‘noise’ observed during normal GPFS usage while the cluster is in production. The magnitude of this ‘noise’ varies greatly depending on factors such as the size of the data, the access pattern, the number of nodes and threads being used, the elapsed execution time of the benchmark and the distribution of the allocated nodes in the cluster; for this reason, and to summarize the data, the table presents qualitative results instead of quantitative data.

Access Pattern	**Separate files per process**	**Single shared file**
Sequential read	COFS performance comparable to GPFS except for small files (< 32MB per node) where COFS suffers an important slowdown.	COFS performance comparable to GPFS.
Random read	COFS performance comparable to GPFS except for small files (<32MB per node) where COFS suffers an important slowdown.	COFS performance comparable to GPFS.
Sequential write	COFS performance drawback for single node and performance improvements of COFS over GPFS as the number of nodes is increased.	COFS performance drawback for single node and comparable performance for multiple nodes.
Random write	COFS performance comparable to GPFS except for small files (<32MB per node) where COFS suffers from slight slowdown.	COFS performance comparable to GPFS.

Table 1 Impact of COFS on data transfers

The only remarkable performance drawback of COFS (with slowdown factors between 2 and 4) occurs when each node access independent small files. For operations on small separate files (less than 32MB,) pure GPFS is able to exploit its optimizations and the cache by locally keeping both the metadata and the file contents for read operations (files were created and written in the same node they were accessed.) Additionally, the total benchmark time for such small files are about a few milliseconds, which is comparable, for example, to the extra round-trips needed by COFS to access its metadata server. In these circumstances, COFS is paying the cost of its infrastructure. The case of writes is slightly different: not being a pure local cache operation (as data has to be eventually sent to file servers) GPFS cannot apply all of its optimizations; consequently, COFS benefits have room to partially mask the infrastructure costs, resulting in only slightly lower performance. The performance penalties disappear with larger file sizes, as transfer times become dominant compared with COFS infrastructure costs.

Noticeably, we also observed a positive effect of COFS when writing sequentially to separate files. In this case, GPFS bandwidth suffers degradation as the number of participating nodes was increased, while COFS was able to neutralize this effect. A closer look revealed that, for a larger number of nodes, the increased cost of the parallel open operation was "serializing" the data transfers (as the last node was able to open the file only much later than the first one, it also started to transfer data later); as a result, the use of the available data bandwidth was reduced. On the contrary, COFS reduced the open time to a minimum, allowing all nodes to start transferring data in parallel and achieving a much better use of the network bandwidth.

In summary, we did not observe a remarkable global impact of the COFS virtualization layer on the data transfer rates. The isolated performance drops affect only the GPFS highly optimized cases (local accesses to independent small files) where there is little room for improvement. Even then, the nature of the cases would make it possible to reduce the differences by incorporating the same aggressive caching and delegation techniques for strictly local accesses to the COFS framework.

4. Metadata Performance in Lustre

The lessons learned during the evaluation of GPFS metadata performance, and the deployment of the COFS framework on top of it were a valuable guide for the assessment of the metadata performance of the Lustre prototype deployed in PRACE test-bed at CEA.

Lustre [9] is a parallel file system able to offer a POSIX compliant interface and based on three types of components: the clients (nodes accessing the file system), the storage servers where the data resides (based on object storage devices) and a metadata server responsible for name space, access rights and consistency management.

One of the key characteristics of Lustre regarding metadata management is that it relies on a single metadata server (possibly replicated for failover replacement) to handle all metadata. This approach simplifies consistency management (compared to a fully distributed locking mechanism for metadata management – as in GPFS).

It is worth mentioning that the COFS infrastructure uses the same approach (dedicated metadata servers) to mitigate the metadata access contention issues detected in GPFS. As a consequence, it is expected that some of such issues resolved by using COFS were already mitigated in the native Lustre file system.

The goal of the evaluation was, in one side, to check if the potential scalability issues in Lustre were analogous to those found in GPFS, or if they were of different nature. On the other side, we wanted to verify if the COFS framework could be used to mitigate the potential issues, or if alternative techniques were needed.

It is important to mention that the specifications of the test bench at CEA changed during the evaluation of Lustre. The prototype was upgraded during December 2012 and January 2013, with overall improvements in the performance of the system. The measurements presented in this document were made with the latest version of the prototype (unless explicitly stated otherwise); as a consequence, some numbers may differ from the previous measurements presented in PRACE-2IP Deliverable 12.4 [12].

For the sake of reference, the original test bench was deployed in the INTI cluster at CEA, consisting of 128 Bullx Inca Nehalem-based blade nodes (with 8 X5560 processors at 2.80 GHz). The InfiniBand network topology was a fat-tree with 9 Mellanox IB QDR top-switches (with 36 ports each). The storage system (Xyratex Exascale I/O prototype) was a ClusterStor Neo 3000, with one node dedicated to the Lustre MGS (ManaGement Service), one node acting as MDS/MDT (MetaData Service/Target) and nine SSUs (Scalable Storage Units) containing 18 embedded Lustre OSS (Object Storage Servers). The SSUs had an IB QDR link each and were connected to the 9 INTI top-switches. The file system was a Xyratex version of Lustre 2.1.

During the upgrade of the test bench, the storage system was replaced by a ClusterStor 6000, including 9 SSUs with 18 CS6000 controllers (SandyBridge-based, FDR capable), a Cluster Management Node based on a quad-redundant node with shared, high performance MDT storage, 2 Mellanox QDR switches (36 ports each) and 1 GbE Network switch (24 ports). The Lustre file system was also upgraded to the latest Xyratex 2.1 prototype branch.

The following subsections summarize the observations of metadata behaviour in the Lustre prototype and the results of applying the COFS infrastructure on top of it.

4.1.Metadata behaviour in Lustre

The Metarates benchmark has been executed in the Lustre prototype in order to compare the results with the observations in GPFS. One of the goals is, given the different strategies used by both systems, to be able to obtain thorough information about the causes of metadata performance issues and which are the best ways to neutralize them.

The first step in the study of metadata behaviour in Lustre was measuring the file system performance on a single node, with the goal of spotting elements affecting the performance but unrelated to parallel access patterns. In particular, there was an interest in assessing the impact of the number of files on a single directory (which notably affected GPFS).

Figure 5 shows the average time spent in a file creation in a Lustre client for increasing sizes of the working directory. There are two important things to observe. First, the pattern of the lines is essentially flat (except when using 8 processors in directories smaller than 256 files, where the ratio between overhead and actual work is higher), indicating that the size of a directory is not, per se, an important factor regarding performance. The second is that using multiple threads inside a node effectively increases the performance.

The same behaviour can be observed for the *utime* system call (Figure 6), and the results for the *stat* system call are very similar (though slightly faster, there are no differences depending on the number of files in the directory).

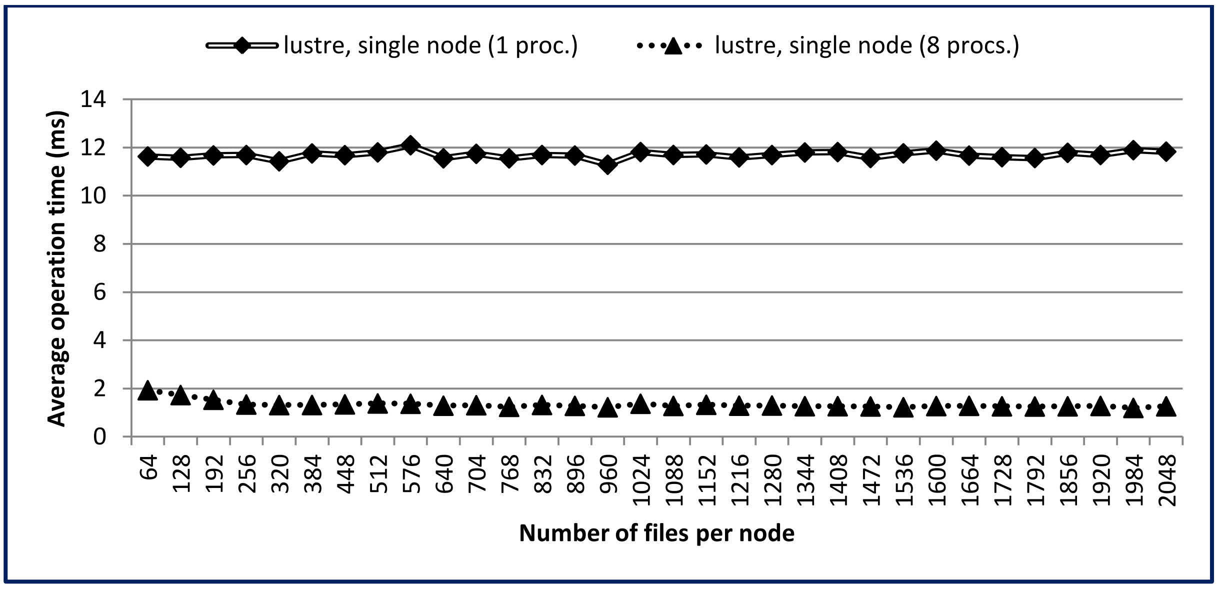

Figure 5 *Create* cost on a single Lustre node

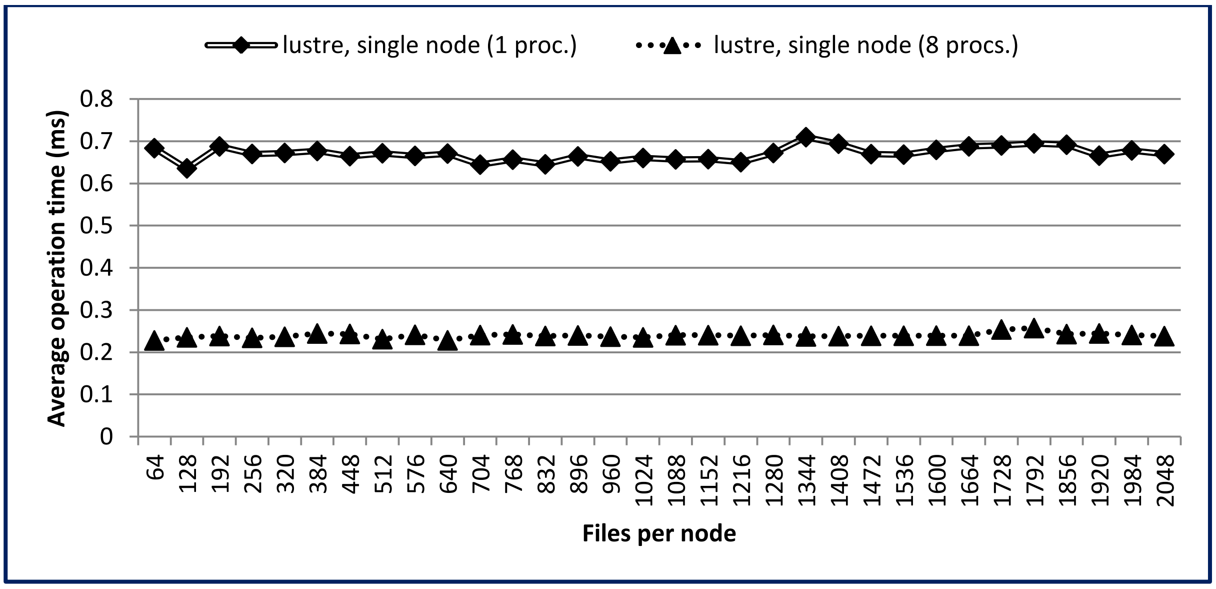

Figure 6 *Utime* cost on a single Lustre node

It is worth mentioning that this behaviour differs significantly from the observations done in the GPFS file system, where small directories used from a single node could benefit from special optimizations, resulting in a highly reduced operation time compared to larger directories (Figure 2).

The next step in the study of Lustre metadata behaviour is to observe the influence of having multiple client nodes using the file system at the same time. We compare the results in a single node with the results of executing the benchmark on 2 and 4 nodes. We have compared results for the same amount of files per node, in order to avoid the effects of different work versus overhead ratios in each node.

The results obtaining using 1 and 8 processes per node are shown in Figure 7. For this experiment, all files were created in a single shared directory. The first notable observation is that, overall, Lustre behaviour is, apparently, able to efficiently exploit internal parallelism, and the Lustre metadata server does not seem to behave too differently if parallel requests come from the same node or from different nodes (though increasing the number of nodes seems to be slightly more efficient than simply increasing the number of threads in a node – for example, the performance of using 4 nodes with a single process is not far from using 8 processes in a single node).

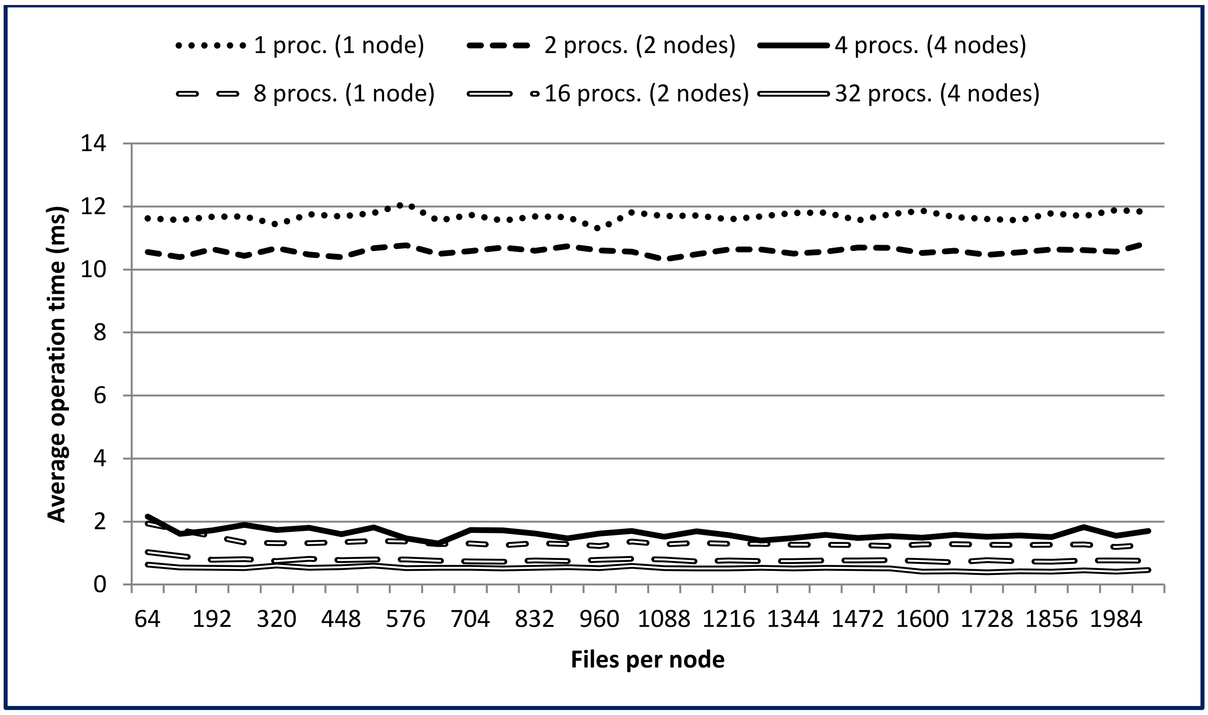

Figure 7 Lustre file creation cost on a shared directory

Nevertheless, it is worth mentioning the lack of improvement in performance when moving from one to two processes: Lustre seems to be able to effectively exploit parallelism only beyond 4 processes. A possible explanation for this effect would be that the overhead of maintaining the consistency when multiple clients use the file system is relatively high, and can only be compensated when there is enough parallelism to be exploited

The same experiments were executed creating the files in separate directories for each process, to avoid intra-directory conflicts. The results obtained showed the same pattern as Figure 7, with just slightly better performance for 8 or more processes.

Other metadata operations such as *utime* and *stat* show a different pattern and reveal a potential scalability issue. Figure 8 shows the average cost of the *utime* system call using up to 4 nodes with 1and 8 processes per node.

The first thing to note is that we do not see the lack of performance increase when moving from 1 to 2 processes (which we could observe for file creation in Figure 7). That means that the overhead in *create* is probably due to either the directory entry creation or the actual data file creation itself (tasks which are exclusively related to the file creation operation), rather than accessing or modifying a file's metadata.

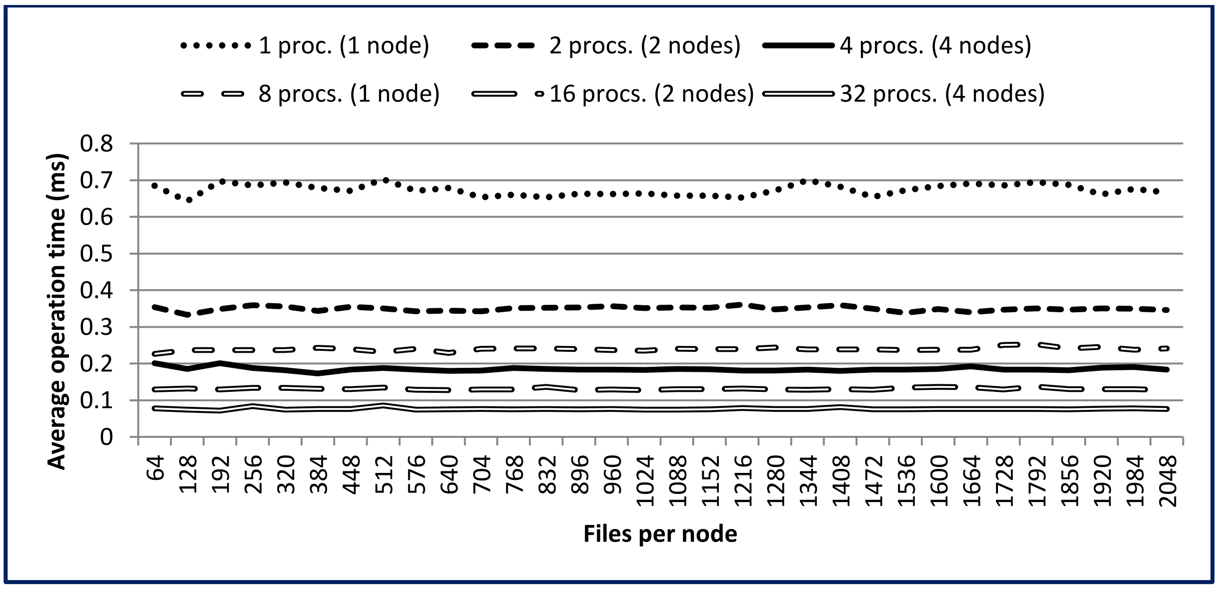

Figure 8 Lustre *utime* costs on a shared directory

The second interesting observation reveals a potential scalability issue. Indeed, we can observe that executing the benchmark with 4 processes offers a noticeably better average performance than using 8 processes. This effect is even bigger for the *stat* system call, where the system obtains an average cost of 0.12 milliseconds when using 4 processes, and then the performance drops to an average cost of 0.21 milliseconds when using 8 processes.

After doing additional measurements, we determined that the anomaly was mainly due to two different causes. First, *utime* and *stat* seem to benefit more from multi-node parallelization than from single-node parallelization; second, parallel access to a shared directory generates contention, which tends to increase with the number of processes accessing the shared resource.

For *utime*, the average cost of the system call when using 8 processes in a single node is about 0.24 milliseconds, and then it drops to 0.13 milliseconds when using 2 nodes with 4 processes each, and finally reaches 0.1 milliseconds for 8 nodes with a single process each. In the case of *stat*, the average costs of the system call for 8 processes are 0.21 ms, 0.11 ms and 0.06 ms when the processes are distributed into 1, 4 and 8 nodes respectively. A possible reason for this effect is that the actual cost of the operations is very small and, therefore, the impact of network aspects like latency and bandwidth (which are potentially improved by using more nodes) is more significant. This effect is not significant in the case of *create*, which has a cost more than an order of magnitude larger.

It is also expected that the effect of increasing the number of nodes versus intra-node parallelism will only be noticeable for a relatively small number of nodes, until either the network or the server is saturated. This point was confirmed with additional measurements showing that using additional threads in each node had no improvement in performance when using 32 nodes or more.

The other factor impacting the performance is the contention caused by parallel access to a shared directory. In order to assess this effect, we re-executed the benchmarks making each process work in a separate directory.

Figure 9 shows the reduction of the average cost of the *stat* operation when each process uses a unique separate directory to keep its files. The improvements in performance are also noticeable for *utime* and *create* operations, though smaller (about a 25% of cost reduction).

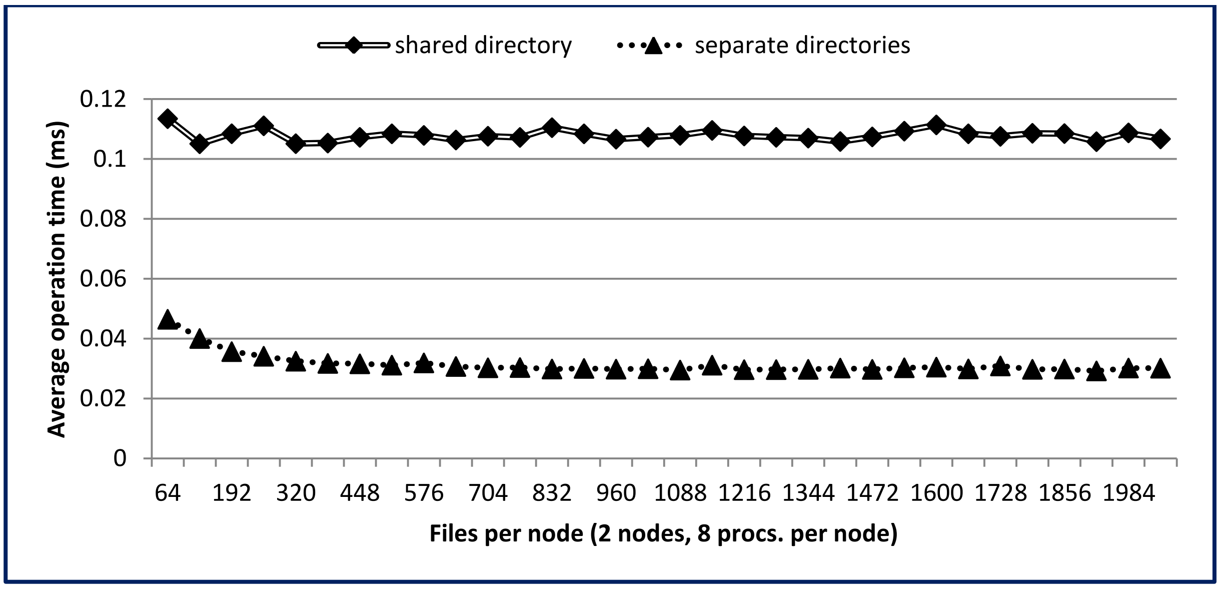

Figure 9 Impact of shared directory contention on *stat* operations

Differently from the effect of multi-node versus intra-node parallelism, the impact of contention during accesses to shared directories is expected to increase with the number of processes participating in the operation.

Figure 10 shows the impact in performance of creating files in a shared directory versus creating them in unique separate directories, and how it increases with the number of nodes involved.

On the contrary, the differences between using shared and non-shared directories tend to disappear for *utime* and *stat* operations, and the cost of the operations converge to a base line for 32 nodes and beyond (see Figure 11 for *stat* behaviour). A possible explanation compatible with this behaviour is that other costs (e.g. network-related costs) become dominant over the actual operation time, so that differences are no longer relevant.

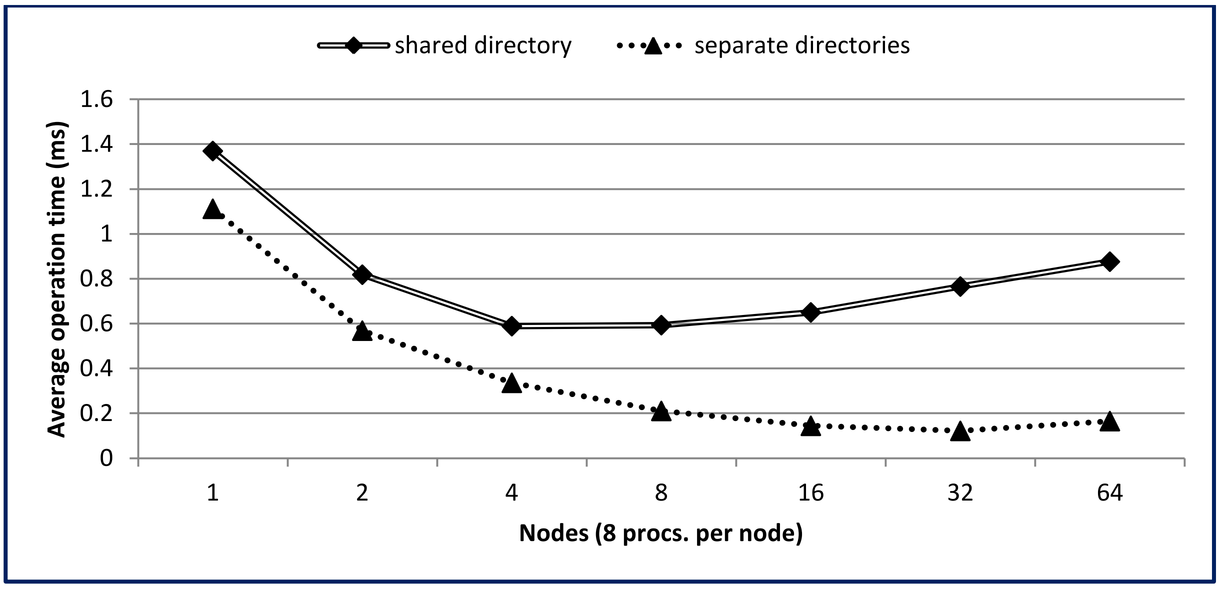

Figure 10 Impact of shared directory contention (1024 files per node) on *create* operations

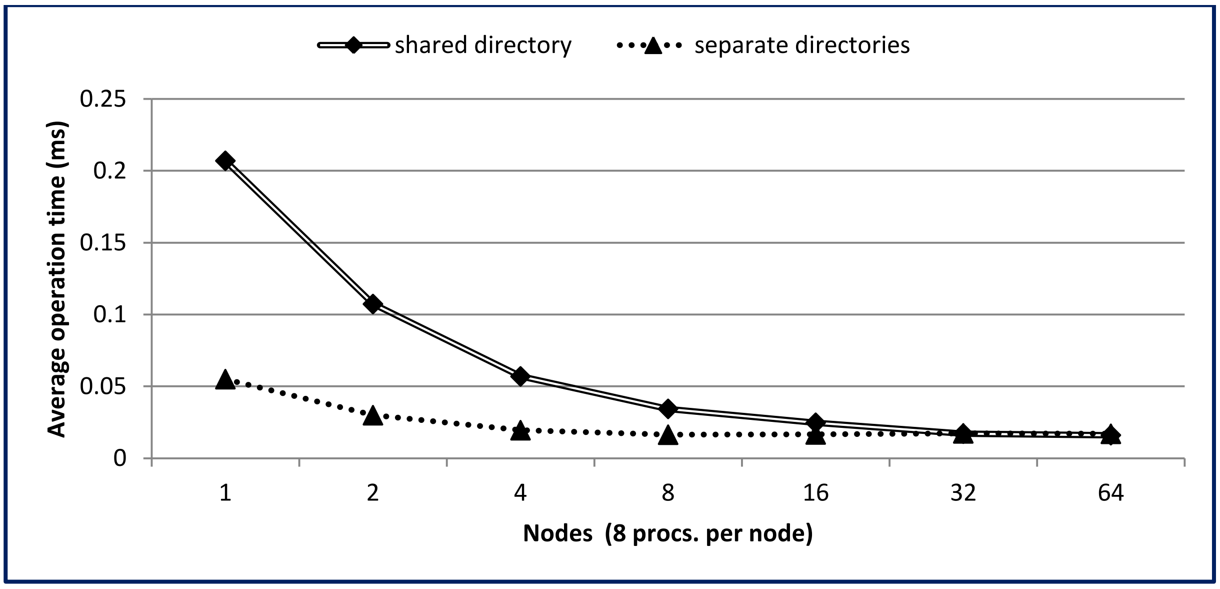

Figure 11 Impact of shared directory contention (1024 files per node) on large scale *stat* operations

There is a metadata-related operation which does not seem to be affected by parallel access to shared directory versus non-shared directories: open. We have not observed any noticeable differences when files are open by different processes in a single shared directory compared with opening the files in unique directories. This is probably due to the fact that other metadata operations (which do show differences) require some kind of synchronization when accessed in parallel (*utime* modifies the access times and must be kept consistent, *stat* requires synchronization of, at least, data sizes, and *create* has to keep the shared directory consistent); on the contrary, open does not require, per-se, to perform any type of synchronization (apart from the minimum required to ensure that the file being open exists and can be accessed).

Another important difference between open and the previously studied metadata operations is that, while the cost of other operations tends to stabilize as we increase the number of nodes, open shows a clear degradation when using more than a few nodes, which is aggravated when using multiple threads per node.

Figure 12 shows the loss of performance of the open operations when the number of nodes is increased. The figure shows the results for parallel open operations in a shared directory, but there are no noticeable differences in behaviour when operations are performed in separate directories dedicated to each process.

In summary, we have observed that parallel creations on shared directories suffer from overhead with respect to parallel creations in separate directories per process (though the overhead is less important than in GPFS).

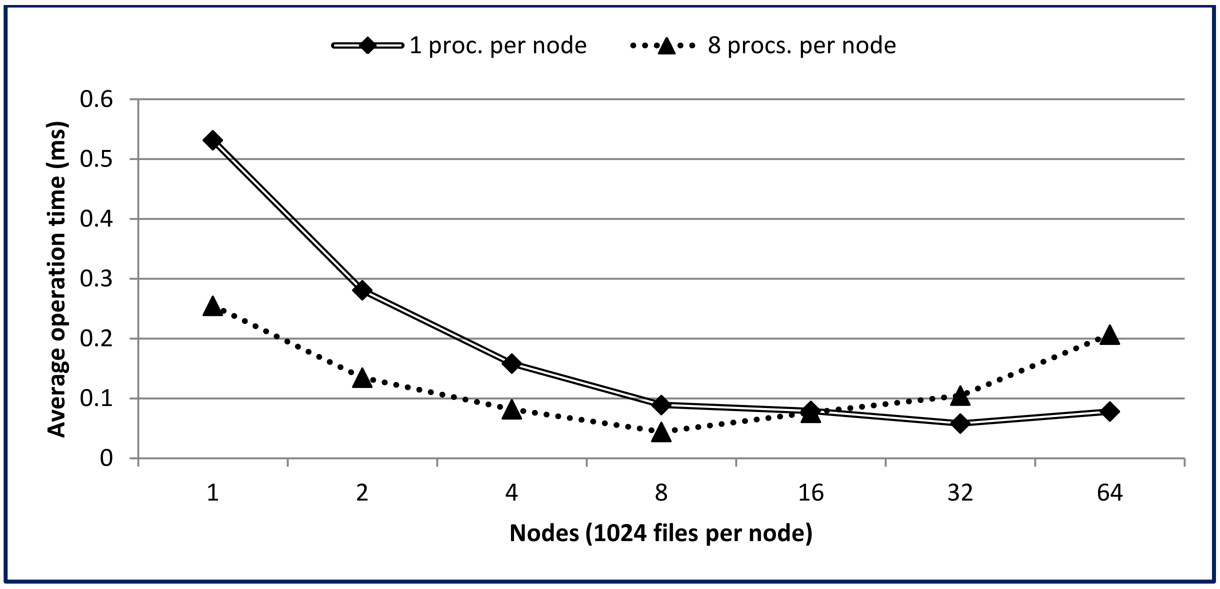

Figure 12 *Open* behaviour on shared directories in Lustre

Parallel operations in Lustre scale quite well up to an optimum value about 16-32 nodes (depending on the operation and conditions) but show hints of performance degradation beyond that point. The goal should be avoiding the performance degradation and even continue the performance improvement with more nodes.

Finally, the sequential behaviour of Lustre regarding metadata operations is not significantly affected by the number of files in the directory (also differing from GPFS) and it does not seem to have specific optimizations to exploit the sequential case (e.g. via control delegation to the client node).

4.2.Using COFS over Lustre

After studying the behaviour of metadata in Lustre, we deployed the COFS prototype (version 1.0.5) to verify that it could mitigate some of the metadata performance issues we observed, especially the loss of performance when creating files in a shared directory.

Being a drop-in component with only generic dependencies, no modifications were done on the prototype initially developed for GPFS, apart from usual rebuilding of binaries. At the CEA prototype, the COFS metadata service was run in a normal computation node of the cluster. Even if this implied some performance handicaps compared with Lustre metadata server setup, we expected to gather enough information to assess the feasibility of COFS mechanisms to improve certain performance aspects of Lustre.

COFS offers a file system namespace decoupled from the underlying file system (Lustre, in this case), and it internally reorganizes the location of the files without altering the user view of the directory hierarchy; specifically, COFS tries to place files in such a way that the underlying file system can obtain a good performance even if the original file distribution would cause parallel access conflicts and, consequently, performance degradation. An application working on top of COFS will send file system requests to COFS metadata service which, eventually, may forward them to the underlying Lustre file system. This involves an overhead with respect to a native Lustre operation, which can be compensated if the gains of the file distribution made by COFS are higher than the cost of the extra work performed by the COFS layer.

According to the results of the study of Lustre metadata behaviour (Section 4.1), one the candidate situations where COFS can mitigate Lustre's performance degradation is the parallel creation of files in a shared directory.

Figure 13 shows the results of using COFS over Lustre for parallel file creation. We can see that, for a small number of nodes, the overhead of COFS is higher than the benefits of file reorganization, but beyond 8 nodes, the gains compensate the overhead, preventing further degradation of the performance.

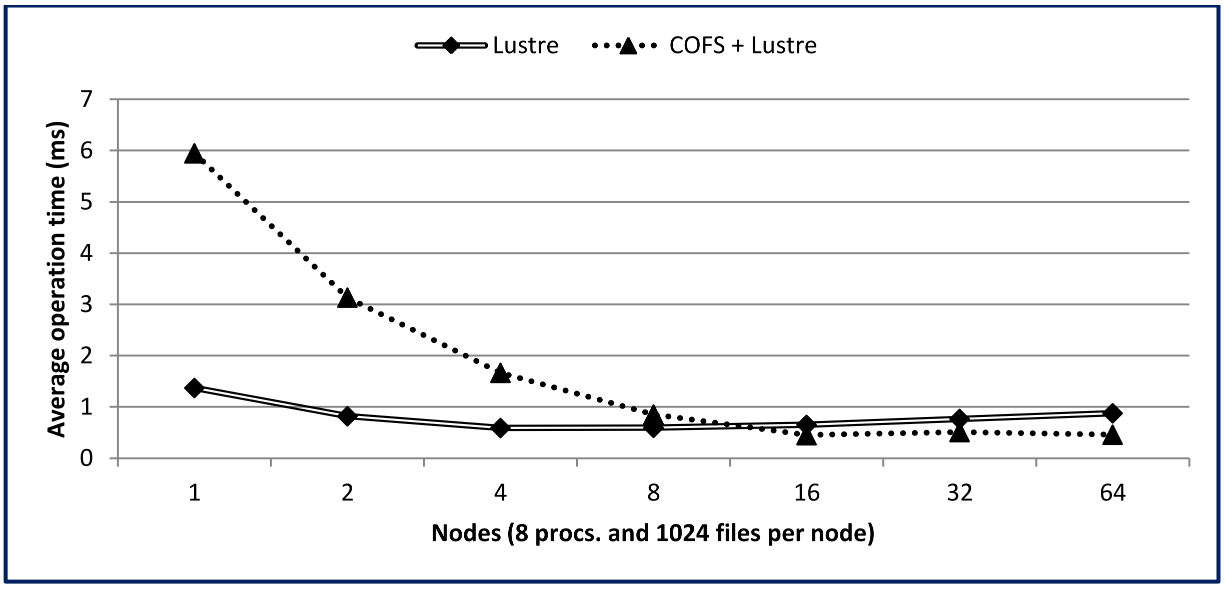

Figure 13 Parallel creation time using COFS over Lustre

Even if the benefits of COFS in the case of parallel creation are smaller than they were for GPFS, there is still an indication that the way COFS manages the directories is adequate for large-scale parallel access, and could be considered as a possible candidate to be integrated inside Lustre itself, in order to avoid parallel creation degradation without the need to pay the overhead of an external layer.

Another situation in which we have observed an improvement due to using COFS over Lustre is file opens. Again, native Lustre showed performance degradation when for large numbers of clients, though it was not related to accessing a shared directory, as it also appeared for non-shared directories.

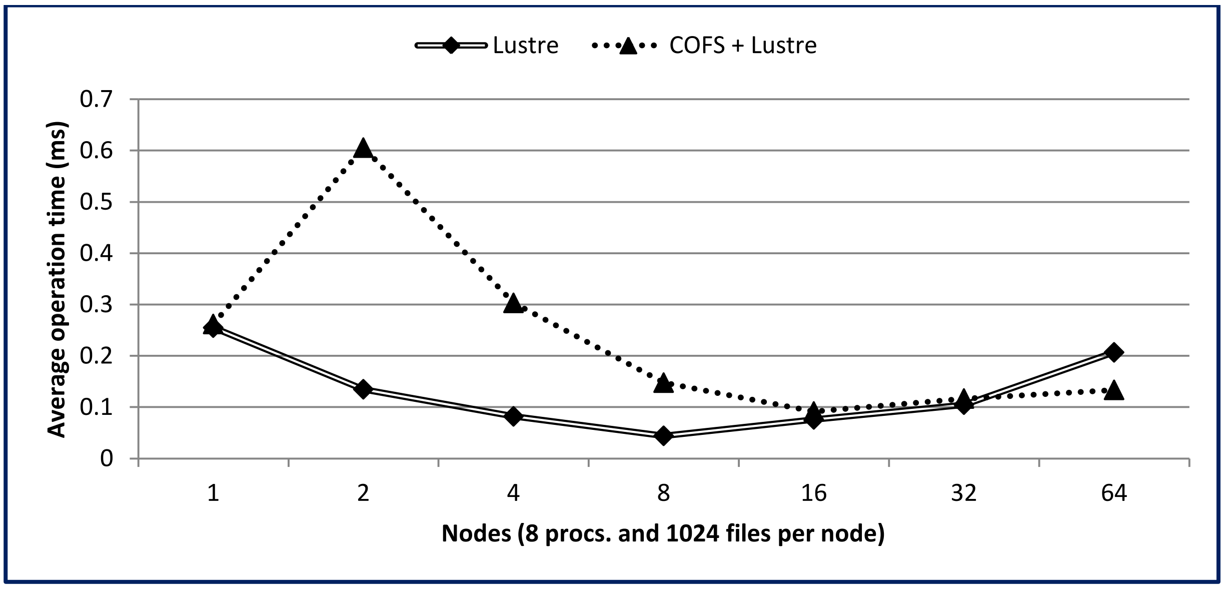

Figure 14 Parallel open time using COFS over Lustre

Figure 14 shows the open behaviour of COFS over Lustre. Again, the overhead introduced by COFS is higher than the benefits for small number of nodes, but it is compensated when the number of nodes increases, mitigating the Lustre degradation. The low overhead of COFS for the single-node case is due to some specific optimizations introduced to be competitive with GPFS control delegation mechanisms for isolated, single-node, file system operations.

The open case is interesting because it is less depending on the underlying file system metadata than other metadata operations like *utime* or *stat*, and it is almost completely handled at COFS level. This explains why COFS can show performance improvements, even when there was no obvious "optimal situation" observed in the study of Lustre.

The execution of the benchmarks for the remaining metadata operations (namely *utime* and *stat*) showed no further improvements derived from the use of COFS because either Lustre was already showing scalable good performance, or because the margins between optimal and non-optimal Lustre behaviour were too small to compensate for COFS overhead.

Nevertheless, the results also indicate that Lustre may obtain benefits from integrating some of the directory and management techniques used by COFS in order to improve situations with degraded performance, without paying the cost of an external layer.

4.3.Summary

From the experiments, we have learned some facts about file system behaviour that may help us to optimize the Lustre prototype and improve its scalability.

The sequential behaviour of Lustre regarding metadata operations is not significantly affected by the number of files in the directory (differing from GPFS). Only very small directories (less than 256 entries) appear to cause performance instabilities during parallel accesses.

Parallel creations on shared directories suffer from overhead with respect to parallel creations in separate directories per process. The overhead is much less significant in Lustre than in GPFS. One of the possible causes is that GPFS has a delegation mechanism that allows transferring the control of metadata operations from the server to the client temporarily, allowing near-local operation times when a node works on a directory which is uniquely accessed by a single node. On the contrary, Lustre does not have such a mechanism and operations to both unique and shared directories need to go to the remote metadata server (resulting in less significant improvements for non-shared operations).

Parallel operations in Lustre scale quite well up to an optimum value about 16-32 nodes (depending on the operation and conditions) and stabilize beyond that point. Notable exceptions are parallel file creations in shared directories and open operations in both shared and non-shared directories. The goal should be avoiding the performance degradation and even continue the performance improvement with more nodes.

Decoupling the name space from the low-level file system layout is the mechanism used by COFS to mitigate metadata issues. The COFS virtual layer offers large shared directory views while internally splitting them to take advantage of non-shared operations. Using this mechanism over Lustre resulted in moderate improvements for both parallel creations and open operations, preventing performance degradation for large numbers of nodes.

While the overhead introduced by COFS appears to be too high to be used effectively in the PRACE Lustre prototype at CEA, the techniques to handle directories and metadata seem to adequate for large scale systems, and they could be integrated inside Lustre to take advantage of the benefits without having to pay the cost introduced by an external virtualization layer.

5. Conclusions

This document presents evaluations to detect problems in metadata management for large-scale storage systems, as well as the exploration of techniques to mitigate metadata-related issues.

We have evaluated the effect the number of files and clients have on the performance of metadata operations in both Lustre and GPFS. In this evaluation we have detected that both the parameters have an impact on the obtained performance, which is especially important in parallel file creation.

GPFS metadata management is optimized for single-node accesses and suffers from severe performance penalties when shared access to directories is requested from a large number of nodes. The shared access overhead affects all metadata operations, but it is particularly important during file creation.

Lustre metadata management appears to scale better than GPFS’, in particular when accessing shared directories in parallel. Nevertheless, there is still certain performance degradation for parallel file creations. Also, scalability issues have been identified for the open operation.

We have investigated the feasibility of using COFS as a middleware to handle metadata. COFS is a middleware that decouples the view the user has from the one implemented by the file system and thus is able to transparently convert not optimized cases from the parallel file system point of view into optimized ones, thus getting all the benefits without asking the user to change their behaviours.

Using COFS on top of GPFS has allowed mitigating the scalability issues in a native GPFS system, improving file creation ratio and preventing performance degradation in other metadata-related operations.

The improvements of using COFS over Lustre are more moderate, but there are still improvements in the areas where Lustre suffered from performance penalties (file creation and open operations).

The information provided by the metadata performance measurements will help to tune the large scale PRACE systems (and high performance systems in general), as well as algorithms and specific applications, by showing which situations are likely to produce performance issues when accessing the storage systems and avoiding them. To this end, the current COFS implementation is available and can be used to mitigate certain metadata performance issues. The flexible architecture of COFS allows using it both as a transparent layer covering a whole file system, and as a tool to improve the behaviour of individual applications, without affecting the rest of the system.

In those situations where the overhead of the additional layer introduced by COFS is too high to obtain significant benefits, the directory and metadata management algorithms used by COFS could be integrated inside the target file system (either GPFS or Lustre) to obtain the benefits without the extra cost.

6. Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° RI-283493.

About PRACE

The Partnership for Advanced Computing in Europe (PRACE) is an international non-profit association with its seat in Brussels. The PRACE Research Infrastructure provides a persistent world-class high performance computing service for scientists and researchers from academia and industry in Europe. The PRACE computer systems and their operations are funded by the governments of the representative organizations hosting the systems. The Implementation Phase of PRACE receives funding from the EU's Seventh Framework Programme (FP7/2007-2013) under grant agreements RI-261557, RI-283493 and RI 312763. For more information, see www.prace-ri.eu.

7. References

[1] F. Schmuck and R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," in *FAST'02: Proc. of the 1st USENIX Conf. on File and Storage Technologies,* 2002.

[2] A. Devulapalli and P. Wyckoff, "File Creation Strategies in a Distributed Metadata File System," in *IPDPS '07: Proceedings of 21st IEEE International Parallel and Distributed Processing Symposium,* Long Beach, CA, USA, 2007.

[3] W. Yu, J. Vetter, R. S. Canon and S. Jiang, "Exploiting Lustre File Joining for Effective Collective IO," in *CCGRID '07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, 2007.

[4] "FUSE: Filesystem in Userspace," [Online]. Available: http://www.fuse.org.

[5] "Erlang/OTP (Open Telecom Platform)," [Online]. Available: http://www.erlang.org.

[6] B. Welch, M. Unagst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka and B. Zhou, "Scalable Performance of the Panasas Parallel File System," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies, FAST'08,* 2008.

[7] PVFS2 Development Team, 2003. [Online]. Available: http://www.pvfs.org/cvs/pvfs-2-7-branch-docs/doc/pvfs2-guide.pdf.

[8] J. R. Douceur and J. Howell, "Distributed Directory Service in the Farsite File System," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, Seattle, 2006.

[9] P. J. Braam, "Lustre File System: High-Performance Storage Architecture and Scalable Cluster File System (White Paper)," Sun Microsystems, Inc., 2007.

[10] "Metarates," University Corporation for Atmospheric Research (UCAR) and NCAR Scientific Computing Division, 2004. [Online]. Available: http://www.cisl.ucar.edu/css/software/metarates/.

[11] "IOR HPC Benchmark," 2011. [Online]. Available: http://sourceforge.net/projects/ior-sio/.

[12] E. Artiaga and A. Miranda, "D12.4 - Performance Optimized Lustre," 2012.

[13] J. Nelson, "Concurrent Caching," in *Proceedings of Erlang'06,* Portland, Oregon, USA, 2006.