Journal of Computer Science 1 (3) : 363-368, 2005
ISSN 1549-3636
© Science Publications, 2005

Towards a Scalable File System on Computer Clusters Using Declustering

Vu Anh Nguyen, Samuel Pierre and Dougoukolo Konaré
Department of Computer Engineering, Ecole Polytechnique de Montreal
C.P. 6079, succ. Centre-ville, Montreal, Quebec, H3C 3A7 Canada

Abstract : This study addresses the scalability issues involving file systems as critical components of computer clusters, especially for commercial applications. Given that wide striping is an effective means of achieving scalability as it warrants good load balancing and allows node cooperation, we choose to implement a new data distribution scheme in order to achieve the scalability of computer clusters. We suggest combining both wide striping and replication techniques using a new data distribution technique based on “chained declustering”. Thus, we suggest a complete architecture, using a cluster of clusters, whose performance is not limited by the network and can be adjusted with one-node precision. In addition, update costs are limited as it is not necessary to redistribute data on the existing nodes every time the system is expanded. The simulations indicate that our data distribution technique and our read algorithm balance the load equally amongst all the nodes of the original cluster and the additional ones. Therefore, the scalability of the system is close to the ideal scenario: once the size of the original cluster is well defined, the total number of nodes in the system is no longer limited, and the performance increases linearly.

Key words: Computer Cluster, Scalability, File System

INTRODUCTION

Large-scale telecommunication and the Internet consistently require enhanced processing capacities. Parallel systems have been used for a long time to respond to this challenge. Internet proxy servers are a crucial part of the Web infrastructure as they absorb some of the growth of the demand by caching data. In this context, the performance scalability of the parallel system used is very important: such systems must support an increasing number of requests while the number of clients increases as well.

Most of the prior parallel system research addresses scientific applications, thus, complete scalability theories and metrics such as *iso-speed* or *iso-efficiency* have already been defined [1, 2]. However, these pioneer studies are mainly focus on the algorithmic aspects of scalability. Recent research has addressed the architecture of scalable parallel systems [3]. In this context, data storage and access are critical and often limit scalability.

This study specifically addresses distributed file systems for computer clusters. A file system performance can be measured by its maximum bandwidth while clients access files. Depending on the application used, there is a minimum level of quality of service for each client. For applications such as video-on-demand, real-time constraints are critical: system performance is measured by the number of clients who can be served simultaneously.

File System Characteristics: Internet proxy servers or multimedia file servers must be scalable in order to serve an increasing number of clients. For a file system, different cases of scalability exist: scalability of the number of clients simultaneously accessing an increasing quantity of data; scalability of the quantity of data accessed by the same number of clients; scalability of the number of clients simultaneously accessing the same quantity of data. Cases where the service capacity of the file system must be expanded are the most complex, whereas increasing storage capacity is easier since this operation consists simply of adding disks to each node.

Such systems require redundancy in order to overcome failures and they also need scalability in order to reduce service time. In order to improve scalability through the use of a new data distribution scheme for computer clusters, we chose to work with the Parallel Virtual File System (PVFS) a file access mechanism, developed at Clemson University [4].

This Linux distributed file system implements wide striping and it has been used extensively in RAID architecture [5]. As shown in Fig. 1, it breaks down files into small blocks of data (PVFS has a default block size of 64 KB) and distributes these blocks throughout a disk array. When accessing a file, all of the disks cooperate to provide a better level of performance than a single disk operating on its own. The difference can be observed within a computer cluster. The data is transmitted through a local network and wide striping is implemented at the software level. Blocks are distributed sequentially on the disk array and the load is thus implicitly balanced. This property allows the wide striping architecture to achieve respectable performance scalability [3]. However, each time a disk is added to the array, all of the data must be redistributed. Secondly, although this is a very important aspect for file systems dedicated to a commercial environment, PVFS has yet to provide a redundancy or fault-tolerance mechanism. This study attempts to overcome the redundancy mechanism and enhance the scalability of a computer cluster by introducing a new data distribution scheme.

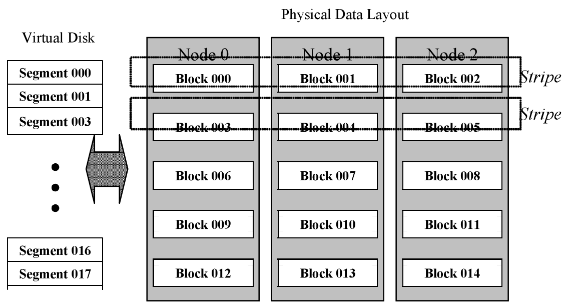


Fig. 1: Wide Striping

In a computer cluster using PVFS, nodes can have three different functions: as shown in Fig. 2, a given node can be an I/O node, the management daemon or a client node. An I/O node is actually a node which stores data blocks.

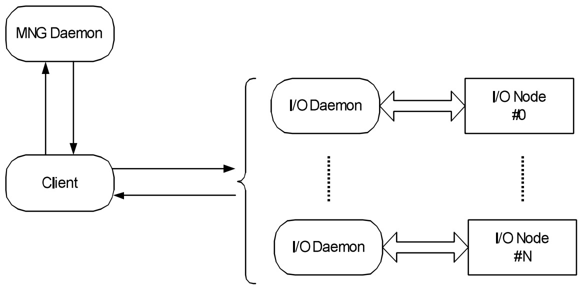


Fig. 2: File Access Mechanisms

A single management daemon node is dedicated to storing the file system metadata (physical location of files, number of nodes, etc). One of the objectives of a future version of PVFS aims to permit multiple management daemons in the same cluster, in order to avoid the bottleneck of a unique management daemon and provide redundancy within the system. Although client nodes are the entities which access the files, they are not necessarily the end clients. For example, the end client could be the portal through which a customer accesses data. Each physical node of the computer cluster can actually perform more than one of these functions. For the purpose of this investigation, a single function is assigned to each node.

ACHIEVING SCALABILTY WITH DECLUSTERING TECHNIQUES

Double Declustering: A few years ago, the performance of a file system composed of a cluster of computers would have been limited by the network throughput. Nowadays, with the increasing speed of networks such as gigabit Ethernet, the file system is likely to be the bottleneck. On the other hand, other types of networks (wireless networks using node mobility) such as PAN (Personnal Area Network) or MANET (Mobile Ad- hoc Network), tend to onfirm that the network is the bottleneck Indeed, such network throughputs are defined in terms of megabits when the node mobility is low and kilobits when the mobility is high. The objective of this study is to overcome file system scalability in any type of network when the load is high in read mode such as internet proxies. The kinds of applications targeted are the ones that saturate the file system and the networks for a long period of time. Some examples of applications are heavy multimedia applications, and high client loads such as the one that can shut down a web server in response to too many requests from clients.

To achieve these goals, we suggest the use of wide striping and replication in the same file system thanks to declustering techniques. In order to remain in the "linear scalability" area of the computer cluster. Linear scalability occurs when the addition of new cluster nodes does not affect file system performance. The concept of declustering regroups the techniques of data block distribution through a disk array. These distributions are traditionally used to provide better fault tolerance or to improve the quality of service [6, 7]. It has already been shown that wide striping offers limited scalability [8]. We thus aim to use declustering techniques to enhance scalability by using clusters of clusters. Thus, our system is based on double declustering: one for data redundancy and one for scalability. The combination of these two features is very important in systems such as Internet proxies.

The only solution to increase the performance of this system is to add new nodes to the original cluster until the performance of the system starts to plummet with more demanding requests. With simple wide striping, the addition of new nodes causes a complete redistribution of the data blocks in order to keep the load balanced. We plan to replicate the existing data blocks on new nodes in the same cluster while retaining considerable load balancing among all of the nodes. The resulting system contains two kinds of nodes: initial nodes and additional nodes from the initial cluster. The size *N* of the initial cluster is determined by the performance of the network. The number *M* of additional nodes ranges from 1 to *N*. Additional nodes are added to a network that is independent of the original cluster.

When the number of additional nodes equals the number of nodes in the initial cluster, the nodes are separated by type to form a distinct cluster, as illustrated in Fig. 3. Finally, the complete architecture is redesigned, using a cluster of clusters, so the network does not limit the performance anymore and load balancing is achieved. Hence, every time the system is separated into two independent clusters, the resulting performance equals the sum of the performances of each independent cluster. Thus, the size of each cluster is limited so that it remains within the "linear scalability" area. In addition, an algorithm that dispatches the clients to each distinct cluster is developed in order to balance the load.

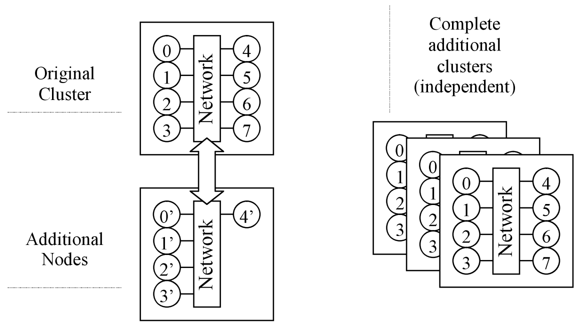


Fig. 3: System Architecture

Description of the Data Distribution: The declustering technique used to provide fault tolerance, an extension of the chained declustering technique [6], is called "multi-chained declustering". This technique strives to equally distribute the redundancy blocks of one node to all of the other nodes. Thus, if a certain node becomes unavailable, all of the remaining nodes support their load without any load imbalance. On the other hand, redundancy schemes such as chained declustering do not warranty good load balancing if a node becomes unavailable. Our second declustering technique seeks to keep the load balanced when new nodes are added to the cluster. The principle is to replicate on the additional node the same number of data blocks from each node of the initial cluster.

A data block can be located in the cluster by its coordinates (*i,j*) of which *i* is the node number and *j* the stripe number. Nodes from the original cluster are numbered from 0 to *N*-1, and additional nodes from 0 to *M*-1. In our system, a maximum of four copies of the same data block can exist within a cluster. To differentiate between the different types of blocks, we use the following notations:

* X denotes a block of primary data located in a node of the original cluster.
* Y denotes a block of redundant data located in a node of the original cluster.
* U denotes a block of primary data located in an additional node.
* V denotes a block of redundant data located in an additional node.

These distributions are transparent to the user of the file system who accesses a virtual disk. On this virtual disk, a data block exists only once and it is identified by the letter *k*:

* *Xk* (*Yk, Uk, Vk* respectively) denotes the coordinates of the block number *k* of type *X* (*Y, U, V* respectively). Blocks are read from left to right and top to bottom within the same cluster. k represents the block number of the same cluster.
* *x(i,j) (y(i,j)* respectively) denotes the block of type *X* (*Y* respectively) which is located on stripe *i* and node *j* from the original cluster;
* *u(i,j)* (*v(i,j)* respectively) denotes the block of type *U* (*V* respectively) which is located on stripe *i* and the additional node *j*.

Using these notations, we have:

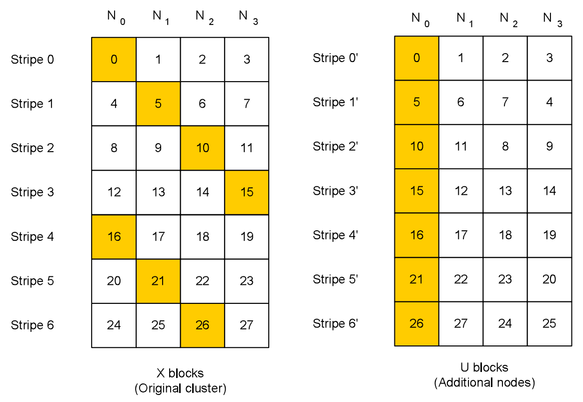
$$x(i, j) = x(X_k) \bullet \quad (i, j) = X_k \quad \text{, with } 0 \bullet i \quad \text{and} \quad 0 \bullet j \bullet (N \bullet 1)$$

We can now analytically formulate the declustering techniques used in our file system. *N* is the number of nodes in the original cluster and *M* is the number of additional nodes.

Our declustering technique, combined with a read algorithm based on the previous equations, guarantees that the average numbers of blocks read are the same for each node. Since network links have all the same maximum capacity, this system provides a theoretically perfect load balancing of each group of *N*(*N*+*M*) blocks. Note some of its simple and static qualities: the only parameters are the number of nodes in the original cluster *N* and the number of additional nodes *M.* The simplicity of this algorithm warranties good scalability and easy implementation. Once provided with the block number to be accessed by the virtual disk *k*, this algorithm decides to retrieve a type X or U block. The amount of blocks to be read in order to have an ideal load balancing is not a constraint since the number of data blocks in a file system is very high compared to this threshold.

Table 1: Data Distribution Formula

Type X Data Blocks (Original Cluster Nodes, Primary Data)	Type Y Data Blocks (Original Cluster Nodes, Redundant Data)
$X_k = \left(\frac{k \bullet k[N]}{N}, k[N]\right)$ $x(i,j) = x(X_k) \Rightarrow k = i \bullet N + j$	$Y_k = \left(\frac{k \bullet k[N]}{N}, \left\{k[N] + \left(\frac{k \bullet k[N]}{k}\right)[N \bullet 1] + 1\right\}[N]\right)$ $y(i,j) = y(Y_k) \Rightarrow \begin{cases} k = i \bullet N + j \bullet (i[N \bullet 1] + 1) if\ j \bullet i[N \bullet 1] + 1 \\ k = i \bullet N + j \bullet (i[N \bullet 1] + 1) + N if\ j \bullet i[N \bullet 1] \end{cases}$
Type U Data Blocks (Additional Nodes, Primary Data)	Type V Data Blocks (Additional Nodes, Redundant Data)
$U_k = \left(\frac{k \bullet k[N]}{N}, \left\{k[N] \bullet \left(\frac{k \bullet k[N]}{k}\right)[N]\right\}[N]\right)$ $u(i,j) = u(U_k) \Rightarrow \begin{cases} k = i \bullet N + j + i[N] if\ j \bullet N \bullet i[N] \bullet 1 \\ k = i \bullet N + j + i[N] \bullet N if\ j \bullet N \bullet i[N] \end{cases}$	$V_k = \left(\frac{k \bullet k[N]}{N}, \left\langle \left\{k[N] \bullet \left(\frac{k \bullet k[N]}{k}\right)[N]\right\}[N] + \left(\frac{k \bullet k[N]}{k}\right)[N \bullet 1] + 1 \right\rangle [N]\right)$ $k = i \bullet N + j + i[N] \bullet (i[N \bullet 1] + 1)$ $if\quad i[N \bullet 1] + 1 \bullet i[N] \bullet j \bullet N + i[N \bullet 1] \bullet i[N]$ $k = i \bullet N + j + i[N] \bullet (i[N \bullet 1] + 1 + N)$ $if\quad i[N \bullet 1] + 1 + N \bullet i[N] \bullet j \bullet 2N + i[N \bullet 1] \bullet i[N]$ $k = i \bullet N + j + i[N] + (N \bullet 1) \bullet (i[N \bullet 1] + 1)$ $if\quad i[N \bullet 1] \bullet (N \bullet 1) \bullet i[N] \bullet j \bullet i[N \bullet 1] \bullet i[N]$ $k = i \bullet N + j + i[N] \bullet (i[N \bullet 1] + 1)$ $if\quad i[N \bullet 1] + 1 \bullet i[N] \bullet j \bullet N + i[N \bullet 1] \bullet i[N]$



	N_0	N_1	N_2	N_3
Stripe 0	0	1	2	3
Stripe 1	4	5	6	7
Stripe 2	8	9	10	11
Stripe 3	12	13	14	15
Stripe 4	16	17	18	19
Stripe 5	20	21	22	23
Stripe 6	24	25	26	27

X blocks
(Original cluster)

	N_0	N_1	N_2	N_3
Stripe 0'	0	1	2	3
Stripe 1'	5	6	7	4
Stripe 2'	10	11	8	9
Stripe 3'	15	12	13	14
Stripe 4'	16	17	18	19
Stripe 5'	21	22	23	20
Stripe 6'	26	27	24	25

U blocks
(Additional nodes)

Fig. 4: Extended Multi-chained Declustering of Four Nodes

SIMULATION AND VALIDATION

In order to validate our file system, we simulate our model using the C-SIM library [9] and the results were compared to other distributed file systems which allow for redundancy and scalability.

Simulation Results : Figure 5 shows the scalability of an 8-node cluster when the number of additional nodes varies from 0 to 8. The maximum bandwidth delivered was measured by the file system and compared to the original cluster. The results show that the performance is close to the ideal case: the degradation rate remains below 1%. Once the number of additional nodes is equal to the number of nodes in the original cluster, the size doubles and the system can be split into two independent clusters: due to our data distribution technique, the cluster formed of additional nodes is almost identical to the original one and provides the same redundancy schemes (multi-chained declustering).

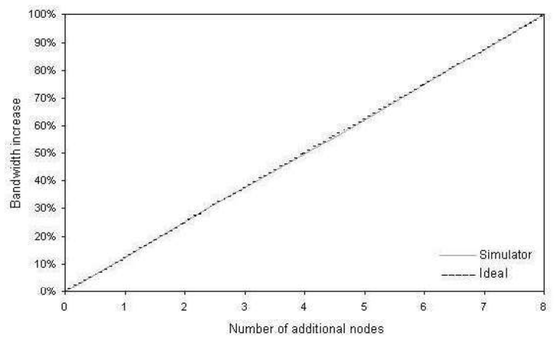


Fig. 5: Scalability of an 8-node Cluster System

Since the number of nodes in a cluster can double, the size of the original cluster must be selected judiciously according to the performance of the network. As shown in Fig. 6, if the original cluster is too large, the system will not remain within the "linear scalability area" due to network congestion. In the configuration used in this experiment, the optimal size for the original cluster turned out to be four nodes. If the size of the original cluster is small, more clusters are necessary to build a large system and cluster load balancing reduces performance.

Update Costs: One of the goals of this file system aims to allow for scalability without service interruption and with limited perturbation during system updates. When a node is added to the system, it must retrieve the data blocks which are stored locally. Since additional nodes

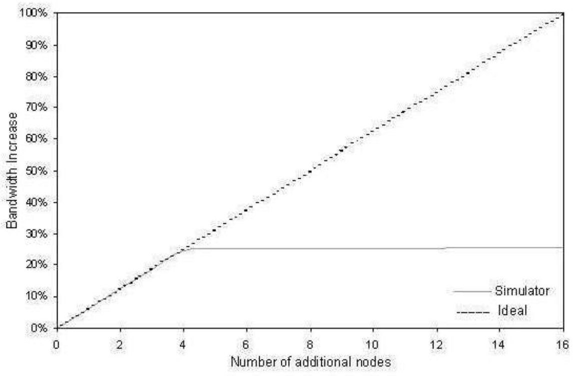


Fig. 6: Scalability of a 16-node Cluster System

also store redundant data, the number of blocks to be retrieved equals the quantity of blocks within an original node. Such a declustering technique distributes this load amongst all of the cluster nodes, including the additional nodes that are already online. A system upgrade was simulated in order to measure the perturbation caused by the introduction of new nodes.

We simulated the addition of the first two nodes in a 10-node cluster with the following parameters:

* Twenty-five clients always read 20 MB files simultaneously.
* The file system contains 1500 files of 20 MB (30 GB in total). Each node stores 3 GB of data. The size of the data blocks is 64 KB.
* The first and second additional nodes are added at t=100 s and t=1500 s. Thus, the second node is added after the first finishes its update and is online. Each node has to update 6 GB of data (about 93,750 blocks).
* To update blocks which are stored locally, the additional nodes behave as clients: they have the same priority in FIFO queues as regular clients.

Figure 7 presents the simulation results of this scenario. We notice that the update of an additional node consumes a bandwidth which is equivalent to that of one client in the system.

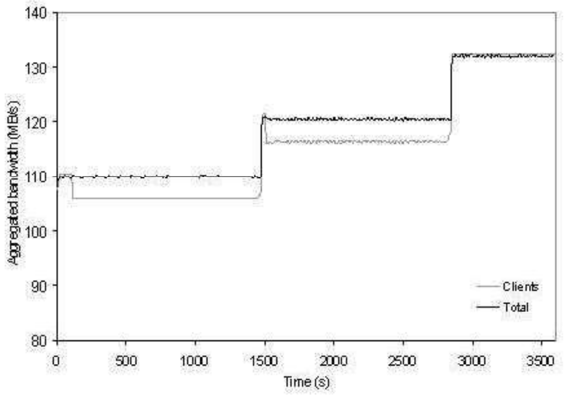


Fig. 7: Additional Online Node Update Scenario

Once the data is updated and the node appears online, the performance of the whole system increases proportionally to the number of clusters of nodes. It is possible to reduce the perturbation caused by the update: there is a trade-off between the time for a node to update its data and the bandwidth used to conduct this operation.

CONCLUSION

This study addressed the scalability issues of file systems for computer clusters. Using the C-SIM library, a parallel file system was modeled and a data distribution technique with declustering was simulated. Our primary analysis shows that wide striping is an efficient way to achieve scalability as it warrants good load balancing and allows nodes to cooperate. Unlike PVFS, which requires a redistribution of the data blocks among all the nodes (a very costly operation, inappropriate for continuous service) when a new node is added, this data distribution technique simply warranties considerable load balancing amongst all of the nodes.

Since the cluster size is limited by the network and sometimes IO node maximum capacity, a cluster of clusters was used, where each cluster remains in the "linear scalability" area. Traditionally used to provide fault tolerance, declustering techniques regroup ways to distribute redundancy blocks in a file system. A first declustering technique called "multi-chained declustering" is used to provide data redundancy and better load balancing when a node is down. In addition, a second declustering level can be performed in order to combine wide striping and replication and provide real scalability. Called "extended multi-chained declustering", a particular declustering technique was developed to take advantage of both techniques. This solution allows for the addition of nodes to the system without reconfiguring existing nodes while keeping the load balanced. In this study, the two declustering techniques used were formalized.

Using our declustering techniques, nodes can be added one by one to the original cluster while the performance borders the ideal case scenario. As soon as the number of additional nodes equals the size of the original cluster, additional nodes are separated from the others to form a new independent cluster. A previous simulator was used to implement and validate this architecture. Simulation results indicate that the system has a linear scalability once the number of nodes in the original cluster is defined according to the network performance. Finally, simulation scenarios were used to show that scaling costs are inexpensive: when a node is added to the cluster, no redistribution of the data blocks on the existing nodes is needed to keep the load balanced.

Future works involve studying the write performance of our file system. It could be interesting to investigate different causes of failure and the optimization that could be performed for each case. Thus, it could be interesting for further studies to implement a prototype of the developed architecture. Other optimizations could also be tested such as the use of caches or dynamic load balancing algorithms, especially if the way data is accessed has known characteristics [10].

ACKNOWLEDGMENTS

This study was supported in part by Ericsson Research Canada and the Natural Sciences and Engineering Research Council (NSERC) of Canada.

REFERENCES

1. Grama, A., A. Gupta and V. Kumar, 1993. Isoefficiency Function: A Scalability Metric for Parallel Algorithms and Architectures, IEEE Parallel and Distributed Technology. Special Issue on Parallel and Distributed Systems: From Theory to Practice, 1: 12-21.
2. Li, K. and X. Sun, 1998. Average-Case Analysis of Isospeed Scalability of Parallel Computations on Multiprocessors. State University of New York at New Paltz, Department of Mathematics and Computer Science, Technical Report pp: 98-108
3. Buyya, R. (ed.), 1999. High Performance Cluster Computing: Architectures and Systems. Prentice Hall PTR
4. Carns, P. H., W. B. Ligon III, R. B. Ross and R. Thakur, 2000. PVFS: A Parallel File System For Linux Clusters. Proceedings of the 4th Annual Linux Showcase and Conference, 317-327.
5. Massiglia, P., 2000. RAID for Enterprise Computing. A Technology White Paper from VERITAS Software Corporation
6. Arpaci-Dusseau, R. H., E. Anderson, N. Treuhaft, D. E. Culler, J. M. Hellerstein, D. A. Patterson and K. Yelick 1999. Cluster I/O with River: Making the Fast Case Common, Sixth Workshop on I/O in Parallel and Distributed Systems
7. Hsiao, H., D. J. Dewitt, 1990. Chained Declustering: A new availability strategy for multiprocessor database machines. Proceedings of the 6th Intl Conf. Data Engineering, 456-465.
8. Chou, C., L. Golubchik and J.C.S. Lui, 1999. Striping Doesn't Scale: How to Achieve Scalability. Technical Report, CS-TR-1999-03, University of Maryland
9. Mesquite CSIM 18 - A Development Toolkit for Simulation and Modeling. http://www.mesquite.com/csim18page.htm
10. Reisslein, M., F. Hartanto, and K. W. Ross, 1999. Interactive video streaming with proxy servers, Technical Report, GMD FOKUS Institute Eurecom. http://www.fokus.gmd.de/usr/reisslein