Interaction of Access Patterns on dNFSp File System

Rodrigo V. Kassick
rvkassick@inf.ufrgs.br

Francieli Z. Boito
fzboito@inf.ufrgs.br

Philippe O. A. Navaux
navaux@inf.ufrgs.br

Universidade Federal do Rio Grande do Sul: Instituto de Informática
Porto Alegre, Brazil

Abstract

HPC applications executed in cluster environments often produce large quantities of data that need to be stored for later analysis. One important issue with the performance of these applications is the transport and storage of this data. To provide an efficient data storage facility, cluster systems may provide a Parallel File System in which applications can store their data. These systems aggregate disk and network bandwidth of several individual machines in order to improve the performance of data storage and retrieval. These storage systems are often shared by different applications running in the cluster environments in which they are deployed. Concurrent access to the storage service may influence the I/O performance of these applications due to their spatial and temporal access patterns. In this paper we explore the influence of the periods of inactivity of an application with periodic behavior over the performance of an I/O bound execution. This study allows us to study the how a periodic application will influence the overall performance of a shared parallel file system.

1. Introduction

In the latest years, Cluster architectures have arise as the *de-facto* architecture for High Performance Computing (HPC) due to their versatility, scalability and low cost when compared to proprietary solutions. Applications developed for such environments are distributed among different processors and employ message-passing mechanisms to communicate. This has led to the development of faster processors and networks, providing a better performance for applications.

Cluster applications may need not only to process and exchange data, but also to store it. This data may be necessary for use in future executions, analysis using external tools, etc. For such cases, the use of a **distributed file system(DFS)** might be suitable. DFSs provide a large capacity, high-throughput storage area that can be accessed by many clients – like a cluster of computers. The tasks relative to data storage and management are distributed among a set of servers – hence distributed file systems – in order to increase data throughput and the system's scalability.

Many distributed file systems have been studied and developed in latest years, like Lustre [6] and PVFS [10]. dNFSp [1] is a DFS developed with the intend of providing high-performance storage while keeping compatibility with the standard NFS protocol.

When using DFSs, huge amounts of data must be transmitted between the clients and the servers. On the other hand, when different applications are executed concurrently on the same cluster, using the same shared parallel file system, their behavior regarding the usage of the common storage area will affect each other.

One common behavior of parallel applications is to alternate between CPU-bound phases in which it generates data and IO-Bound phases in which it will flush this data to the storage subsystem. In such cases, there will be a period with few or none I/O requests. During such time, other clients might be doing their own I/O operations.

This paper studies the behavior of two concurrent applications executing over the same shared set of storage nodes using dNFSp. We present results regarding the behavior of these applications with different processing times between each of their I/O phases.

The paper is divided as follows: Section 2 gives a brief overview of the temporal access patterns used in the tests. Section 3 introduces dNFSp, the distributed file system we used for our experiments. In Section 4 we show the proposed tests and the test-environment used. Section 5 presents the results and draws some conclusions on them.

Finally, in Section 6 we draw our conclusions.

2. Temporal Access Patterns

Scientific applications usually need to access large data sets used as input and, after processing such input, generate data to be written to a high performance storage system.

This access and generation of data is not necessarily done in well defined periods at the beginning and end of the execution. Applications may read portions of the input as they process previously accessed data[1] or as new instances of the application are created[2]. Output data may be generated along the whole execution of the application and sent to the file system[3], as memory constraints require offloading of data from the processing nodes or as the application writes checkpoint files to allow for later execution[4].

In such cases, access to the file system may be bursty during some periods of time during the execution, separated by periods of I/O idleness.[8, 7]. Figure 1 shows an access-trace taken from dNFSp while a synthetic benchmark with interleaved I/O and processing phases. While the client 3 does almost no access to the storage system, the other clients exit from one processing phase and perform a set of write operations. At time offset $4000ms$, there is a period of few activity, followed by the beginning of a new I/O phase.

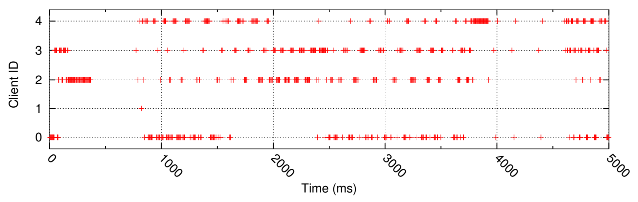


Figure 1. A Temporal Access Pattern on dNFSp: Time x Client ID

When more than one application share the storage system, such temporal patterns may influence the I/O performance of all the applications that make use if. Applications with a large I/O inactivity time may have little influence over the overall performance of the system while long-running I/O-bound applications may degrade the performance of applications less data-intensive.

1 mpiBLAST, a MPI-based parallel implementation of BLAST, for genomic sequence search [3]

2 The phase one of ESCAT, a parallel implementation of the Schwinger Multichannel method for study of low-level electron-molecule collisions [9]

3 The phase two of ESCAT.

4 Flash, a code for astrophysical simulations [4]

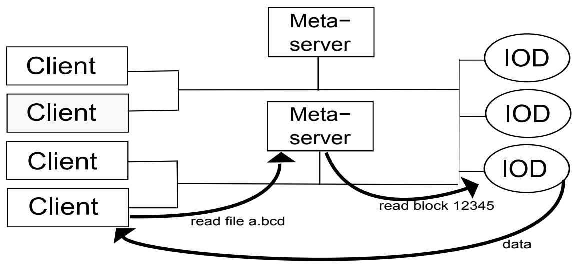


Figure 2. Read Operation in dNFSp

3. dNFSp

dNFSp — distributed NFS parallel — is an extension of the traditional NFS implementation that distributes the functionality of the server over several processes on the cluster. The idea behind dNFSp is to improve the performance and scalability and, at the same time, keep the system simple and fully compatible with standard NFS clients.

Similarly to PVFS, dNFSp makes use of I/O daemons – IODs – to access data on the disks. The role of the NFS server is played by the meta-data servers – *meta-servers*. These are seen by the clients as regular *nfsd* servers; when a request is received, the meta-server forwards it to the corresponding IOD to perform the operation.

The clients are distributed over several *meta-server* and only send requests to their associated server. The I/O daemons are shared by all meta-servers. When using dNFSp, if all the clients access the system at the same time, the load will be distributed among the meta-servers. This is specially important in write operations, once clients send whole data blocks to the meta-servers.

Figure 2 illustrates a typical read operation in dNFSp. The client sends a normal NFS read request to its associated meta-server. The meta-server then forwards the request to the IOD responsible for that portion of the file. After processing the request, the IOD answers directly to the client, avoiding unnecessary data retransmissions.

4. Two-Fold Application Interaction on dNFSp

To study the performance of dNFSp under the load from different applications, we utilized the *MPI-IO Test 21*[5] synthetic benchmark. It utilizes MPI and MPI-IO to simulate an application writing an specified number of *objects* of a given *size*. Each object represents a contiguous region of the file. Two instances of the benchmark, called *foreground* and *background* respectively, were executed concurrently over a shared dNFSp system.

In each test, both instances are executed with the same object size during a fixed amount of time, writing as many objects as the time permits. The foreground instance is configured to wait an specific time in between writing two objects, simulating the processing time of a real application. The background instance writes its objects with no such interval, simulating an I/O-bound application writing a checkpoint file, for example.

The benchmark was executed with three object sizes: $128Kb$, $2Mb$ and $4Mb$. Small object represent applications that have a very long processing phase to produce very few data. *MPI-IO Test 21* was configured to execute with one file per MPI process. This way, the access was contiguous per process.

The tests were executed on the Pastel cluster, part of the Toulouse site of Grid5000 [2]. All the nodes are equipped with AMD Opteron processors of 2.6GHz, 8Gb of RAM and 250GB SATA hard disks. The nodes are interconnected by a Gigabit Ethernet network.

dNFSp was executed on 6 nodes, each acting as both meta-server and IOD. 24 nodes acted as clients, being evenly distributed among the meta-servers. The nodes were divided in two sets, each executing an instance of the benchmark. The foreground and the background instances of the benchmark were executed with 12 clients each. The bandwidths presented only account for the I/O portion of the benchmark bandwidth, not including the time spent in between writes. The tests were executed 6 times. The bandwidth presented is the average of these executions.

5. Results

In this section we present the results obtained with the tests described. The objective of this test is to study the bandwidth of both instances of the benchmark in the presence of a well-defined temporal access pattern.

Figure 3 shows the performance for the I/O portion of the benchmark. Each graph shows the write bandwidth for the foreground and background instances of the benchmark along with their combined bandwidth. The X-axis represents the time interval between each write operation in the foreground. Immediately below to each case is the **time ratio** for the foreground instance, i.e. the ratio between the time spent on I/O over the total execution time (including the waiting time).

With intervals ranging from 10 to $500ms$, the foreground write bandwidth for the tree object sizes showed few variation. As the interval reaches 5 seconds, the 2 and $4Mb$ cases have a drop in performance, while the small object-size case shows less variability.

The foreground $128Kb$ object size apparently takes better advantage of the client's caching subsystem, since it outperformed the cases with greater write-unit size. Since the NFS clients used in the tests employ a delayed write technique, small write units interleaved with I/O idleness result in the operating system performing the writes in the background while the application is performing other operations, resulting in a short I/O time from the point of view of the application. For 2 and $4Mb$ object-sizes, the operating system's write cache won't be able to hold all the data and, therefore, the application will spend more time blocked in the *write* call.

Since the background instance does not wait for the specified interval, it profits from the idleness in the foreground. With 128Kb size objects, the background has a significant performance improvement with bigger intervals. This is due to the very short I/O bursts from foreground followed by long periods of idleness, leaving the system free to process other requests. For 2Mb and 4Mb sized objects, on the other hand, I/O bursts from the foreground are longer, and even with long periods of exclusiveness, the background instance gets a more modest increase in it's write bandwidth.

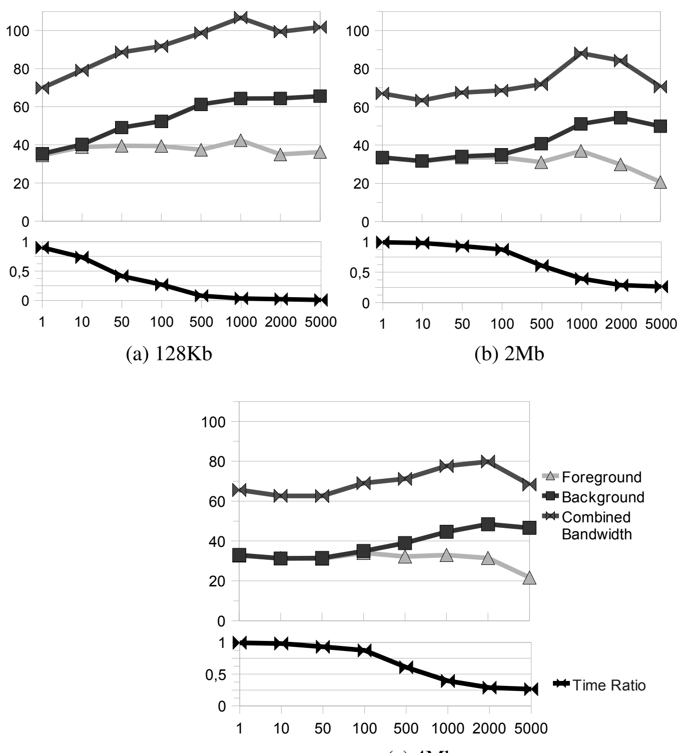
Nonetheless, with the increase of the idle time, the background will profit from higher bandwidth availability during such periods of inactivity and, as a result, will increase it's I/O bandwidth. This increase seems to be limited on the background clients: for 2 and $4Mb$ object sizes, the bandwidth decreases with a $5s$ interval on the foreground. Despite a greater resource availability on the servers, the background instance is not able to produce enough data to profit from this situation. This is a likely result of the delayed write mechanisms in the Linux NFS client.

The stability of the background bandwidth, allied with the drop in bandwidth on the foreground, results in smaller combined bandwidths for longer idleness periods. The reason for the decrease in the foreground bandwidth with the longer intervals, as seen for object sizes of 2 and $4Mb$ remains to be studied.

One question that arises from these tests is how much the bandwidth of the two instances diverge with the different intervals and what's the point in witch this difference starts to get more noticeable.

As seen in Figure 3a, the bandwidth for both instances is very close for intervals of 1 and $10ms$, diverging for larger intervals. The time ratio is 0.9, 0.74 and 0.41 for intervals 1, 10 and $50ms$ respectively. In Figure 3b, the divergence started with intervals of $500ms$, when time ratio dropped from 0.87 to 0.61. Figure 3c shows that with $4Mb$ sized objects, divergence starts at the same point as with $2Mb$, with ratio dropping from 0.93 in $100ms$ to 0.74 in $500ms$.

This leads us to infer that the minimum time ratio in which both instances will still have similar performances is proportional to the size of the objects. This is explained by the relation of the idle interval to the amount of data written after it. With small sized objects, the idle interval can become significantly longer than the time to write one single object. Since such interval happens between writing two objects, the total execution time becomes dominated by the idle time and the ratio tends to fall more quickly with longer intervals. With larger objects, small idle times are less significant to the total execution time and so the ratio takes longer to fall.



(a) 128Kb (b) 2Mb (c) 4Mb

Figure 3. Bandwidth in Mb/s for Foreground and Background instances and I/O-to-Execution Time Ratio

6. Conclusions and Future Work

In this paper, we have shown the behavior of dNFSp file system when two different applications make heavy use of it. We have evaluated the I/O performance of these applications under different idleness periods and objects sizes.

The results indicate that the performance for an individual application is affected by the execution of other applications over the shared system. We could observe that long idle I/O periods in one application allowed for increased I/O performance in a concurrent execution. This performance, on the other hand, is bounded by the rate in which client nodes send data to be written in the file system, limiting the overall performance of the file system under the studied access pattern. We also could observe that the time ratio at which the performance of both applications diverge seems to be proportional to the size of the objects written in the system.

As future work, we intend to further study the relation between the time ratio and the performance of the I/O subsystem and the reasons for the decay in the foreground bandwidth under longer idle intervals. These information will be used in the development of an on-line monitoring and reconfiguration tool for the dNFSp file system.

References

[1] R. B. Ávila, P. O. A. Navaux, P. Lombard, A. Lebre, and Y. Denneulin. Performance evaluation of a prototype distributed NFS server. pages 100–105.

[2] R. Bolze, F. Cappello, E. Caron, M. Dayde, F. Desprez, E. Jeannot, Y. Jegou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and I. Touche. Grid'5000: A large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494, 2006.

[3] A. E. Darling, L. Carey, and W.-C. Feng. The design, implementation, and evaluation of mpiblast. In *In Proceedings of ClusterWorld 2003*, 2003.

[4] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, J. W. Truran, and H. Tufo. Flash: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes. *The Astrophysical Journal Supplement Series*, 131(1):273–334, 2000.

[5] Los Alamos National Laboratory (LANL). MPI-IO Test 21, 2009. Available in `http://institutes.lanl.gov/data/software/#mpi-io`.

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

[7] S. Seelam, I.-H. Chung, D.-Y. Hong, H.-F. Wen, and H. Yu. Early experiences in application level i/o tracing on blue gene systems. *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–8, April 2008.

[8] N. Tran and D. A. Reed. Arima time series modeling and forecasting for adaptive i/o prefetching. In *ICS '01: Proceedings of the 15th international conference on Supercomputing*, pages 473–485, New York, NY, USA, 2001. ACM.

[9] C. Winstead, H. Pritchard, and V. McKoy. Parallel computation of electron-molecule collisions. *IEEE Comput. Sci. Eng.*, 2(3):34–42, 1995.

[10] Y. Zhu, H. Jiang, X. Qin, D. Feng, and D. R. Swanson. Design, implementation and performance evaluation of a cost-effective, fault-tolerant parallel virtual file system. *International Workshop on Storage Network Architecture and Parallel I/Os, held with 12th International Conference on Parallel Architectures and Compilation Techniques*, 2003.