Performance Evaluation of Meta-Data Transfer and Storage in Clusters

Everton Hermann[1], Rodrigo V. Kassick[1], Rafael B. Ávila[1], Carlos J. Barrios-Hernández[2,3], Michel Riveill[3], Yves Denneulin[2], Philippe O. A. Navaux[1]

[1] Instituto de Informática/UFRGS
{ehermann,rvkassick,avila,navaux}@inf.ufrgs.br

[2] Laboratoire d'Informatique de Grenoble/IMAG
Projet Mescal
{barrios,Yves.Denneulin}@imag.fr

[3] Laboratoire de Informatique, Signaux et Systèmes/UNSA
Projet Rainbow
riveill@unice.fr

Abstract

An expressive number of scientific application executed on cluster architectures need some form of permanent storage with high capacity. As a consequence, the performance of the storage system becomes crucial to such applications. Besides granting data I/O performance, it's important to provide efficient and scalable access to file's meta-data, allowing the system to scale and, yet, offer a good level of consistency in the shared file system. Also the file transfer between nodes of the cluster architectures is an important factor that affects the I/O performance. And this is important to know the behavior and influence in the shared file system. In this paper we present a low-overhead meta-data synchronization algorithm used in dNFSp, a distributed file system based on NFS that proposes changes exclusively on the server side, keeping compatibility with traditional NFS clients. With this synchronization schema, dNFSp can offer the same level of coherence of traditional NFS servers. We also evaluate the communication characteristics of the environment where the tests were executed in order to analyze possibilities of scalability of the system.

1 Introduction

With the popularization of Beowulf-class parallel machines and the constant improvements in technologies for computer components, it is becoming more and more common the deployment of clusters with hundreds or thousands of nodes. For such large-scale systems, concurrent access to shared I/O services becomes a critical bottleneck. Traditional distributed systems such as NFS [1] are not suitable for such scenarios due to its centralized nature. On the other hand, such systems are convenient to use due to their maturity and wide availability of implementations and tools. The challenge is then to provide file systems with the high performance and scalability levels needed in a cluster and the compatibility with widely accepted standards like NFS.

dNFSp [2, 3] is one such attempt. Based in the NFSp filesystem, dNFSp extends the traditional NFS implementation in order to support high performance I/O with the particular interest of not touching the client side NFS protocol, remaining compatible with traditional implementations such as the one found in the Linux kernel, commonly used in Beowulf clusters.

dNFSp, in contrast with NFSp, distributes the meta-data management onto several servers (called meta-servers) and delegates a subset of the clients to each of these servers. This way, dNFSp, is able to scale better in write operations then NFSp [4]. This distribution requires the implementation of synchronization mechanisms to keep meta-data consistent among the clients, allowing them to share files in a consistent manner.

The first version of dNFSp used a relaxed consistency schema that allowed meta-data to remain unsynchronized and relied on the client to make accesses to the file independently from it's meta-data.

In order to improve the level of consistency among the clients of different meta-servers, we developed a new synchronization mechanism for dNFSp2 using hashing for locating meta-data. While the hash-based schema is com-

monly used in other filesystems by the client, our implementation operates solely on the meta-server level, maintaining compatibility with the standard NFS protocol. This paper describes this synchronization mechanism it's performance evaluation.

On another hand, the file transfer process is critical during the execution of parallel applications in clusters. As it is known, it exists a direct relation between the performance of the massive/intensive communications and the file system, like the are with the transfer protocol, architecture of the system, the model of communication and another, that also, your performance is related with the file system. Different strategies are implemented in the file system, for example, to guarantee the a synchronous behavior, the coherence of the data, a minimum delay in the communication. This paper present an evaluation experimental of performance in the highbandwidth data transfer, in order to show the impact of the transfer process in the system behavior.

The remainder of the paper brings some background information dNFSp in Section 2, a description of our proposed method and obtained results in Sections 3 and 4, a description about the file transfer process is presented in Section 5, a discussion on related work in Section 6 and our final conclusions in Section 7.

2 An Overview of dNFSp

dNFSp — distributed NFS parallel — is an extension of the traditional NFS implementation that distributes the functionality of the NFS daemon 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 daemon are seen by the clients as regular nfsd servers; when a request for a given piece of data is received, the meta-server forwards the request to the corresponding IOD to perform the operation and answer to the client.

dNFSp distributes the *meta-server* function onto several computing nodes. The I/O daemons are shared by all meta-servers. However, each meta-server is associated to only a subset of the clients. When using dNFSp, if all the clients access the file 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 1 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 storing that piece of the file. After processing the request, the IOD answers directly to the client.

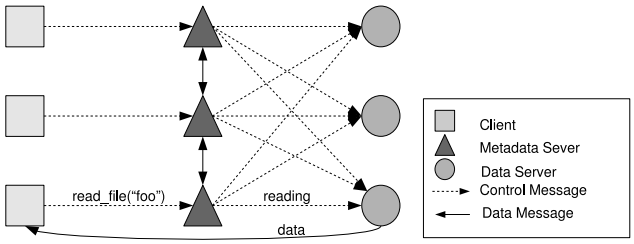

Figure 1. Example of a read operation on dNFSp

The distribution of the meta-server brings a new problem: keeping the meta-servers synchronized in order to allow consistent access to the same file from clients in different meta-servers. In the first version of dNFSp (dNFSp1), this synchronization was made with an LRC [1] based algorithm, in which data is updated in one node only when it needs access to it. The employment of LRC allows nodes to have outdated information, leaving the system partially unsynchronized.

In dNFSp1's LRC-based synchronization method, the meta-data of a file created in one meta-server is propagated only when a client in another meta-server tries to access the same file for the first time. In this case, the outdated meta-server will search on the other meta-servers for the corresponding meta-data and then make a local copy of it. No further synchronization is made afterwards, allowing some file attributes to remain incoherent. For a large number of applications this level of synchronization is enough to keep the system working correctly. This scheme also has the advantage of requiring a small number of messages among the meta-servers.

3 Hash-Based Synchronization Schema

The presence of outdated meta-data can lead to inconsistencies when clients in different meta-servers access the same file. This inconsistency appears when data written by a client depends on some meta-data information like the file size. When two clients associated to different meta-server coordinately append content to the same file, inconsistency on the file size can result in overlapped writes, resulting in data loss.

To avoid such inconsistencies, the algorithm employed in the second version of dNFSp (dNFSp2) ensures that the meta-data viewed by one meta-server is the most recent in the system. This synchronization schema is based on a hash mechanism, as used by other file systems like Vesta [5, 6],

[1] Lazy Release Consistency

Expand [7] and Archipelago. In dNFSp, however, the hashing technique is used only in the meta-server level.

Each file is associated to a Hash meta-server, which is responsible for the file's meta-data. This association is done using a hash function based on the name of the file. When a meta-data server other then the given Hash tries to access a file, it needs first to contact the Hash to obtain the meta-data.

File creation follows the same hashing scheme. When a file is created, the meta-server receiving the request computes the id of the Hash server based on the file name, and contacts it, notifying that a file will be created. The Hash will then create a local copy of the meta-data for future access.

To avoid unnecessary communication between nodes, we have adopted a caching mechanism based on tokens. The token indicates which server has the most recent version of the meta-data. This information is kept in the file's meta-data in the corresponding Hash server.

The token represents the meta-server with the most recent version of the meta-data. It can by owned by any of the meta-servers; if a meta-server owns the token, it doesn't needs to contact the Hash server before accessing the meta-data as the cached meta-data is up to date. The token transmission from a meta-data server to the other is always managed by the Hash server.

The algorithm used to manage the token is represented on Figure 2. The simplest case is when the meta-data is found locally and server treating the request owns the token – there is no need to contact the Hash before answering to the client (1). If the meta-data is not found locally, or the meta-server does not owns the token, the meta-server communicates with the Hash (Meta B) before treating the request (2). When the Hash owns the token, it can immediately answer with the most recent meta-data. Otherwise, the Hash answers with the address of the meta-server owning the token (Meta C). The meta-data server then contacts owner of the token (5) requesting the meta-data. After receiving the meta-data, it can continue treating the client's request.

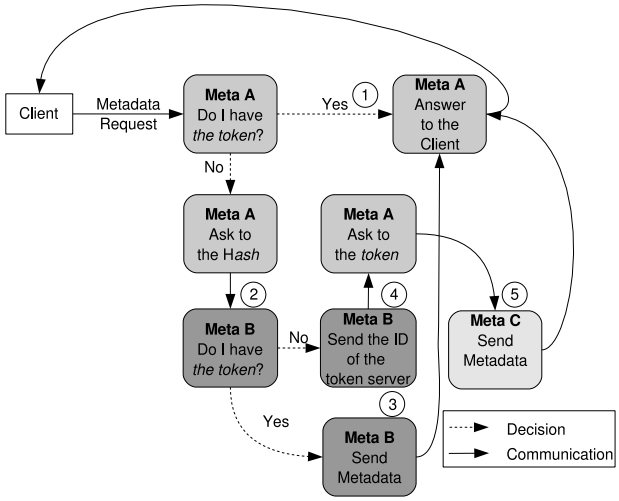

Figure 2. Token management fluxogram

4 Results and Discussion

The tests presented on this section were executed on the *I-Cluster2* [8] installed in Montbonnot Saint Martin, France, in the INRIA Rhône-Alpes facility.

The I-Cluster2 is part of the Grid 5000 infrastructure[2], the french grid for research in high performance computing and e-Science. The cluster is composed by 100 nodes. Each node is equipped with a dual Itanium2 900 MHz with 3 gigabytes of memory and a disk storage with 72 gigabytes, 10000 rpm, SCSI. All the nodes are interconnected using a 1 Gigabit Ethernet network, Fast Ethernet network and Myrinet Network. The experiments were performed using the 1 Gigabit Ethernet network. The software installed on *I-Cluster2* is based on Debian stable distribution, with a Linux kernel version 2.6.15. In the tests we compared dNFSp1, dNFSp2 and PVFS2. The version of PVFS2 used on our tests was 1.5.

[2]http://www.grid5000.fr

4.1 Raw data transfer tests

This test aims to evaluate the low level performance of dNFSp and compare it with other parallel filesystems. The test consists in writing the same amount of data into a different number of files (e.g. 1 file of 50Mb, 2 files of 25Mb, 128 files of 400Kb). With this method, we can see the impact of meta-data processing, once the total amount of data written is constant. In the ideal case, the increase in the number of files should not degrade the performance of the filesystem.

The tests were performed using 4 nodes as meta-servers, 4 as IOD's and 4 as clients of the filesystems. Each of these clients writes 50Mb of data splitted into 1 to 128 files in each step of the benchmark, summing 200Mb of data written in the filesystem.

Figure 3 shows the execution time of the benchmark for dNFSp1, dNFSp2 and PVFS2. By the results, we can see that dNFSp2 sustains almost the same transfer time for all the given sizes, while dNFSp1 and PVFSv2 have a significant performance degradation as the number of files grow and the file sizes decrease. As the minimum file sizes transferred is larger than the maximum block size written by dNFSp1, the file size does not represents the main reason of it's low performance. This performance degradation is due the $O(n)$ nature of dNFSp1 synchronization protocol. With the increasing number of files being created – and con-

sequently, lookup request from clients to servers – more and more messages need to be exchanged among the meta-servers to check for the existence of the file. With a smaller number of files, on the other hand, dNFSp1 and dNFSp2 performed almost the same, once few synchronization was needed.

PVFS also performed poorly with a greater number of smaller files. This poor performance with small writes has already been reported in other works [9] and can be attributed to it's write thought cache and poor paralelization of requests.

NFS stable transfer time was expected, since it's single server architecture does not demand any synchonization due to increased number of meta-data operations. dNFSp2 has the same behaviour (with a higher transfer rate), since the number of messages exchanged between the meta-servers due to synchronization is small.

5 Analysis of Data Transfer

The implementation of distributed and high performance resources requires the knowledege of the system's capacity and limitations. It is important then to evaluate two different aspects of the system: 1) The system's behavior when transfering data normal conditions and 2) the characteristics of the transfer in the moment of the execution of parallel applications.

For this case, the tests are proposed to measure communication aspects during the runtime of a parallel application. For this, it's used the code Logp-multites in the LogP-MPI Benchmark Tool[3]. LogP/MPI Benchmark Tool provides data that are expressed in terms of parametrized LogP.

The terms of LogP are defined in parameters for messages of size *m*: *P* is the number of processors, *L* the latency, $o_s(m)$ is the send overhead time and $o_r(m)$ is the reception overhead time, $g(m)$ is the gap between consecutive messages. So, a network can be described as $N(m) = (L, o_s, o_r, g, P)$.

The measurement during the execution is made between pairs of processors and it supposes the communication to be symmetrical in both ways. *Logp_multitest* is a program that allows to make measures in parallel proposed by Luiz-Angelo Estefanel from the ID-IMAG Laboratory[4]. In general, the options are the same that exist for the $logp\_test$ program and can be consulted in [10].

From the parameters, it's possible to build a graphic to observe the behavior for the transfer of 50MB in I-Cluster-2, as it is possible to see in the figure 4.

The points in figure 4 show the space of time between two consecutive transfers of files in a RTT[5] experience – in other words, the values of gap between each pair of nodes. The measure has been taken for many transfers between peers and with a fixed size of message of 50Mb. The values of gap are in microseconds. It's important to say that for this experience, one node uses only one processor.

With the different values is calculated the average to see a general behavior of the transfer. Then, it's possible to see two aspects: 1) the increase in the gap when there are many nodes in the transfer and 2) the peaks and irregularities in different points. The first aspect is explained by the saturation and increased use of the cluster network. The second aspect may be interpreted as the cost of the transfer in the time of execution by the influence of the architecture in the system, and in fact of the file system. To observe the general cost of the transfer, it is necessary to see the bandwidth behavior.

Figure 5 shows the bandwidth values in I-Cluster-2. These values are calculated with the expression proposed in the LogGP model described in [11] to incorporate long messages in the logP Model. The logGP model proposes to compute the end-to-end delivery time for k-bytes of a message m from a node to another with the following equation:

$$T(m) = O_{send}(m) + \frac{(k-1)}{g(m)} + L(m) + O_{rcvd}(m) \quad (1)$$

where O_{send} and O_{rcvd} are respectively the overhead time of the node involved in sending and receive the k byte of a long message, g is the gap, that is, the minimum time interval between consecutive messages transfers and L is the latency.

The figure 5 shows the decrease of the bandwidth in the process due to the use of the network resources in the cluster in the time of execution. But, the interesting aspect is the peaks in the graphic that they are associated to the own characteristics of the system, then, the file system.

6 Related Work

Among the main approaches to distribute data and meta-data in systems, we highlight two classes: the tree partitioning and the hash distribution. In the first case, the meta-data in the same sub-tree are stored in the same server. To access a given file, the client must access the meta-servers responsible to each sub-tree, following the file path. The meta-data servers are accessed hierarchically until the information is found. File systems like AFS (Andrew File System) [12], Coda [13], NFS and LFS (Log-Structured File System) [14] are implemented by servers that can be responsible for one or more sub-trees on the system.

[3]developed by the albatross project in Netherlands, http://www.cs.vu.nl/albatross/

[4]http://www-id.imag.fr/Laboratoire/Membres/Estefanel_Luiz-Angelo

[5]Round Trip Time

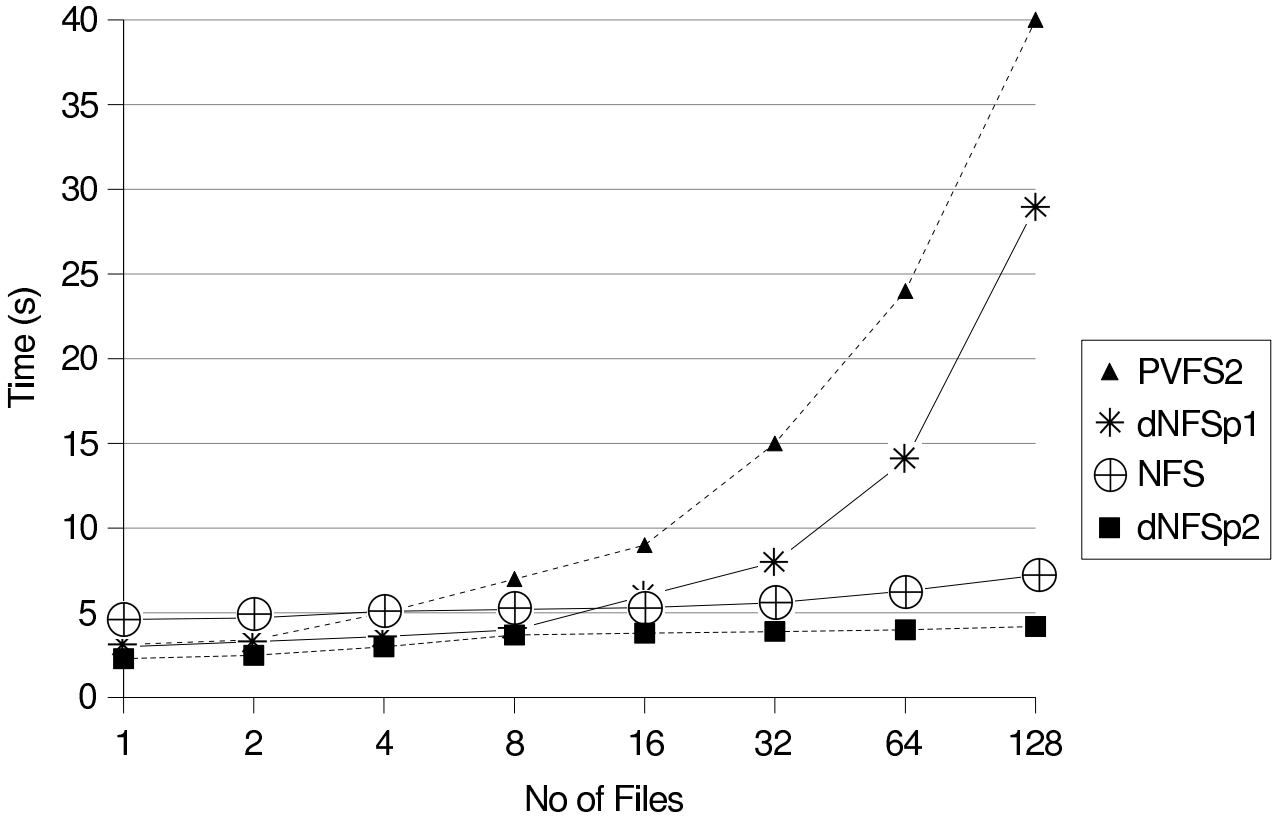

Figure 3. 50 Mb transferred by each client divided in x files

On works like the one of Levy and Silberschatz [15], it is possible to find more details about the sub-tree mechanism together with means to implement it. In these implementations, the main disadvantage is the load distribution. When a sub-tree is accessed more frequently than the other, there is an unbalancing among the servers. It results on more requests being treated by the server storing the most popular sub-tree.

The second way used to manage meta-data was introduced by the Vesta file system [5]. This system uses a hash function on the file name to obtain the meta-data location. Using an adequate hash function reduces the overload of a server in the system. With this technique it is possible to avoid the load unbalancing, which is the main problem of implementing a sub-tree algorithm. In the Vesta file system, the clients compute the hash based on the path of the file. With the result of this function the client knows which node it needs to contact to obtain the meta-data of the desired file.

The PVFS2 file system [16] uses an algorithm similar to the hash to support multiple meta-data servers. It is done by distributing across the server ranges of values that can be results of the hash function. This configuration is made at the time of the initialization of the system.

The meta-data management system called LH (*Lazy Hybrid*) [17] uses part of both methods, it distributes the directories using sub-tree and the files using a hash based algorithm. Another system implementing NFS that is based on the Vesta file system is the Expand [7]. The Expand system only modifies the client, keeping the servers intact. This characteristic is the opposite of the dNFSp objective which is to keep all the changes on the server side.

7 Conclusions

This paper presents a low-latency coherence mechanism that allows to perform efficient meta-data updates among distributed servers. The practical results have shown that the system is able to present high performance even in the presence of large amounts of meta-data operations. Also of great importance, the implemented mechanism does not degrade the high performance for read/write operations (i.e. data operations) that had been previously achieved by early implementations of dNFSp.

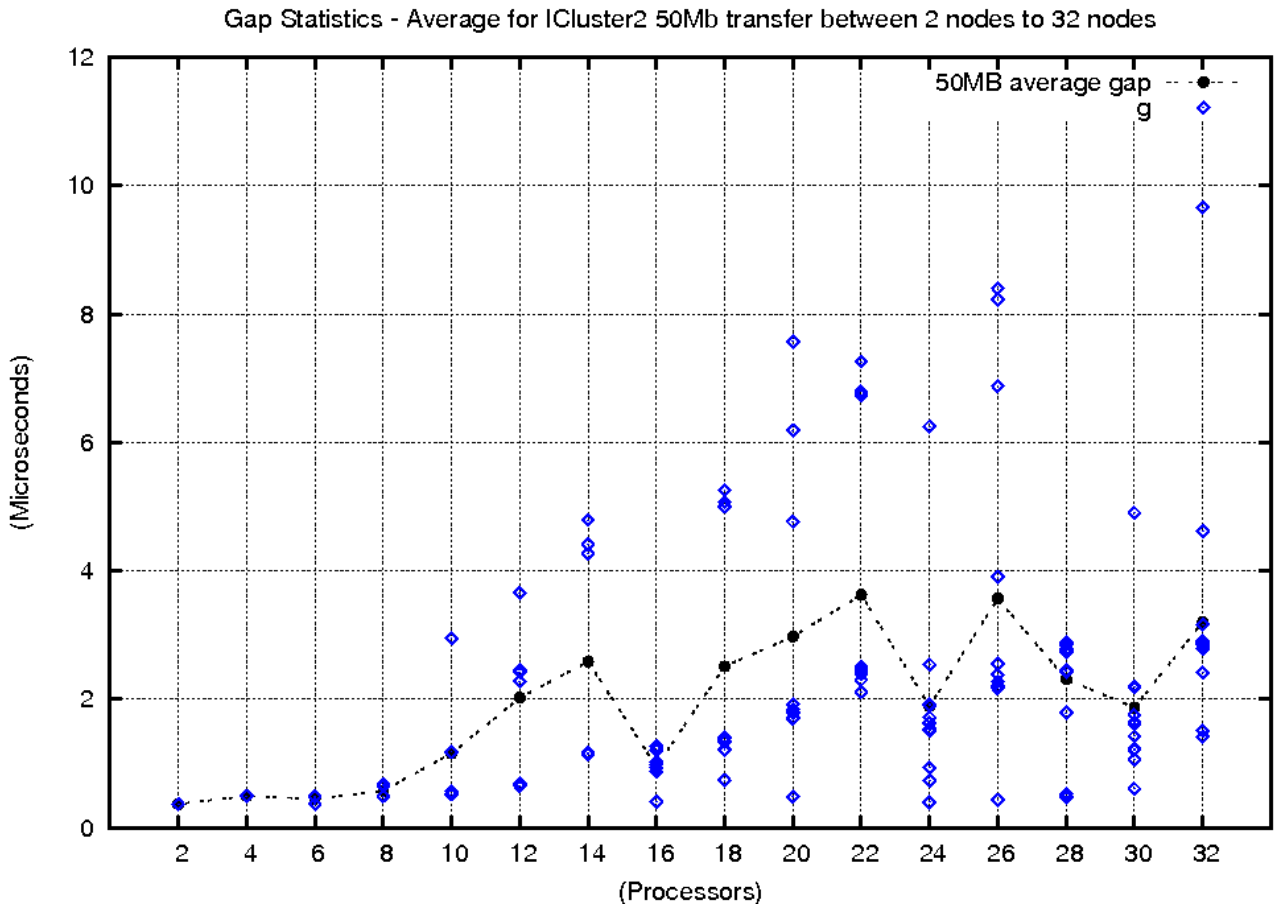

Figure 4. I-Cluster-2 gap in Transfer of 50MB

It's also important to note that the new mechanism allows dNFSp to serve several files without degrading data operations due to small block sizes, like PVFS.

In addition, the analysis of the transfer behavior suggests that is important to define a model of management of communication file system level guarantee the smaller loss in the transfer. The falls points identified in the transfer analysis allows to know the performance limits of the applications that can be executed in the infrastructure will have to deal with. In the tests developed in this work, it's observable these limitations.

Our future activities include further fine tuning of the hash-based coherence mechanism, in order to try to extract some more bits of performance, and mainly the inclusion of fault-tolerance characteristics in dNFSp. We intend to achieve a more dynamic environment which might be better suitable to a resource-harvesting system within a local network or even to a grid computing scenario.

Also, activities are planned for the evaluation of performance context. It will make another tests and measurements with nodes of different clusters from grid5000 facility, in order to characterize the different behaviors in different architectures and platforms and to experiment the behavior in of dNFSp in the grid environment.

8 Acknowledgements

The authors express acknowledgments to the program CAPES/COFECUB (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior/Comité Français d'Evaluation de la Coopération Universitaire et Scientifique avec le Brésil) and to CNPQ (Conselho Nacional de Desenvolvimento Científico e Tecnológico)

References

[1] Callaghan, B., Pawlowski, B., Staubach, P.: NFS Version 3 Protocol Specification: RFC 1831. Internet Engineering Task Force, Network Working Group. (1995)

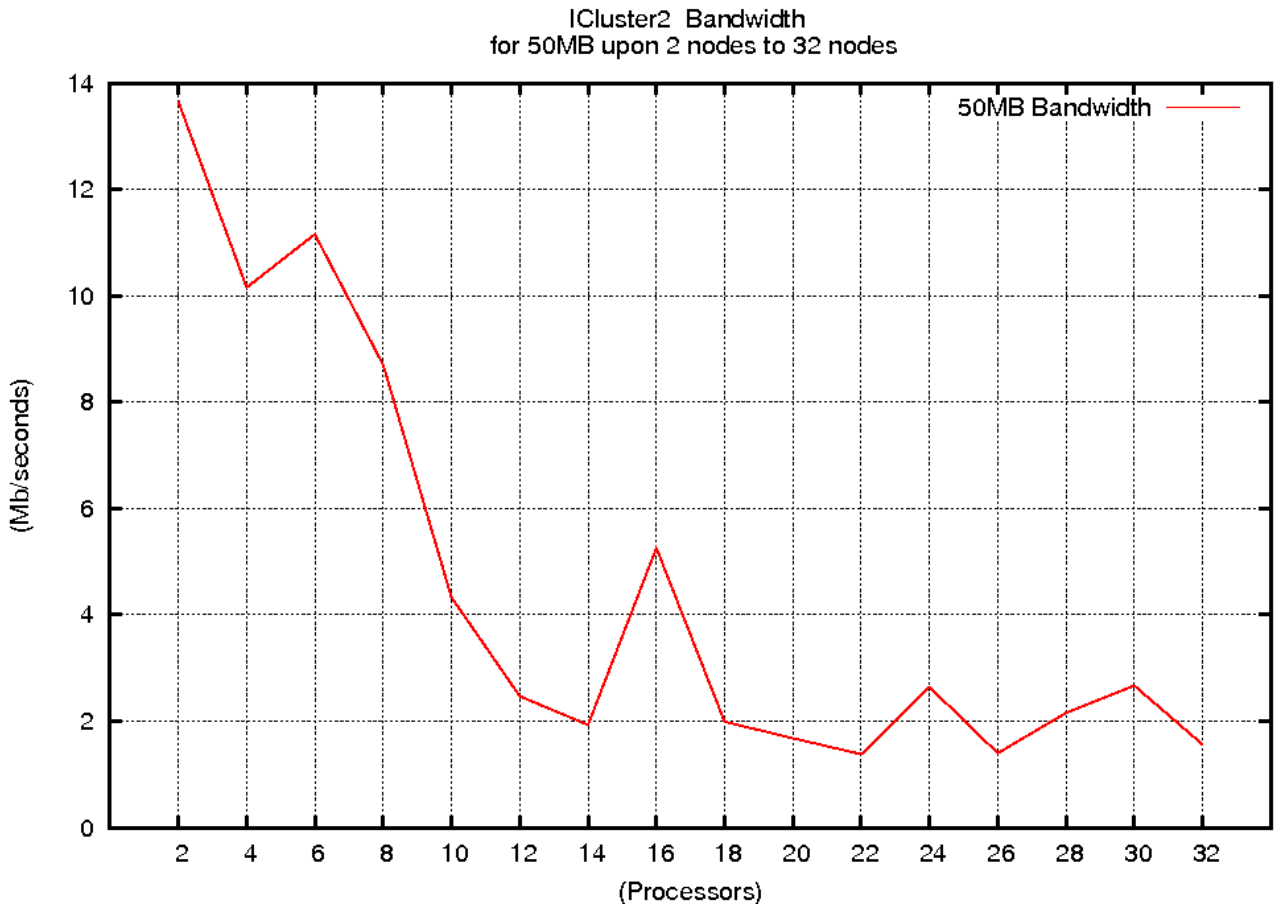

Figure 5. ICluster-2 Bandwidth in Transfer of 50MB

[2] Ávila, R.B.: Uma Proposta de Distribuição do Servidor de Arquivos em Clusters. Tese de doutorado em ciência da computação, Universidade Federal do Rio Grande do Sul, Brasil (2005)

[3] Lombard, P., Denneulin, Y.: nfsp: a distributed NFS server for clusters of workstations. In: Proc. of the 16th International Parallel & Distributed Processing Symposium, IPDPS, Ft. Lauderdale, Florida, USA, Los Alamitos, IEEE Computer Society (2002) 35 Abstract only, full paper available in CD-ROM.

[4] Ávila, R.B., Navaux, P.O.A., Lombard, P., Lebre, A., Denneulin, Y.: Performance evaluation of a prototype distributed NFS server. In Gaudiot, J.L., Pilla, M.L., Navaux, P.O.A., Song, S.W., eds.: Proceedings of the 16th Symposium on Computer Architecture and High-Performance Computing, Foz do Iguaçu, Brazil, Washington, IEEE (2004) 100–105

[5] Corbett, P.F., Feitelson, D.G.: The vesta parallel file system. ACM Trans. Comput. Syst. **14**(3) (1996) 225–264

[6] Corbett, P.F., Baylor, S.J., Feitelson, D.G.: Overview of the vesta parallel file system. SIGARCH Comput. Archit. News **21**(5) (1993) 7–14

[7] Garcia-Carballeira, F., Calderon, A., Carretero, J., Fernandez, J., Perez, J.M.: The design of the expand parallel file system. International Journal of High Performance Computing Applications **17**(1) (2003) 21–37

[8] : i-cluster 2 (2006) Available at: <http://i-cluster2.inrialpes.fr>. Access in: May 2006.

[9] Hildebrand, D., Ward, L., Honeyman, P.: Large files, small writes, and pnfs. In: ICS '06: Proceedings of the 20th annual international conference on Supercomputing, New York, NY, USA, ACM Press (2006) 116–124

[10] Kielmann, T., Bal, H.E., Verstoep, K.: Fast measurement of LogP parameters for message passing

platforms. Lecture Notes in Computer Science **1800** (2000) 1176–??

[11] Alexandrov, A., Ionescu, M.F., Schauser, K.E., Scheiman, C.: Loggp: incorporating long messages into the logp model one step closer towards a realistic model for parallel computation. In: SPAA '95: Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures, New York, NY, USA, ACM Press (1995) 95–105

[12] Howard, J.H., Kazar, M.L., Menees, S.G., Nichols, D.A., Satyanarayanan, M., Sidebotham, R.N., West, M.J.: Scale and performance in a distributed file system. ACM Trans. Comput. Syst. **6**(1) (1988) 51–81

[13] Braam, P.J.: The coda distributed file system. Linux J. **1998**(50es) (1998) 6

[14] Rosenblum, M., Ousterhout, J.K.: The design and implementation of a log-structured file system. ACM Transactions on Computer Systems **10**(1) (1992) 26–52

[15] Levy, E., Silberschatz, A.: Distributed file systems: concepts and examples. ACM Comput. Surv. **22**(4) (1990) 321–374

[16] Carns, P.H., Ligon III, W.B., Ross, R.B., Thakur, R.: PVFS: A parallel file system for linux clusters. In: Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, GA, USENIX Association (2000) 317–327

[17] Brandt, S.A., Miller, E.L., Long, D.D.E., Xue, L.: Efficient metadata management in large distributed storage systems. In: Proceedings of the 20 th IEEE/11 th NASA Goddard Conference on Mass Storage Systems and Technologies (MSS'03), Washington, DC, USA, IEEE Computer Society (2003) 290