Improvement of a Parallel File System Simulator by Data Striping Implementation

Vivien Michel
Polytech'Grenoble
Saint-Martin-d'Hères, France
Vivien.Michel@e.ujf-grenoble.fr

Francieli Zanon Boito
Instituto de Informática
Universidade Federal do Rio Grande do Sul, Brazil
fzboito@inf.ufrgs.br

Mario A. R. Dantas
Departamento de Informática e Estatística
Universidade Federal de Santa Catarina, Brazil
mario.dantas@ufsc.br

Abstract

In this paper, we discuss modifications to the PFSSim parallel file system simulator aiming to improve its accuracy. Simulation is an important tool for research as it allows for experimentation in situations where a real system is not available. However, accuracy is essential for simulators' usefulness. PFSSim has an accuracy problem due to not implementing data striping. We have implemented this operation and evaluated the new version of PFSSim against a real system. Our results show good results with the file-per-process approach and small requests, but also highlight the importance of adequate disk access simulation.

1. Introduction

High performance computing (HPC) applications rely on parallel file systems (PFS) to achieve good performance when handling large amounts of data. These systems distribute files' data - though an operation called *data striping* - among multiple data servers, which can be accessed in parallel for performance. Because of the historic gap between processing and data access speeds, there is an active research field aiming to improve parallel I/O techniques.

Research on parallel file systems requires experiments on large scale architectures in order to understand how the interaction between the different levels of the I/O stack affects performance behavior. Such experiments are also needed to validate new techniques, test predictive models' accuracy, etc. Nonetheless, researchers often do not have access to supercomputers or are not allowed to allocate the machine's time for their experiments. Even when access is possible, the experiments may require deep modifications in the system, which may not be allowed. In this context, a simulator is an important tool for research. Through a simulator, researchers are able to run needed experiments and explore deep changes that would not be possible in a real system. Moreover, using a simulator saves money in supercomputer time and energy consumption.

Simulation is an important and powerful tool as long as results are accurate. In other words, for a simulator to be useful, it needs to correctly represent the performance and behavior of a real system. In this paper, we discuss modifications we have implemented in the PFSSim [3, 4] parallel file system simulator for correctly simulating data striping and its effects. These modifications were performed aiming to improve the simulator's accuracy, specially for experiments with I/O scheduling algorithms.

The rest of this paper is organized as follows: the next section briefly presents the PFSSim simulator. Section 3 discusses the data striping implementation. Results obtained with both versions of the simulator (before and after our modifications), compared with results obtained from a real system, are discussed in Section 4. Section 5 brings final remarks and future work.

2. The PFSSim simulator

PFSSim is a parallel file system simulator built over the Omnet++ [8] network simulator. The file system components (client, data server, metadata server, and I/O gateway) and local components (local file system, virtual file system layer, disk cache, and disk) were implemented through the Omnet++ infrastructure. When configuring a simulation, the user must model the network topology and connect the components through the NED description language.

The system behavior was meant to replicate the Parallel Virtual File System (PVFS) [6] version 2.8.2, but without its implementation specifics. Therefore, the authors claim other file systems, such as Lustre [1], can also be simulated by adjusting some parameters.

An interface to describe I/O scheduling algorithms - to work in the context of requests to PFS servers - is offered by the simulator through the I/O gateway component (called "proxy"), which can be placed at client-side or server-side (by connecting it to the client or server component through a perfect channel). There is also the possibility of placing these components as intermediate machines between clients and servers, replicating an I/O forwarding organization. There is a communication channel between proxies that can be used to coordinate I/O scheduling instances.

Other parallel file system simulators [5, 2] have been proposed in the literature. However, we have chosen to work with PFSSim due to its capabilities for I/O scheduling algorithms simulation.

A weakness of PFSSim is that it does not implement the data striping operation, where a file is striped into fixed size portions and these portions are distributed among the data servers following a round-robin approach. In its original implementation, each data server receives a "disk image" where the whole file is present.

Since I/O scheduling algorithms work to adjust the access pattern at the servers, the access to the storage device is an important factor in the resulting performance. This access time is directly affected by the offset distance between consecutive accesses. For these reasons, the simulator's accuracy is affected due to the disk image not reflecting what would be observed in a real system. The next section will describe the modifications we have done in the simulator in order to adequately implement data striping.

3. Modifications in the PFSSim simulator

The PFSSim implementation includes some scripts to generate input files required for the simulation. One of them is in charge of creating a file for each storage device. Before the modification, the size of the whole files were used to write this configuration file, and all servers received identical files. Next, the configuration file is read by the simulated local file system. In other words, the file system on each disk have the whole files inside. The improvement consists in the distribution of the total size of each file on each server based on the "stripe size". The algorithm can be splitted in three parts:

1. **Use the previously generated layout file**: the first step consists in parsing the layout file in order to know how files are striped. In fact, for each simulated file, the metadata server knows the stripe size and the number of the n_{th} server, since the first server (the one receiving the first portion of the file) is not necessarily server zero. With this parsing, the script takes all information needed to split the given amount of data among the servers. Additionally, the stripe size unit can be kilobytes or megabytes.
2. **Split data among the servers**: then data must be split among the servers accordingly to previously recovered information. Each server has its own total amount of data which is a subdivision of the total file size.
3. **Write data on per-server files**: finally, each file and server have their own amount of data (in bytes) to write in an individual "disk image" file. They will be read by the simulator at its initialization.

In order to verify if these changes were enough to correctly implement data striping, it was necessary to verify if offsets requested by clients were relative to the data servers' local files or to the global file stored in the PFS. Through a script that logs non-contiguous accesses to data servers, we have confirmed that offset were already correctly translated.

4. Performance Evaluation

We have conducted a performance evaluation in order to see how accurate simulation results are before and after our modifications. For that, we have compared the simulated results to those obtained from a real system. The next section describes the evaluation methodology, and Section 4.2 discusses the obtained results.

4.1. Methodology

The real system used in our experiments was the Graphene cluster from *Grid'5000*. Graphene's nodes have a quad-core Intel Xeon $X3440$ 2.53GHz, 16GB of RAM and a 320GB SATA II hard disk. The operating system used was Debian 6 (Squeeze) - kernel 2.6 - with MPICH2, PVFS version 2.8.2 (not the most recent version, but the same considered during the simulator's development).

We have used 1 node as a PVFS metadata server, 6 nodes as data servers and 32 as clients. The used stripe size was 64KB. Tests were executed on the clients through the IOR[1] benchmarking tool (version 3.0.1), which allows access patterns description. We have executed tests varying the following aspects:

- file-per-process approach or shared file (independent files have 2GB each, the shared file has 2GB per client);

1 https://github.com/chaos/ior

- request size: 32KB (smaller than the stripe size) or 1MB (larger than the stripe size times the number of servers);
- read or write tests;

The Graphene's Gigabit Ethernet interconnection was modeled in Omnet++ and its nodes' parameters (such as page size, dirty_ratio, etc) were obtained to configure the simulations. In the simulator, proxy nodes were placed with the data servers (one proxy per server) and we have experimented with the EDF, SFQ and FIFO scheduling algorithms. Nonetheless, initial tests have not presented significant differences between results with different scheduling algorithms, so we have proceeded with the tests using the EDF algorithm only. The LMBench [7] benchmark was used to measure the time needed to read data from the main memory - the buffer cache - in Graphene's nodes and we have used this value to configure the simulator.

The presented results correspond to the arithmetic average of 5 repetitions in order to reduce the influence of aleatory aspect.

4.2. Results

All tested combinations between the parameters listed in the previous section added up to 14 experiments. Due to space constraints, we have chosen to show here 3 of these tests we consider to be representative of the observed behaviors.

Figure 1 presents these three sets of results. All of them are for read tests. Figures 1a and 1b show results obtained with the file-per-process approach with different request sizes - 1MB and 32KB. The results from tests where all processes share a file (with 1MB requests) are presented in Figure 1c. Simulation results are represented by the first (before the data striping implementation) and second (after the data striping implementation) bars. The third bars represent the results observed in the real system. The y axis gives the total I/O time in seconds. The times to open and close the file are not included in the results.

For the tests where each client has an independent file, the correct data striping results in shorter I/O times. This happens because files are stored sequentially in the disk image. When an access to file B is done right after an access to file A, the offset distance used to obtain the disk access time depends on A's size (and size of all files stored between A and B, if this is the case). With data striping, each file is smaller on each data server's storage device (because each data server has only a part of this file) and hence offset distances are smaller. PFSSim considers disk access time to be linearly proportional to the offset distance, so smaller offset distances result in smaller disk access times, decreasing the simulated time.

Our best results - where the new version of the simulator accurately represents the real system - were observed with the file-per-process approach with small requests (Figure 1b). As the request size increases (Figure 1a), the observed error increases. We believe this is due to PFSSim's disk access time modeling. Results obtained by UFRGS' researchers working in the SeRRa project[2] suggest that considering access time to be linearly proportional to the offset distance **does not** reflect real disk behaviors for large requests.

On the other hand, in tests with the shared file approach (Figure 1c), there were no significant difference between times obtained by both versions of the simulator. This happened because most accesses were served by the buffer cache - in main memory - so disk access and data striping played no role. At each server, after the first access to the shared file, **prefetching** is able to load data to serve future requests to the main memory. We can see that PFSSim underestimates I/O time in this situation. We believe this happens because the simulated prefetching mechanism is unrealistically successful in masking disk access time. This is supported by the results obtained for the real system: in all three tests the same total amount of data is accessed from the file system, but I/O time in Figure 1c is the highest. If all accesses were served by the buffer cache, it should be the lowest.

5. Conclusions

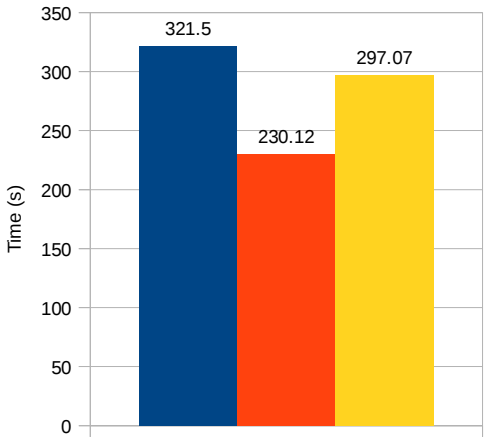
This paper discussed an improvement to a parallel file system simulator called PFSSim. It was chosen due to its I/O scheduling simulation capabilities, which are in the research group's interests. We have implemented the data striping operation aiming to improve disk access simulation.

We have evaluated the two versions of the simulator against a real system. Good results have been observed for the file-per-process approach with small requests. Nonetheless, other tested situations presented poor results. Ongoing work focuses on improving disk access simulation by using models provided by the SeRRa devices profiling tool.

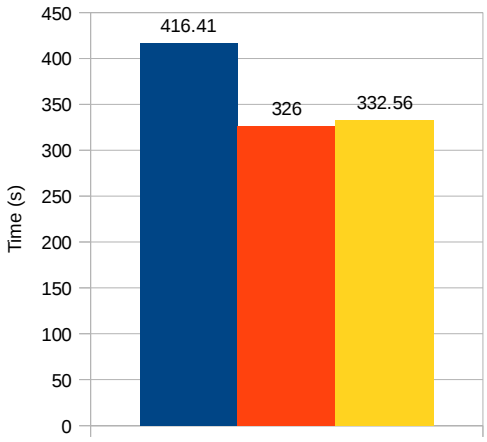
6. Acknowledgements

The experiments presented in this paper were carried out on the Grid'5000 experimental test bed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

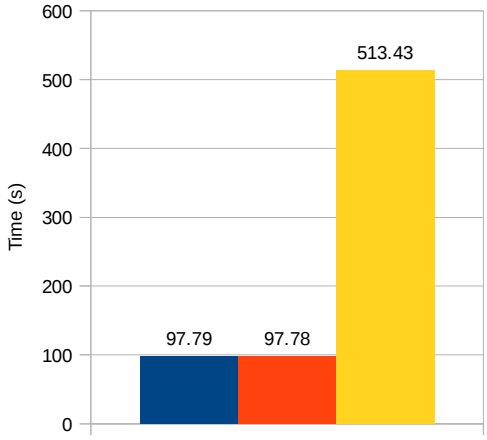
2 http://serratool.bitbucket.org



(a) File-per-process - 1MB requests



(b) File-per-process - 32KB requests



(c) Shared file - 1MB requests

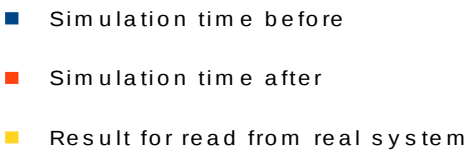


Figure 1: Results comparing two versions of the simulator with a real system

References

[1] Lustre: A scalable, high-performance file system, November 2002. http://www.lustre.org/docs/whitepaper.pdf.

[2] P. H. Carns, B. W. Settlemyer, and W. B. Ligon, III. Using server-to-server communication in parallel file systems to simplify consistency and improve performance. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, pages 6:1–6:8, Piscataway, NJ, USA, 2008. IEEE Press.

[3] Y. Liu, R. Figueiredo, D. Clavijo, Y. Xu, and M. Zhao. Towards simulation of parallel file system scheduling algorithms with pfssim. In *Proceedings of the 7th IEEE International Workshop on Storage Network Architectures and Parallel I/O*, 2011.

[4] Y. Liu, R. Figueiredo, Y. Xu, and M. Zhao. On the design and implementation of a simulator for parallel file system research. In *Mass Storage Systems and Technologies (MSST), IEEE 29th Symposium on*, pages 1–5, May 2013.

[5] E. Molina-Estolano, C. Maltzahn, J. Bent, and S. Brandt. Building a parallel file system simulator. In *Journal of Physics: Conference Series*, volume 180, page 012050. IOP Publishing, 2009.

[6] R. B. Ross, R. Thakur, et al. Pvfs: A parallel file system for linux clusters. In *Proceedings of the 4th annual Linux Showcase and Conference*, pages 391–430, 2000.

[7] C. Staelin. lmbench: Portable Tools for Performance Analysis. In *In USENIX Annual Technical Conference*, 1996.

[8] A. Varga and R. Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, pages 60:1–60:10. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.