Evaluating the performance of Lustre File System

Francieli Zanon Boito   Rodrigo Virote Kassick
Philippe O. A. Navaux
Universidade Federal do Rio Grande do Sul
Instituto de Informática
Porto Alegre - RS - Brasil
{francieli.zanon, rvkassick, navaux}@inf.ufrgs.br

Abstract

Applications that execute on cluster environments usually generate huge data loads. This data may be needed in an efficient manner by the computing nodes. In these cases, Distributed File Systems (DFS) appear as a natural solution. File systems may present different behaviours for different access patterns, depending on their projects. Therefore, it is important to study the performance behaviour of a DFS under access patterns verified in practice, so we can have tools to compare and measure the file systems. Besides, with this study, application may be adapted to take advantage of the DFS with changes on their I/O operations. In this work, we study the performance behavior of Lustre File System under common access patterns of scientific applications. To simulate the workloads, we propose and apply classes of tests. The obtained results helped the creation of a more detailed profile about Lustre's operation.

1. Introduction

With the disparity in the growth of the capacity of processors and I/O devices, and with High Performance Computing (HPC) applications generating huge data loads, the I/O became a crucial part for performance of the applications. The data usually shall be acessible by all the computing nodes. For providing this access, Distributed File Systems (DFS) are a natural solution.

Depending on their design, file systems can present different behaviours under different access patterns. Thereby, a simple benchmark can be not enough to decide between systems, as it simulates just one condition. To study the behaviour of performance of the systems under different access patterns is important, so we can have tools for a better comparison.

The applications can, having the knowledge of the DFS to be used, adapt their I/O operations for taking advantage

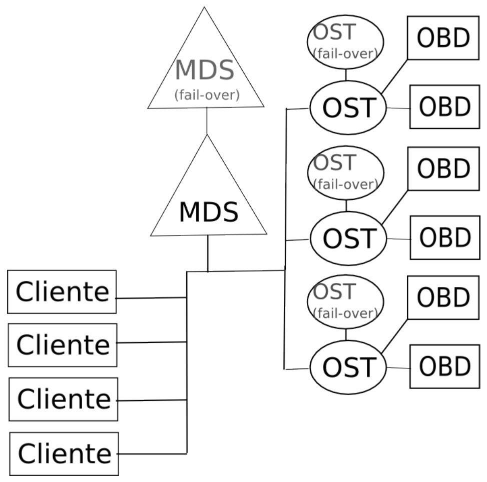

Figure 1. Lustre File System architecture.

of the system. In the same way, knowing the impact in performance of different project options, distributed file system designers can adapt the system for the applications that will use it.

In this paper, we present and apply classes of tests for simulate scientific applications workloads. Then, the obtained results are used to the creation of a profile about Lustre File System's behaviour.

The rest of the paper is organized as follows: the next section will present the Lustre File System. In Section 3, the method for simulating workloads will be introduced. Then the obtained results will be showed in Section 4, followed by conclusions and future work in Section 5.

2. Lustre File System

Lustre File System ([2], [8]) is a system that aims providing high performance and scalability, developed by Sun Microsystems. Its architecture, ilustred in Figure 1, is composed by a centralized metadata server (MDS) and several data servers (Object Storage Targets, OSTs). The MDS is responsible for all the operations on the file system name space, while the OSTs take care of the data accesses and of the interaction with the storage devices, the Object-Based Disks (OBDs).

Though the storage devices are called OBDs, they do not have to be disks. As their interface with the OSTs uses a device driver that hides their identity, any storage technology can be applied, like Linux file systems (ext3, ReiserFS, etc). For the communication, Lustre uses a Lustre Networking (LNET), which supports several network technologies with connectable drivers, called Lustre Network Drivers (LNDs).

The clients access the file service through a provided virtual file system layer. The data servers do not apply any cache. Nevertheless, it can be provided by the OBD. In the clients, there is cache with head-ahead.

3. Simulating practice-observed workloads

Lustre was choosed for this work due to their visibility and extensive researchs. Most of the top-30 supercomputers use this file system today. Besides, it was constructed for high performance, not being based on other systems. Therefore, Lustre tends to do not heir fragilities of aged file systems.

Aiming simulate scientific applications observed workloads, it is presented six classes of tests: SFWA, SFWA-SMP, SFSA, SFSA-S, MFWA and MFWA-S. They focus on two specifics characteristics that influence the performance: cache and data distribution. [5]

There can be a file per client, or only one to be accessed for all of them (SF, Single File, or MF, Multiple Files). All the data in the file can be accessed in each client, or they can operate in segments (WA, Whole Access, of SA, Segmented Access). Thereby, can be or not intersection between the data accessed for the nodes (S, Shared).

For **SFWA**, all the clients access for concurrently read all the data in a shared file. This pattern is observed in applications that need the read of a big file (with checkpoint of previous results) by all the nodes involved before the start of the computation, like MESSKit and NWChem [7]. The **SFWA-SMP** variation applies several processes in each node, exploring the client cache.

In **SFSA** and **SFSA-S** tests, each node accesses its own segments on a shared file. This allow us to observe the system behaviour without cache effects. In the shared variation, after write on its segment, each client accesses for read the part of a neighbor. This pattern is observed in applications like Flash [4].

MFWA and **MFWA-S** classes of tests simulate a situation where the clients operate in their own files. The situation is similar to the SFSA tests, but in a file scale. An example of application that does its I/O like this is sPPM [1].

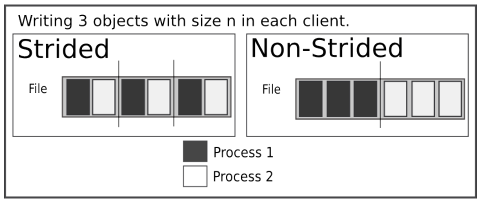

Figure 2. File segmentation with the options of MPI-IO Test with 3 objects with size n per client.

The execution of the tests was done using MPI-IO Test [6], an open-source tool that works in top of MPI-IO calls. The test is described for the arguments passed, like operation (read or write), files involved, barriers and segmentation of the file (strided or non-strided).

In a non-strided approach, all the clients have a portion of the file where they exclusively read and write into. In a strided scenario, all the nodes access a portion of the file, and then seek to the next portion. Figure 2 ilustrates the two situations. The number of segments and their size are also configurable.

3.1. Methodology

For the experiments, Lustre was used with 4 OSTs and at most 40 clients. The striping configuration was circular, starting at a random node, in segments of 64KB. In the OSTs, the storage environment is the local file system (ext3). The amount of data accessed in each client is 2GB.

The cluster utilized was Helios, from Grid 5000. It has AMD Opteron nodes, 4GB of RAM and a Gigabit Ethernet network. The results have a confidence of 90% and relative error of at most 10%. For all the classes, there are 2 major tests:

- Varying the number of clients: the segments and their size if fixed (32 objects of 64MB each, for the most of the tests), while the number of clientes involved grows from 1 to 40.
- Varying the number of segments: the number of clients is fixed in 40, while the number of segments grows. The size of each of them is adapted to maintain the amount of data accessed.

4. Results

This section presents the results obtained with the tests described above for some of the classes proposed.

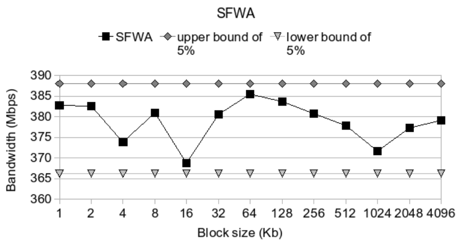

Figure 3. SFWA test varying the block size of the requests.

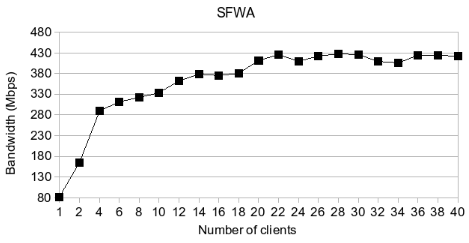

Figure 4. SFWA test varying the number of clients involved.

4.1. SFWA - Single File, Whole Access

The tests of this class do not use MPI-IO Test, but the dd Unix command. In the Figure 3 the results of the test varying the block size of the requests done are shown. There is no sensibility for this parameter.

In the Figure 4, we can see the results of the test varying the number of clients. The block size is 2KB. The performance grows until becoming stable, and then does not suffer degradation with the growth in the number of clients. The same can be observed for SFWA-SMP results in the Figure 5.

The reference of 4 times the results for SFWA is not alcanced for SFWA-SMP. However, the use of the cache in the clients provides a good performance (about 3.5 times the bandwidth of the single-processed version).

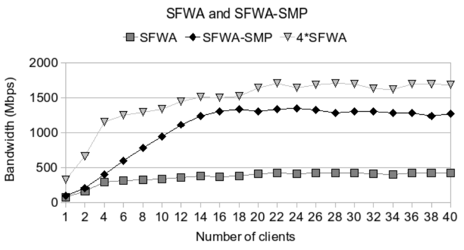

Figure 5. SFWA and SFWA-SMP tests varying the number of clients involved.

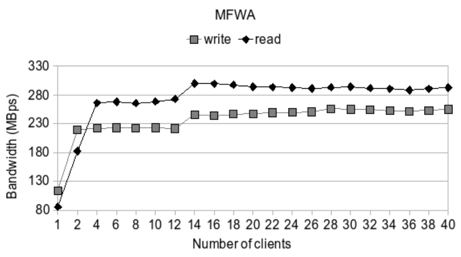

Figure 6. MFWA test varying the number of clients involved.

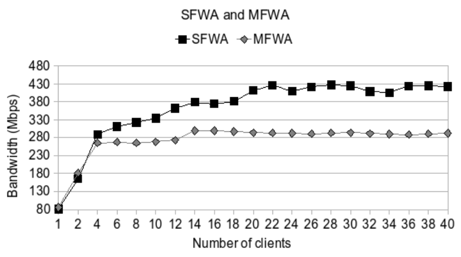

Figure 7. MFWA and SFWA read operations tests varying the number of clients involved.

4.2. MFWA - Multiple File, Whole Access

The results of the test of the class MFWA varying the number of clients involved is shown in Figure 6. We can see that, like in the previous tests, the performance grows until become stable. The variation observed in 12 clients is contained within the error tolerance. Figure 7 brings a comparison between the step that does read operations of this test and the test of the class SFWA. The last one has a better performance due to the cache in the servers, as the data accessed is the same for all the clients.

The test varying the number of segments passed as parameter for the MPI-IO Test tool is shown in the Figure 8. The striping configuration was modified for starting in the first server, and the results obtained were compared with the random-start approach. There is no sensibility for the number of segments (all the variations are within the error tolerance). We can see that the fixed-start situation has better performance in read operations.

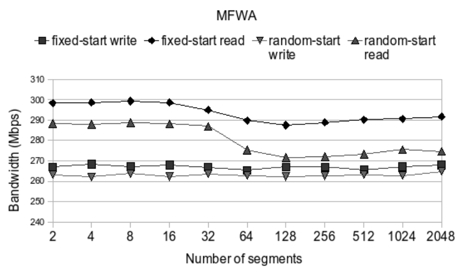

Figure 8. MFWA test varying the number of objects acessed in each client.

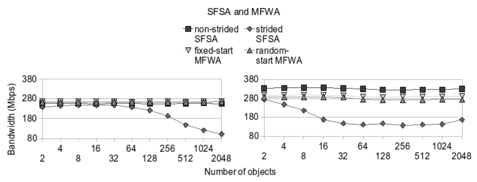

Figure 9. SFSA and MFWA tests varying the number of objects passed as parameter for the MPI-IO Test tool.

4.3. SFSA - Single File, Segmented Access

The results for the test varying the number of segments utilized is shown in Figure 9, with the options strided and non-strided and compared with the test of the class MFWA. The performance of the version non-strided does not change with the growth in the number of objects.

Having just one segment per process has, as we can see, better performance. Increasing the number of segments, the bandwidth decreases. It is more noticeable for the read operations, because of the head-ahead cache, which losts its functionality. The results for the read operations in SFSA non-strided test is better than in MFWA tests.

5. Conclusion and Future Work

Analysing and comparing the performance of a distributed file system just with the results of one benchmark is not enough to have a real idea of its behaviour under different access patterns. In order to provide tools for better comparisons and choices, this paper presented classes of tests that simulate the I/O operations patterns practice-observed in scientific applications. Results for some of the suggested tests were showed, and its results allow conclusions about Lustre's behaviour:

- Lustre scales well, not sufffering degradation with the growth in the number of clients acessing the service.
- The cache in the clients avoid repeated requests for the same data when applying several processes per node. The cache in the server can provide better performance too when there is intersection between the data acessed by the clients.
- Read operations have better performance than write operations.
- If a client will access a contiguous portion of data, the number of requests with it will be done does not affect the performance.
- The more granular a client do its operations in a file, the worst the performance.
- It's better when all the clients starts their requests in the same server.
- The single-file approach is better than the multiple-file even when there's no intersection between data accessed in the nodes.

As future work, more situations can be simuled, providing more informations about Lustre behaviour. The tests can be used for analysing other DFSs, allowing a comparison between them. The conclusions about the system can, still, be used for increasing the performance of an application by adapting their I/O operations for take better advantage of the file system being used.

References

[1] The asci sppm benchmark code, 1998.

[2] Lustre: A scalable, high-performance file system. Cluster File Systems Inc. white paper, version 1.0, November 2002. http://www.lustre.org/docs/whitepaper.pdf.

[3] T. M.-I. Committee. Mpi-io: A parallel file i/o interface for mpi version 0.5, 1996.

[4] B. Fryxell et al. Flash: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes. *Astrophys. J. Suppl.*, 131:273–334, 2000.

[5] R. V. Kassick, F. Z. Boito, and P. O. A. Navaux. Testing the performance of parallel file systems. In *Anais do VI Workshop de Processamento Paralelo e Distribuído*, Porto Alegre - RS, 2008.

[6] Mpi-io test user's guide. Los Alamos National Laboratory, 2008.

[7] E. Smirni and D. A. Reed. Lessons from characterizing input/output bahavior of parallel scientific applications, 1998.

[8] Lustre networking: High-performance features and flexible support for a wide array of networks. Sun Microsystems white paper, version 1.0, November 2008.

[9] T. D. Thanh, S. Mohan, E. Choi, S. Kim, and P. Kim. A taxonomy and survey on distributed file systems. *Networked Computing and Advanced Information Management, International Conference on*, 1:144–149, 2008.