Testing the Performance of Parallel File Systems

Rodrigo Virote Kassick Francieli Zanon Boito
Philippe O.A. Navaux
Grupo de Processamento Paralelo e Distribuído
Universidade Federal do Rio Grande do Sul
{rvkassick,fzboito,navaux}@inf.ufrgs.br

Abstract

Cluster architectures allow applications to profit from it's distributed processing and to scale in an efficient and cheap manner. The slow evolution of I/O devices, on the other hand, turns the I/O subsystem into a great bottleneck for applications that need to store and manage large sets of data. Parallel File Systems try to overcome such problems through the distribution of the IO system into several independent servers, thus aggregating their storage and network bandwidths. The performance of such systems can be affected by how they manage the application's data on the storage level or how they transport such data through the network between the servers and the clients. This work proposes a study of such behaviors to define what are the best strategies to be used by an application when using such systems.

1. Introduction

Cluster Architectures, as defined by Baker et al[1] are composed by a set of independent machines connected through a network. Applications gain in performance exploring the processing parallelism of such architecture and using the network to communicate data between different instances when needed.

Several scientific applications manipulate large sets of data and need to store this data in between executions. Since the data must be available to all the machines in the cluster, the utilization of a distributed file system is a very natural solution.

Given the necessity of good performance on such systems, traditional centralized distributed file systems are not suitable to this task, since the central server will quickly became a bottleneck when several clients try to access the files. Parallel File Systems or High Performance Distributed File Systems are a kind of distributed file system in which the client's data is distributed onto several servers. This way, the network bandwidth and the disk capacities are summed, improving the performance of the storage system. Two of such storage systems are the dNFSp and the Lustre File System.

dNFSp – Distributed NFS, Parallel – is a parallel file systems that keeps full compatibility with the NFSv3 protocol. Such protocol has the advantage of it's widespread availability – the necessary drivers have been part of the Linux kernel for several years – thus not requiring any special configuration on the client side.

Lustre (a blend between the words Linux and Cluster), on the other hand, is a built-from-the-ground distributed file system that aims at profiting from modern network and storage technologies (like multiport FiberChannel-connected disks and high performance SAN[1] like Infiniband and Myrinet).

These systems treat both the data transport and it's storage in different manners. As a result, same data-access patterns may result in different performances in these file systems.

Getting to know how these access patterns behave in each of these file systems is important both to know how to set up the systems for different applications and to know how to develop the application's I/O subsystem to profit the best out of these storage systems.

This work proposes a set of tests to evaluate these file systems in situations similar to those experienced when executing scientific applications.

The remainder of this paper is divided as follows. Section 2 and Section 3 give an overview of the dNFSp and Lustre parallel file systems. In Section 4 we present a test set that will allow us to examine certain aspects of both file systems. In Section 5 we present some conclusions.

1 System Area Network

2. Parallel File Systems

dNFSP[2] is a parallel file system based on the NFSv3 protocol. This way, it is compatible with any client that has drivers to use normal NFS servers - virtually any UNIX based system, like the Linux and BSD family.

Client access the file system through a metadata server or metaservers. A metaserver is responsible for keeping track of the file's metadata. Clients are divided among the set of available metaservers, so the load is divided among them.

dNFSp divides data in fixed-length blocks and distributes them equaly among a fixed-length-set of storage destinations (VIOD's). This technique is well known on parallel file systems and is called "striping".

A VIOD in dNFSp is composed by a set of at least one storage server (or IOD, I/O Daemon). A single IOD may be present in more than one VIOD, although in this case the performance of the system may be degraded. VIOD's with more than one IOD use these servers to replicate data. This is useful for fault-tolerance, allowing up-to-date servers to replace failed ones.

dNFSp also allows the utilization of exclusive data servers for an application. These IOD's are called "application IOD's" and they treat requests coming from a list of clients belonging to a same application.

An application IOD substitutes a whole VIOD during the execution. As a result, there's no mirroring of data until the end of execution. While this sacrifices fault-tolerance while the application is being run, the gains in performance are significant[3].

Since dNFSp utilizes NFS protocol, client-server communication is done via Sun RPC. The metaserver-IOD communication is done via simpler message-driven protocol.

3. Lustre File System

The Lustre File System [4] was created intending to remove the bottlenecks commonly found in traditional parallel file systems. Lustre tries to scale to a large set of client nodes (1000 and above). It's architecture is composed by a centralized metadata service (MDS server) and several object-based storage servers (Object Storage Targets, OST's).

Like in dNFSp, the metadata service is responsible for information on the data and it's distribution on the OST's. The centralized MDS was a choice made during the development of the file system - it was then considered that intensive metadata operations workloads were not very common, and as a result, the cost of keeping a distributed version of such service would be too high compared to the advantages.

To decrease the impact of such workloads, Lustre clients implement metadata cache. The MDS must then keep track

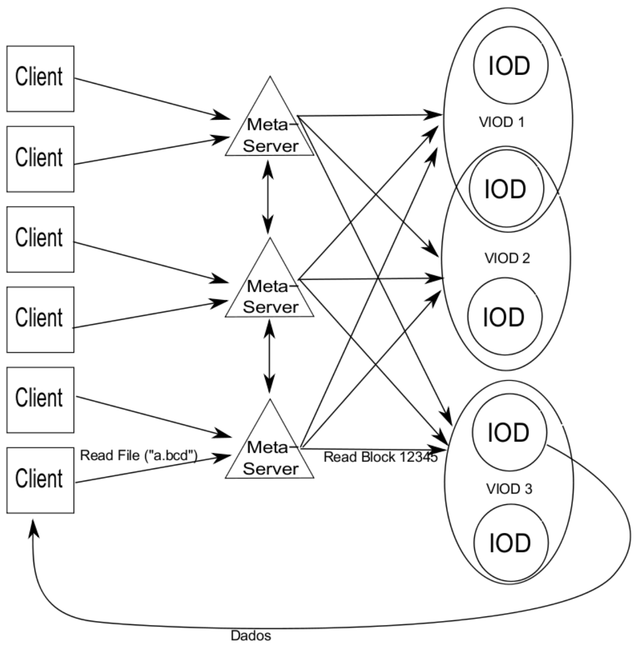

Figure 1. dNFSp file system's architecture

of such cached data and assure some level of consistency among them.

Lustre data storage is based in objects. An object represents a storage unit and resides in a single OST. An object may be anything from a whole file to a slice of it. The striping of the file into objects and their placement on the available OST's might be defined by the user via a command line interface.

Lustre fault tolerance is based on fail-over servers. MDS and OST servers may have backup server that shares the same storage device (as a multiport disk or a network-synchronized partition line DRBD[5]).

Lustre's network communication layer was based on the Portals API [6]. Such library allows the utilization of advanced network level resources like RDMA[2] transfers and can be implemented within advanced network interconnection chips to profit from some network specific details.

4. Parallel File Systems Testing

The main objective of any parallel file system is to increase the total performance of file I/O for applications. On the other hand, applications utilize such systems in very specific patterns that may cause the system to perform poorly.

2 Remote Direct Memory Access

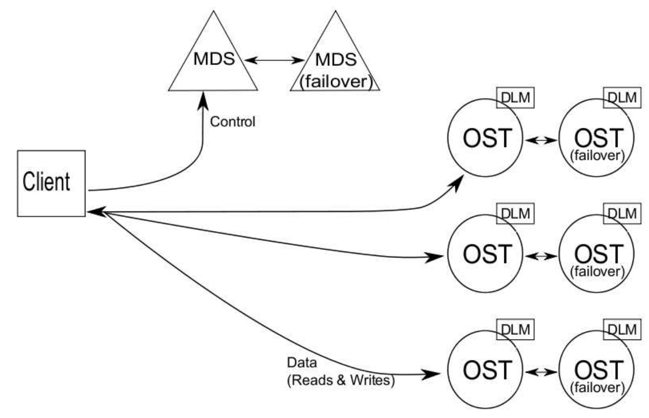

Figure 2. Lustre File System's Architecture

Figure 2. Lustre file system's architecture

Simply testing the file system to evaluate it's raw performance (how much data it is able to transfer to client's in a given time) does help giving an overall idea of the PFS performance, but there is no guarantee that applications will have the same performance level as the benchmarks.

Some aspects of the file system may affect the performance of applications according to how they make access to data. We focus on two of these aspects: data caching and data distribution.

Data caching is used to avoid repeated requests to be treated repeated times. In distributed file systems, caching is yet more important as it avoids not only the treating of the requests multiple times, but also avoids the transport of significant amounts of data from clients to servers.

Since the storage on PFS is normally divided among several servers, the way that an application's data is distributed into them may influence the overall performance of the I/O system. Additionally, the distribution of data from different applications may influence the performance when they are executed concurrently on the cluster.

With this in mind, we propose a set of tests that try to evaluate the dNFSp and Lustre file system according to some access patterns observed in bibliography[7, 8, 9].

The proposed tests are:

SFWR - Single File, Whole Read The test consist in several clients reading a whole shared file.

SFWR-SMP - Single File, Whole Read, SMP-Nodes Just like SFWR, but several instances of the application are executed on a same computing node.

SFSR - Single File, Segmented Read This tests utilizes sequential reads just like SFWR, but there is no overlapping in data read by two instances of the application.

SFSS - Single File, Shared Segments This test is divided into 2 distinct phases: first, each instance of the parallel application does read and write operations on it's own segment of a shared file. In the next phase, segment's offsets are altered so that they include data written by a neighbor instance.

MFNS - Multiple File, No Sharing In this test, each instance of the application reads and writes from it's own file instead of it's exclusive segment (as if SFSR).

MFSS - Multiple File, Shared Segments This test evaluates the behavior seen in SFSS with the one-file-per instance strategy of MFNS.

The **SFWR** and **SFWR-SMP** tests focus on read performance. They try to emulate applications that have large input datasets that must be read by all clients as part of the application initialization. The SMP variant of such test tries to evaluate if client-side caching is able to cope with several repeated read requests. This situation tends to become more common with the deployment of several-core computing nodes.

SFSR simulates distributed applications that operate over a shared file but that work over independent segments of it. This is common for applications that read large datasets in their initialization but that need to operate over independent data through the first computing phase. This test allows us to see the performance of the file system without the influence of caches (client side or server side), given that no two instances read the same data. Performance change due to data distribution might be noticeable in this test.

The **SFSS** tries to simulate applications that initially operate over independent datasets but that eventually need access to a neighbor's data. This test focus on the change of application's behavior and if original data distribution will affect performance of the second I/O phase.

The Multiple-File family of tests intends to observe multiple-file data distribution and how it affects the performance of the applications on the I/O level. **MFNS** is an equivalent of SFSR, since both operate over independent segments of a larger dataset. In MFNS, on the other hand, each segment of data is stored into an independent file.

MFSS is an equivalent of SFSS. Into it's second phase, application's instances will to access data from neighbors and, in MFSS case, that means reading from different files.

5. Conclusions

Several aspects may influence the performance of a parallel file system. We are proposing some tests to evaluate two PFS's behaviors under situations similar to those found in real-life HPC applications and study some aspects considered important in a PFS.

Knowing the behavior of file systems under certain conditions is important both to the development of parallel applications. Developers may try to make their code in manners that profit from certain characteristics of the file system while avoiding to do operations that are known to degrade performance.

Administrators might also use the results of such tests to deploy a PFS configuration that better fits the kind of application expected to be executed in their clusters.

References

[1] Mark Baker and Rajkumar Buyya, "Cluster computing: the commodity supercomputer", in *Software and Practice and Experience*, 1999.

[2] Rafael Bohrer Ávila, *Uma Proposta de distribuição do servidor de arquivos em clusters*, PhD thesis, Universidade Federal do Rio Grande do Sul, 2005.

[3] Everton Hermann, Danilo Fukuda Conrad, Francieli Zanon Boito, Rodrigo Virote Kassick, Rafael Bohrer Ávila, and Philippe Olivier Alexandre Navaux, "Utilização de recursos alocados pelo usuário para armazenamento de dados no sistema de arquivos dnfsp", in *Anais do VII Workshop em Sistemas Computacionais de Alto Desempenho (WSCAD 2006)*, Ouro Preto - MG, October 2006.

[4] "Lustre: A scalable, high-performance file system", Cluster File Systems Inc. white paper, version 1.0, November 2002.

[5] "Distributed replicated block device homepage", 2006, `http://www.drbd.org/`.

[6] "Portals api", 2008, `http://www.cs.sandia.gov/Portals/`.

[7] Tara M. Madhyastha and Daniel A. Reed, "Learning to classify parallel input/output access patterns", *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 8, pp. 802–813, 2002.

[8] Evgenia Smirni and Daniel A. Reed, "Workload characterization of input/output intensive parallel applications", in *in Proceedings of the 9th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*. 1997, pp. 169–180, Springer-Verlag.

[9] Tara M. Madhyastha and Daniel A. Reed, "Input/output access pattern classification using hidden markov models", in *In Proceedings of the Fifth Workshop on Input/Output in Parallel and Distributed Systems*. 1997, pp. 57–67, ACM Press.