Cost Analysis of a Distributed Video Storage System

Alice Bonhomme

Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON n° 5668

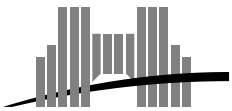
CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

Cost Analysis of a Distributed Video Storage System

Alice Bonhomme

June 2001

Research Report N° 2001-23



École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : lip@ens-lyon.fr

INRIA

Cost Analysis of a Distributed Video Storage System

Alice Bonhomme

June 2001

Abstract

We describe the design and the implementation of the CFS (Cluster File System) storage system which is dedicated to video streams. Our goal is to provide a system with the following features: 1) High number of supported streams at a low cost. 2) Transparent management with respect to the clients. 3) Reliability regarding data storage and service continuity. The CFS is implemented on a cluster of PCs connected with a high speed internal network. Its management is fully distributed among the cluster nodes so that there is no central component. Data are stored and retrieved among the cluster nodes using a Streaming RAID strategy, to enhance reliability.
We evaluate the overhead introduced by the CFS management with respect to disk performance, by varying the number of nodes involved in file storage, the block size and the reliability level. The impact of the global server load on this overhead is moreover discussed to demonstrate scalability. As matter of fact, the overhead is between 3–15% depending on the configuration.

Keywords: Video server, Data striping, Reliability, Performance analysis

Résumé

Nous présentons la conception et l'implémentation du CFS (Cluster File System), un système de stockage dédié aux données vidéo. Notre objectif est de procurer un système intégrant les caractéristiques suivantes : 1) Grand nombre de flux supportés, pour un faible coût. 2) Gestion transparente vis à vis des clients. 3) Fiabilité des données et de la continuité du service. Le CFS est implémenté sur une grappe de PC connectés avec un réseau haut débit. Sa gestion est totalement distribuée entre les nœuds de la grappe. Il n'y a donc pas de composant central. Afin de gèrer la fiabilité du système, les données sont stockées et accèdées sur les nœuds de la grappe d'après la stratégie Streaming RAID.
Nous évaluons le surcoût introduit par la gestion du CFS par rapport aux performances des disques, en variant le nombre de nœuds participant au stockage des données, la taille de bloc, et le niveau de fiabilité. L'influence de la charge globale du serveur sur ce surcoût, est aussi discutée afin de montrer les capacités d'extensibilité du CFS. De manière générale, le surcoût du CFS représente entre 3–15%, suivant la configuration.

Mots-clés: Serveur Vidéo, Distribution de données, Tolérance aux pannes, Analyse de performance

1 Introduction

Many multimedia applications, such as Video-On-Demand, or interactive television, require a video server capable of supporting hundreds of simultaneous clients. Due to the number of large objects stored and the real-time requirements for their retrieval, the design of video storage servers significantly differs from that of traditional high capacity data storage servers. Database servers, for example, are designed to optimize the number of requests per second, and to allow a fast response to the clients. In contrast, video storage servers must meet the requirements resulting from the continuous nature of the stored multimedia streams, and must guarantee the delivery of video data at precise time according to the bitrate of stored streams.

In this paper, we introduce the design of the CFS (Cluster File System), an efficient storage system dedicated to video streams. Our goal is to provide a system with the following features : 1) High number of supported streams at a low cost. 2) Transparent management with respect to the clients. 3) Reliability regarding data storage and service continuity.

It has been recognized for several years that distributed architectures based on COTS components are best suited for low cost servers [TMDV96]. Techniques [ORS96, GB99a] have been proposed to efficiently manage video data on such distributed clusters of nodes. Video streams are spread across the nodes using a variety of schemes, called *data striping*, similar to the well known RAID approach. Reliability can be moreover obtained by using redundant storage. Among the existing storage techniques to enhance reliability [SV00, GLP98], we can point out the Streaming RAID approach [TPBG93], which enables to support the failure of a node without loss in the number of supported output streams.

Such distributed storage architectures require an additional functional component to manage the relationship between the client and the various storage nodes. It is called a *meta-server*. Clients are connected to the meta-server through an external *distribution network*. We do not assume any specific property for this external distribution network, except that it is probably slower than the internal cluster network by one or two orders of magnitude. Several designs have been proposed for the meta-server. In the *centralized*

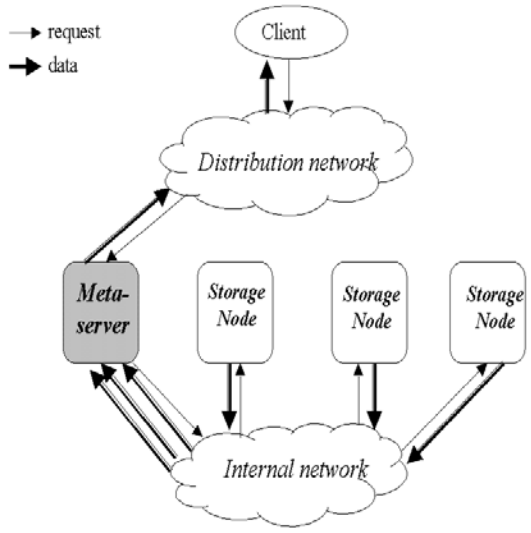


FIG. 1: *The centralized design*

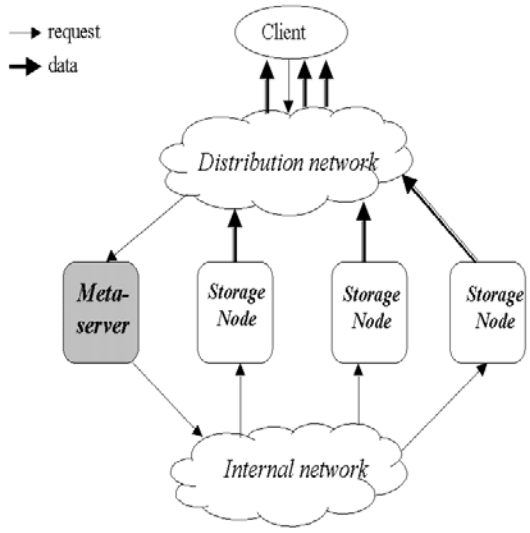


FIG. 2: *The semi-centralized design*

design (Figure 1), the meta-server component is implemented with an additional node, which is in charge of handling the whole client's request. It forwards the request to the storage nodes, receives the data from these nodes, and then sends the data back to the clients. However, the meta-server stands as a bottleneck for communication between the client and the storage nodes, which hinders it from scaling. Therefore, alternative designs have been proposed. In the *semi-centralized design* (Figure 2), the meta-server is also implemented

as an additional node, but data are directly sent from the storage nodes back to the client, in a scalable way. However, this design is not transparent with respect to the clients, as a client must be *explicitly* connected to all the storage nodes in order to get the data. Moreover, it is up to the client to merge the various data blocks together, in order to rebuild the original data. In particular, the client is in charge of implementing the reliability strategy, if any. The *distributed design* (Figure 3) aims at providing transparency together with

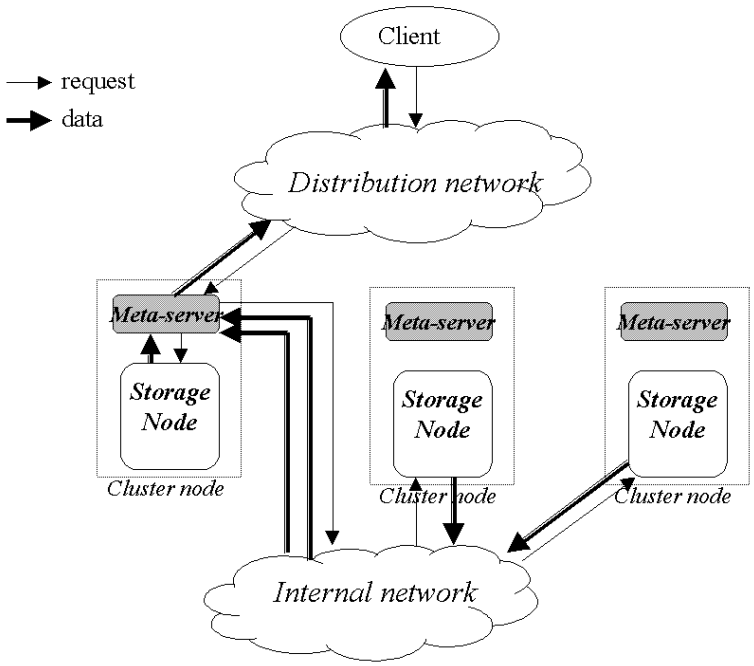


FIG. 3: *The distributed design*

scalability. The meta-server functionality is distributed among the storage nodes instead of being centralized at a single one. The client connects to any single meta-server component which is in charge of gathering data from storage nodes before forwarding them to the client. This approach enhances transparency as the client knows of only one meta-server which is in charge of all data-rebuilding activity. Successive clients can connect to different meta-servers, so that scalability and load-balancing are enhanced. In the event of a meta-server failure, only the clients connected to it are disturbed.

The Cluster File System (CFS) described in this paper is made up of a cluster of COTS components, namely PCs with large disks, interconnected by a Gigabit Myrinet network. In accordance with the distributed design, each node hosts both a meta-server and a storage component. Data are stored and retrieved among the storage components using a Streaming RAID strategy. As discussed above, this yields a video storage system with the expected properties.

In this paper, we focus on evaluating the performance of various aspects of our CFS proposal. With respect to the centralized design, CFS enhances scalability by using a distributed meta-server. This induces additional management tasks to keep all meta-servers components in consistent states. We study the overhead of this choice in terms of CPU load. With respect to the semi-centralized design, CFS enhances transparency by gathering data at the meta-server component before forwarding them to the clients. This induces an additional message exchanges through the internal network on the critical execution path. We study this overhead in terms of latency and bandwidth.

Our experimental protocol focuses on read requests as they make up the dominant cost in common video applications. Also, we disregard data transfer between the CFS and the client as it strongly depends on the characteristics of the external network. Only intra-cluster communication is considered here. The latency of a read request is made up of three contributions : 1) Parallel disk IO to retrieve data blocks ; 2) Communication

time to gather individual data blocks at the meta-server component ; 3) Computation time to receive the series of blocks, and build up the read request out of them. We define the overhead of our CFS proposal as contributions 2) and 3) with respect to the sum of all three contributions. We provide a detailed study of this overhead by varying the number of nodes involved in storing files, the block size and the selected reliability level. The evolution of the overhead with respect to the global server load is moreover discussed to demonstrate scalability. As matter of fact, the overhead is between 3–15% depending of the configuration.

The rest of the paper is organized as follows : Section 2 describes the design of the CFS (cluster of PCs, Streaming RAID strategy, distributed design) and its implementation. Section 3 outlines the methodology we use to evaluate the CFS cost, and presents the various test parameters. Section 4 reports experimental results, and Section 5 discusses some related work.

2 Design of the CFS

In this section, we present the design and the implementation of the CFS. The basic interface of the CFS is similar to that of most high performance file systems with asynchronous reading and writing functionalities beside the regular ones : open, close, etc. The CFS interface allows the user to create and write a video file with or without reliability. A *reliable* file remains accessible even after the failure of one single node, whereas no guarantee is provided for a non-reliable file. In this paper, we consider both types of files.

2.1 Data striping

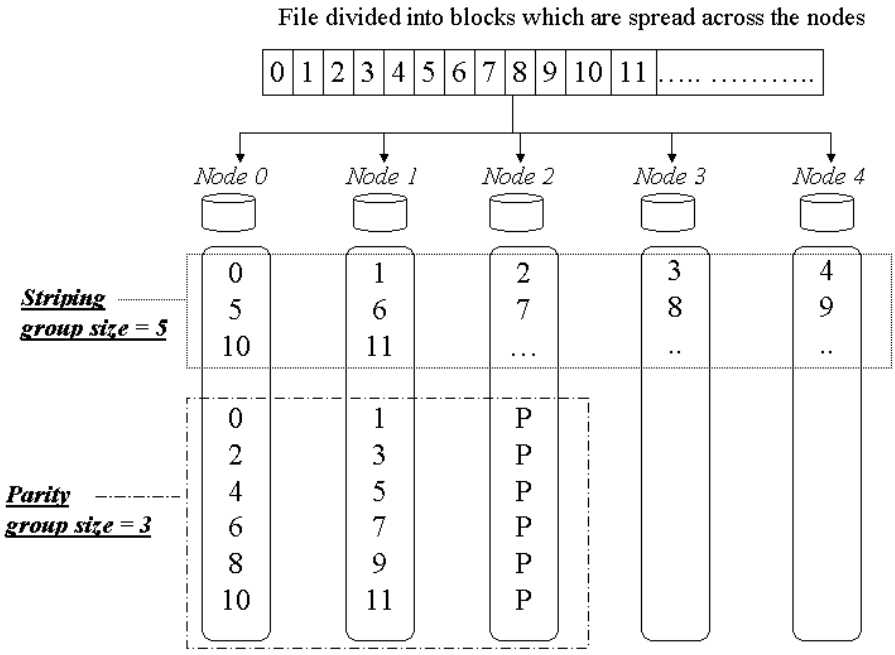


FIG. 4: *Two examples of data striping using a striping group size equal to 5 or a parity group size equal to 3*

Video data are distributed according to a *striping strategy*. A video file is divided into blocks of equal size, called *block size*. These blocks are then distributed among cluster nodes or a subset of cluster nodes. The nodes involved in the file storage define a *striping group*. The number of nodes involved in a striping group is called *striping group size*. Furthermore, in order to support at most one node failure, reliable files

use additional *parity blocks*. For a striping group of size P, one finds one parity block for $(P-1)$ data blocks. This additional block is built as the result of a XOR operation between the regular data blocks. This technique is similar to the well-known RAID5 [PGS88] technique applied to a multi-node platform, instead of a multi-disk one. For reliable files, striping groups are also called *parity groups*. Figure 4 illustrates striping and parity groups. On creating a file, the user specifies the striping group size, the block size and the reliability level. This information is stored at the meta-server. It is used each time the file is read.

2.2 Data retrieval

In most of retrieval schemes found in the literature, data blocks are retrieved one after the other, at the time they are requested. However, on the occurrence of a node failure, a delay is required to get the parity block and the other data blocks necessary to perform the XOR operation. Retrieving these blocks consumes extra disk bandwidth. If this unexpected consumption has not been taken into account in scheduling the client requests, then this results in a slow down for the overall system. In order to avoid this time and bandwidth overhead, a specific scheme [TPBG93] called Streaming RAID (SR), has been proposed for a multi-disk video server. SR exploits the specificity of video files, as they are usually accessed in a sequential way : it is extremely likely to retrieve block $(i+1)$ after block i. Instead of getting blocks one after the other, SR gets all the blocks of a striping group at the same time, including the parity block. Thus, all the blocks necessary to the XOR operation are already available and, the only remaining overhead is the XOR computation. SR consumes an extra bandwidth to retrieve the parity block in all cases. Therefore, the behavior is the same in a regular case as in a failure case. The system supports as many streams under failure as it supports in the regular case. We use this retrieval scheme for both types of files : reliable files and non-reliable files. Hence, for non-reliable files, we only retrieve data blocks, and no parity blocks. If a node fails, then the reading operation just fails.

However, a major disadvantage of the SR scheme is the large amount of memory required by the system. To make this point clearer, let us consider a video server application. Video servers usually serve clients in a cyclic way. At each cycle, data is first read, and then it is sent to the client. This requires one block buffer. Let us now consider the SR scheme with a striping group size P. At each cycle, P blocks are read in parallel ; then, they are sequentially sent to the client. This requires P block buffers instead of just one. The memory requirement linearly grows with the striping group size. This hinders the system scalability. In [Bon01], we showed that this is not the case as our distributed design spreads the memory requirement across the cluster nodes.

2.3 CFS Implementation

The CFS is implemented as a set of processes, one on each node. On each node, additional processes are also used to implement the meta-server functionality. Each meta-server is in charge of managing a number of clients. It translates external client requests into internal service requests targeted to the local CFS process. The CFS process is made up of two threads :

- The *Job Treatment Thread (JTT)* is in charge of handling local meta-server requests, remote CFS requests and remote CFS replies. For this purpose, the JTT posts either asynchronous I/O to the local disk subsystem, or messages through the internal network. It communicates with the meta-server processes using shared memory in order to avoid copies.
- The *Message Treatment Thread (MTT)* receives messages through the internal network and forwards them to the Job Treatment Thread. It is also in charge of the detection of node failure.

The implementation of the open/close operations requires a complex fault-tolerant distributed synchronization between the CFS processes in order to manage file accesses. In this paper, we focus on the implementation of the reading operation.

The following example illustrates this scheme (Figure 5). Assume a client is connected to the meta-server on node i. This client requires data of a file striped using a striping group size equal to 1. In this case, a reading request consists in retrieving only one block. Assume this block is stored on node j $(i \neq j)$. The reading request is performed as follows. The meta-server posts a request to the JTT through the CFS API (Step 1). The JTT gets the request, and sends a request message to node j (Step 2). On node j, the MTT

receives the message and forwards it to the JTT (Step 3). The JTT posts a reading request to its local disk subsystem (Step 4). The JTT is informed of the I/O completion (Step 5) and it sends the data back to node i (Step 6). The data are received by the MTT on node i and are forwarded to the JTT and then to the meta-server (Step 7). The MTT on node i is also in charge of detecting the failure of node j by periodically sending probe messages. Detecting the failure of a node typically requires around 10 milliseconds.

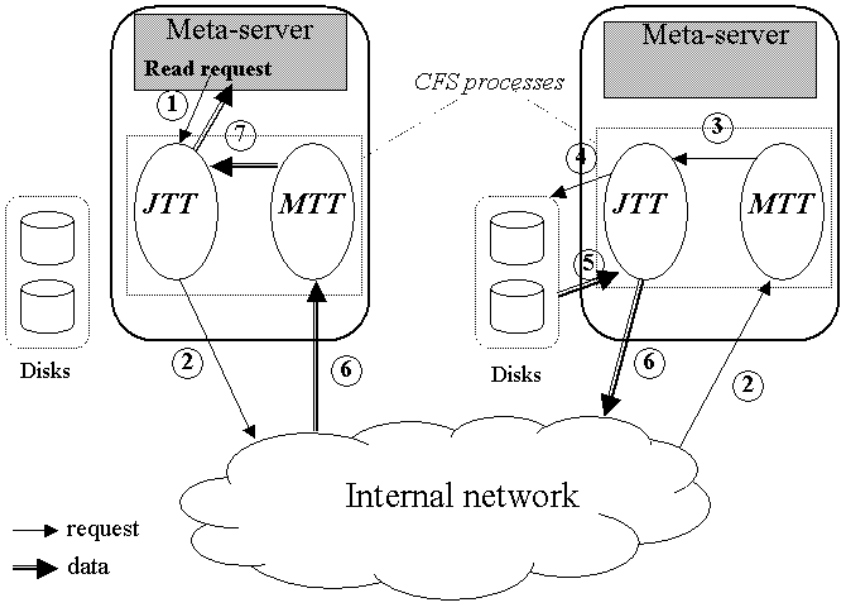


FIG. 5: *Example of the retrieval of a remote block*

Regarding files striped with a striping group size P greater than 1, P remote blocks are retrieved in parallel through the retrieval scheme described in the above example.

Consider first a non-reliable file. If a failure is detected while waiting for the blocks, then the reading request returns an error. Consider now a reliable file. In the case of the failure of one node, it would be possible to reconstruct the missing data block on the fly. However, this would induce an additional delay equal to the fault detection timeout. Therefore, an alternative strategy is implemented in the CFS. As soon as $(P-1)$ blocks are available, the last block is reconstructed on the fly, using the XOR operation. All P blocks are immediately returned to the meta-server. If the remaining block arrives later, then it is merely discarded. The overall behavior is exactly the same in the regular case as in the case of a single failure. The overhead is the systematic computation of the XOR operation. If more than one failure occur, then an error is returned to the meta-server, as soon as the second fault is detected.

3 Methodology for cost analysis

In this section, we propose a scheme to experimentally evaluate the CFS overhead. We consider only reading operations, because they are the most frequent in video applications. We follow a reading request throughout its course inside the CFS implementation. Our goal is *a)* to investigate how the cluster resources are used ; *b)* to identify possible bottlenecks and *c)* to evaluate the CFS management cost in terms of latency time. Cluster resources are : the disk subsystem, the network subsystem and the CPU. We first decompose the action of retrieving a block into 7 steps, and we identify the resources involved in those steps. Then, we consider a complete reading request (striping group size $\neq 1$), and we decompose its latency time with

	meaning	node involved	resource involved
Step 1	handling meta-server request	local node	CPU
Step 2	sending request to the remote node	-	network
Step 3	receiving and handling request	remote node	CPU
Step 4	posting and performing reading request	remote node	disk
Step 5	handling reading completion	remote node	CPU
step 6	sending back data	-	network
Step 7	handling data reception	local node	CPU

TAB. 1: *Presentation of the resources used in the 7 steps of a block retrieval : the request is performed on a node called local node whereas the block is stored on a node called remote node*

respect to the three resources of the cluster. Finally, we introduce the various parameters considered in the experimental tests.

3.1 Study of a block retrieval steps

The retrieval of a block is similar to a reading request performed on a file with a striping group size equal to 1, as illustrated in Figure 5. We did determine 7 steps in terms of their functionality. Now, we associate to each step, the resources which are used during each step, and the node involved (the local node on which the request is done or, the remote node on which the block is stored). We do not consider the case in which the local node and the remote node are the same. This represents a special case which uses only step 1-3-7. Table 3.1 summarizes the step characteristics.

In order to measure the duration of each step, we introduce six timers within the CFS code. We use an efficient timer function whose call costs around 4μs. Thus, it does not perturbate the performance too much. As shown in performance results, 24μs are insignificant compared to latency values.

3.2 Decomposition of a request latency time into 3 time components

In order to make some statistics about resources utilization, we combined previous results about steps, depending on the associated resource. Thus, a latency time is decomposed into the following components.

- *CPU time* = Step 1 + Step 3 + Step 5 + Step 7.
- *Disk time* = Step 4.
- *Network time* = Step 2 + Step 6.

Let us consider a reading request about a file with a striping group size equal to $n \neq 1$. Thus, the local node performs n block retrievals in parallel. So, for each block, we have the time spent in each of the 7 steps. We can therefore compute the CPU time, the disk time and the network time, associated to each block retrieval. Then, the global time values associated to the global latency, are computed as the average between the individual times of each block retrieval. However, if we consider a reliable file, when block $(P-1)$ arrives, the JTT computes block P. Thus, step 7 is longer for block $(P-1)$ retrieval than for the others. In this case, we do not consider the average time of steps 7, we rather consider the maximum one.

3.3 Tests parameters

For our experiments, we perform various tests about reading requests. These tests allow to study the CFS under various system loads and under various configurations. The test parameters are the following.

Block Size (BS), in KBytes : we use block sizes ranging from 16 KBytes to 128 KBytes.

Striping Group Size (SGS) : we use striping group sizes ranging from 1 to 8.

Number of simultaneous requests : we perform tests with one reading request, or with 10 reading requests which are simultaneously posted.

Number of nodes from which the requests are performed : we use either reading requests posted from one node, or reading requests simultaneously posted from several nodes.

Reliability level : we use reading requests about non-reliable files or about reliable files.

4 Experimental results

Experimental measures have been performed on a 8-node cluster made up of Intel bi-processors, connected through a Myrinet network using the GM communication system (developed by Myricom). Each PC has six 10000-rpm disks in a RAID 1 configuration. The operating system is NT4 and the disks are accessed through the NTFS file system.

This section presents results measured in various configurations and using the methodology described in the previous section. We first study the case of a unique reading request. Then, we consider the case of simultaneous reading requests. Finally, we analyze those results from a throughput point of view.

4.1 Latency of one request

4.1.1 With striping group size = 1

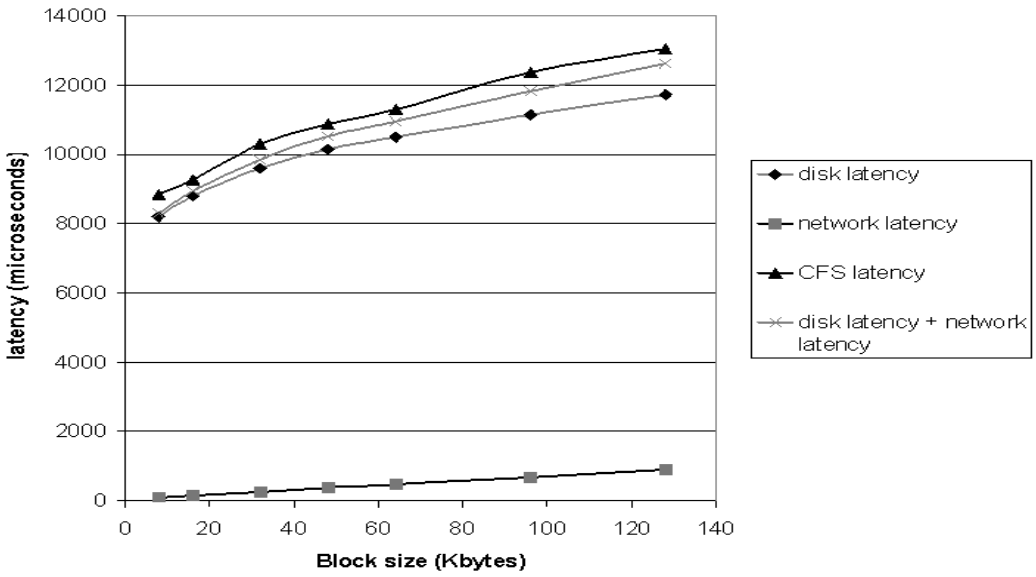


FIG. 6: *Evaluation of the CFS overhead on retrieving one remote block, versus the block size*

We consider reading requests about non-reliable files (reliable files require that $SGS > 1$). In this case, reading request is equivalent to the retrieval of one block. We more particularly study the retrieval of a remote block in order to observe the network overhead.

Figure 6 shows latencies of : the disk subsystem, the network subsystem and the entire CFS request, versus the block size. The difference between the disk latency and the CFS latency represents the CFS overhead. The network latency represents the cost of the block transfer between two nodes. The difference between the CFS latency and, the sum of the disk latency and the network latency, represents the time used for CPU treatment and for sending the request over the network.

We notice that as the block size increases, the CFS overhead also increases. This due to the block transfer on the network which linearly increases with the block size. Nevertheless, the CFS overhead remains acceptable, representing from 3% to 10% of the global latency.

4.1.2 Non-reliable files with striping group size $\neq 1$

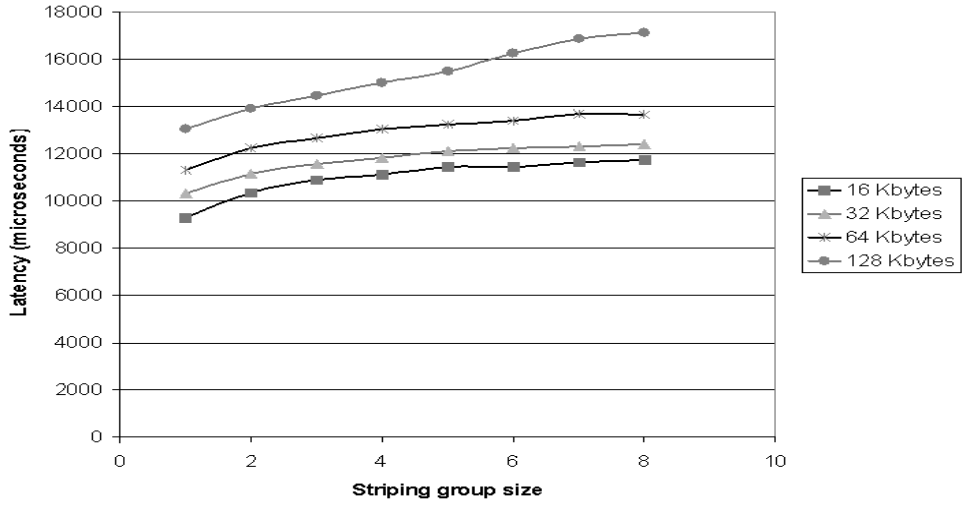


FIG. 7: *Evolution of the CFS latency versus the striping group size and the block size, for non-reliable files*

Figure 7 shows latency results for reading requests with various striping group sizes and various block sizes. In an ideal case (i.e. the CFS overhead equals zero), curves would have been straight, representing a perfect parallelism. In effect, as the block retrievals are performed in parallel, retrieving one block or P blocks should take the same time. In contrast, latencies plotted in Figure 7 increase as the striping group size increases. Furthermore, we notice that the speed of increase is more important for large block sizes, than for small ones.

To understand the reasons of those latency rises, we study the decomposition of the latency time, as described in the methodology section. Results, obtained for $SGS = 6$, are shown in Figure 8 (for $BS = 16$) and in Figure 9 (for $BS = 128$).

For $BS = 16$, we notice that the network overhead (Steps 2 and 6) is very small, almost insignificant, as well as the CPU overhead. The increase of the CFS latency is explained by another factor, related to the disk subsystem. In effect, the disk latency can be very varied. The disk latency is made up of two components : the time of the data transfer between the disk and the memory and, the time required to find the data on the disk. This second component includes the movement of the disk head to reach the accurate track and, the disk rotation. It represents an overhead compared to the disk bandwidth. Furthermore, this overhead is not related to the data size, it is rather related to the data position on the disk. Thus, no matter what is the data size, the overhead remains the same. Accordingly, on reading a big amount of data, this overhead represents a small percentage of the global latency. In contrast, for a small amount of data, it can represent more than half of the latency. This latency variability has therefore an impact on the CFS results. On reading P blocks in parallel on P remote disks, the global latency time equals the maximum value of the P disk latencies. Thus for a large value of P, the probability to have a long disk latency is greater than the one obtained

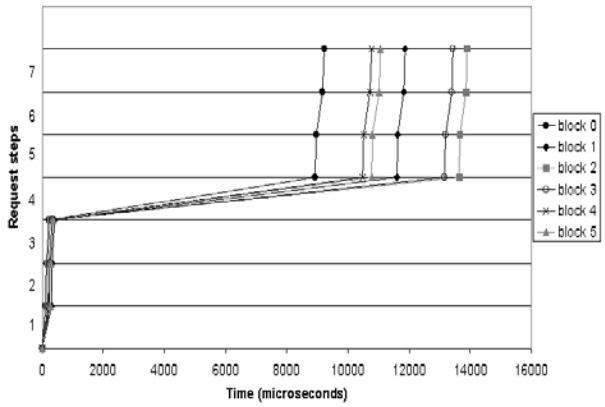


FIG. 8: *Decomposition of the latency into the request steps (SGS = 6 and BS = 16 KBytes)*

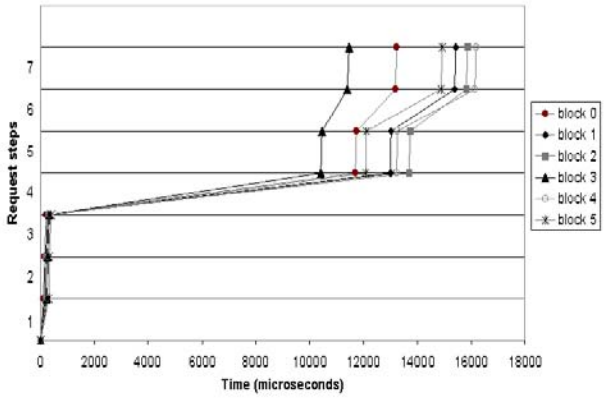


FIG. 9: *Decomposition of the latency into the request steps (SGS 6 and BS = 128 KBytes)*

for a small value of P. Consequently, part of the increase of the CFS latency is due to the disk subsystem. Nevertheless, the latency decomposition into various steps allows us to make the distinction between the latency rise due to the CFS overhead and the one due to the disk subsystem. Let us consider the case for $BS = 128$. We notice the increase of the CFS overhead, in particular the network time increase. The time spent in step 6 is greater than for $BS = 16$. This is firstly due to the linear latency of the network. Secondly, some of the messages overlap with each other over the network. As an example, block 1 is sent whereas block 5 has not been received yet. Thus, we have several transfers at the same time which consequently increase the network latency. Nevertheless, this overhead (around 10%) remains acceptable.

4.1.3 Reliable files

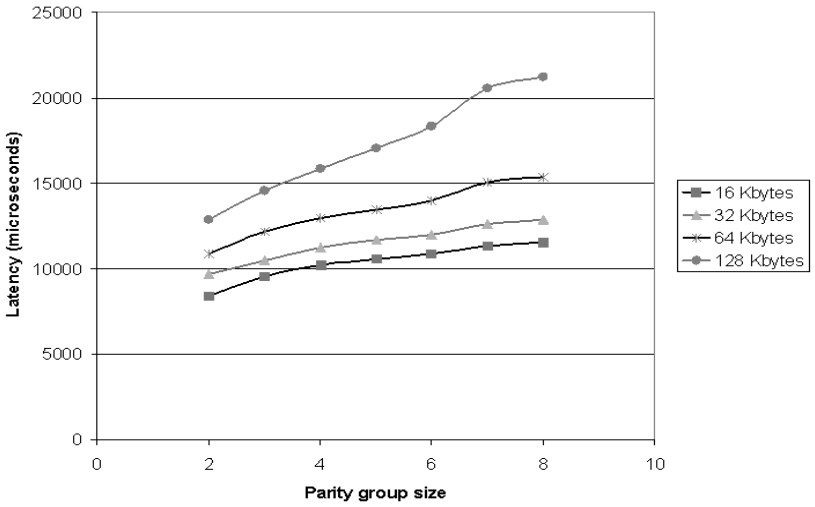


FIG. 10: *Evolution of the CFS latency versus the parity group size and the block size, for reliable files*

Figure 10 shows global latency results for reliable files, versus the parity group size and the block size. We notice that latency is more sensitive to the parity group size than for non-reliable files. This can be explained by studying in more details the latency. Figure 11 shows the decomposition of the latency, for a

reliable file with $SGS = 6$ and, $BS = 16$. Results are similar to the one for non-reliable files, except the fact that there is an additional CPU time when the XOR operation is computed after the arrival of block $(P - 1)$. In Figure 11, this overhead appears after the retrieval of block 4. The CPU overhead for the XOR computation increases linearly with the block size and the striping group size. Figure 12 gives an example of the CPU cost for $BS = 128$.

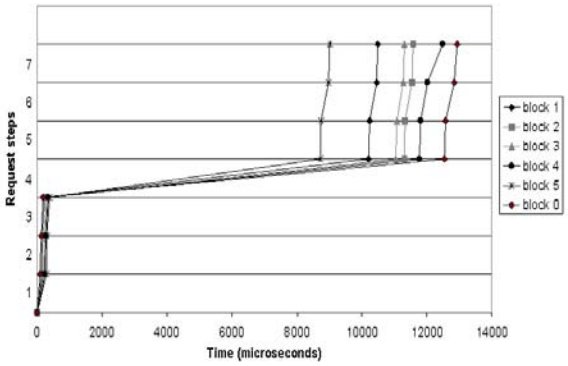


FIG. 11: *Decomposition of the latency into the request steps (SGS = 6 and BS = 16 KBytes), for reliable files*

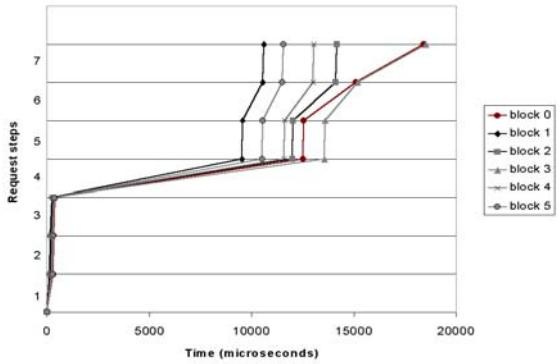


FIG. 12: *Decomposition of the latency into the request steps (SGS = 6 and BS = 128 KBytes), for reliable files*

4.2 Latency of simultaneous requests

We perform 10 simultaneous reading requests. For each request, we get information about the time spent using CPU, network and disk resources. Figure 13 plots results obtained for $BS = 128$ and various striping group sizes. Naturally, the disk utilization remains steady. The CPU utilization is so small that it does not appear on the plot. However, the network utilization increases as the striping group size increases. This is related to the data overlapping over the network. In worst cases, the network contention represents 20% of the request latency. Figure 14 illustrates the influence of the block size on the resources utilization, with $SGS = 6$. As for the one request case, the network overhead increases as the block size increases.

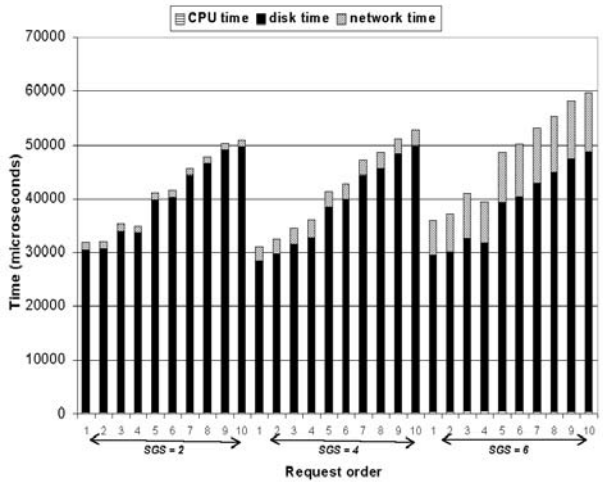


FIG. 13: *Decomposition of the latency for 10 simultaneous requests (BS = 128 KBytes, SGS = 2, 4 or 6)*

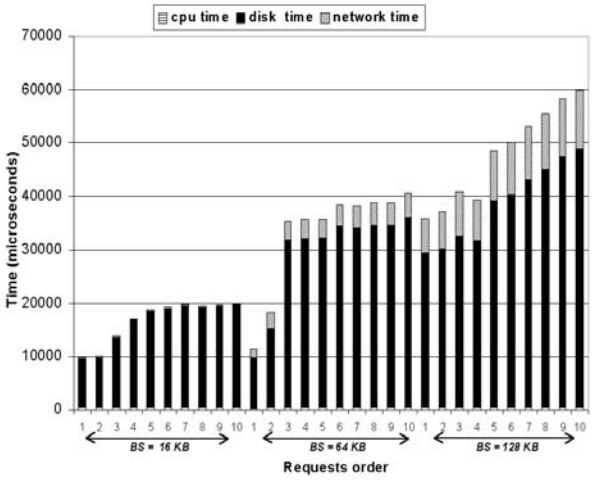


FIG. 14: *Decomposition of the latency for 10 simultaneous requests (SGS = 6 and BS = 16 KBytes, 64 KBytes or 128 KBytes)*

Figure 15 and Figure 16 show some experiments with reliable files. Results outline the CPU utilization required for the XOR computation. The plotting clearly show the linear evolution of the CPU time versus the striping group size and the block size.

Finally, we made some experiments with 10 simultaneous requests performed from several nodes. We do not plot the results here because they are similar to the one presented above.

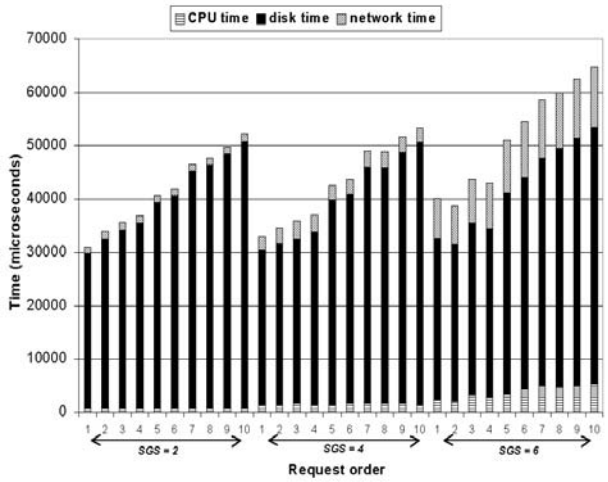


FIG. 15: *Decomposition of the latency for 10 simultaneous requests (BS = 128 KBytes, and SGS = 2, 4 or 6), for reliable files*

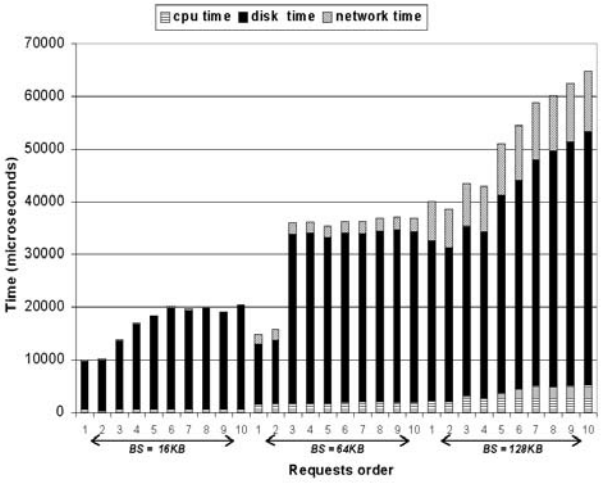


FIG. 16: *Decomposition of the latency for 10 simultaneous requests (SGS = 6 and BS = 16 KBytes, 64 KBytes or 128 KBytes), for reliable files*

4.3 Global throughput of the CFS

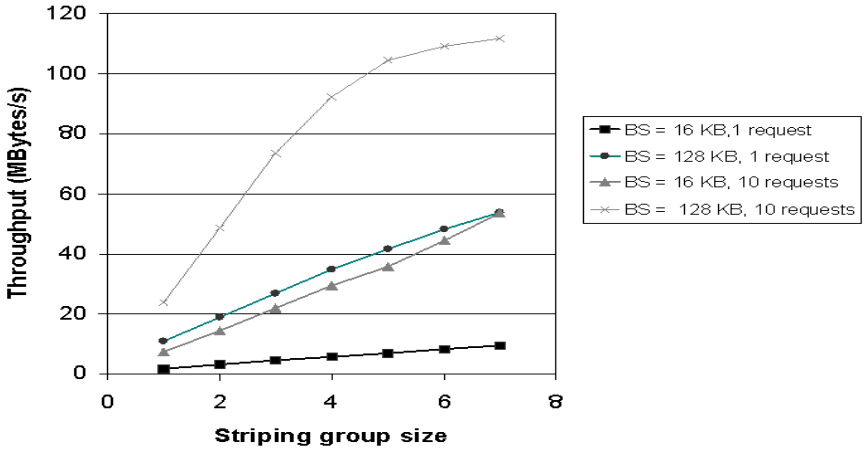


FIG. 17: *Global CFS throughput versus the striping group size*

So far, we have only considered latency times and percentages of these times due to the CFS overhead. Results clearly show that this overhead increases with the striping group size and the block size. Nonetheless, this does not mean that it is better to use small values of those parameters. In effect, for large block sizes

and large striping group sizes, the resulted throughput is higher. As an example, a reading request using $SGS = 6$ is slightly slower than a reading request using $SGS = 3$. Yet, the first request does retrieve twice much data than the second one. Figure 17 shows the results of previous experiments about non-reliable files, from a throughput point of view. We clearly notice that the higher throughput is reached using a large block size, even if the CFS overhead is more important in that case. However, when the striping group size is greater than 5, the throughput growth slows down. This is explained by the network overlapping due to simultaneous data transfers over the network.

To study the CFS scalability, we measure the maximum throughput we can reach with the CFS. We proceed as follows : during a certain time duration, we try to complete as many reading requests as possible. The throughput is computed based on the amount of data retrieved by these requests. For reliable files, we only consider data blocks to compute the throughput. The best results are obtained using $BS = 128$, and posting requests from several nodes. Furthermore, we post requests with a regular interval between them. In effect, if we post all the requests at the same time, a contention firstly appears on the disk subsystem and secondly on the network. Results are plotted in Figure 18. They demonstrate the scalability capability of the CFS design, for non-reliable files as well as for reliable files. We also notice the cost in terms of throughput of reliable files compared to the one of non-reliable files. This cost remains almost steady in terms of throughput decrease. This is due to the fact that no matter what is the striping group size, one additional block is retrieved. Nevertheless, we do not insist on that point since it is not the main topic of this paper.

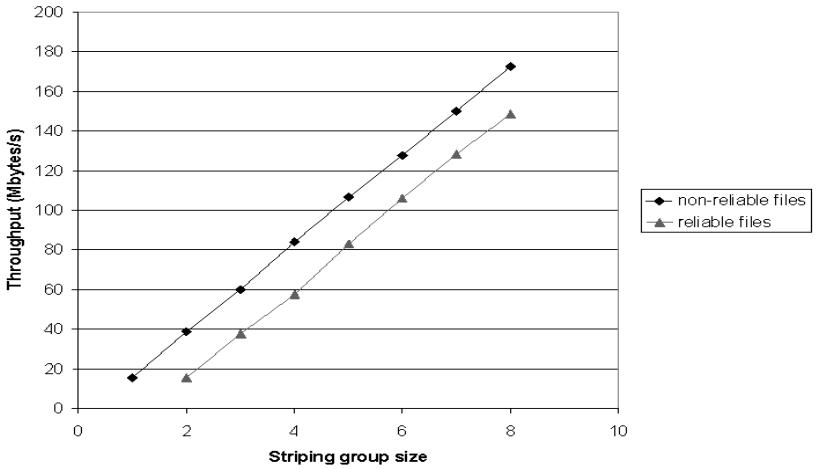


FIG. 18: *Global CFS throughput versus the striping group size*

5 Related Work

Many techniques have been investigated in order to achieve scalability for video servers : Ghandeharizadeh *et al.* [GZS+97], Muntz *et al.* [FSM98], Biersack *et al.* [GB99b], Lee and Wong [LW00]. A comprehensive study of approaches used by existing systems can be found in [GVK+95, Lee98]. In most cases, the storage system is integrated within the video server design.

Ghandeharizadeh *et al.* [GZS+97], work on optimizing local disk accesses. They particularly focus on data placement on one disk in order to reduce the variability of the disk latency. The Mitra server implements this principle and, experimental results show that it reaches good performance. In Mitra, reliability is enhanced using mirror disks. However, this solution requires very large storage space because of the extensive mirroring. Moreover, the mirror disk bandwidth is only used in case of failure. It is thus wasted most of the time.

Muntz *et al.* [FSM98] propose to randomly distributed the data blocks among the disks. They demonstrate

that this solution is very well suited for non sequential accesses (like in video editing or virtual reality applications, etc.) or for heterogeneous disk platforms. The RIO prototype is based on this idea.

Biersack et al. [GB99b] study striping methods and various reliability schemes based on mirror or parity techniques. They compare the different strategies, in terms of bandwidth and memory requirement. They conclude that the striping method and the reliability strategy are interdependent. In particular, they demonstrate that the mirror technique is best suited for classical retrieval scheme whereas, parity techniques are best suited for sequential scheme, like SR. Their ideas have been implemented in the Server Array prototype, which is based on classical retrieval scheme combined with a mirror technique. However, the reliability strategy is not transparent to the client. In fact, the client itself is in charge of part of the reliability management : it detects node failures, and explicitly requests mirror blocks.

In [LW00], Lee *et al.* focus on performance analysis for a distributed video server using a semi-centralized design. Their studies reveal that large number of simultaneous client requests may overwhelm the server. To solve this problem, they propose and analyze a staggering-based admission scheduler. Using modeling and simulation, they show that this scheduler enables the server to achieve scalability.

In contrast with these systems, we propose a storage system which is independent from the video server, and whose distributed management is transparent to the client. Furthermore, our experimental measures have shown that the internal network behavior has a significant impact on the overall performance. This aspect is too often underestimated in the analytical modeling.

6 Conclusion

In this paper, we have presented the design of the CFS, a distributed storage system dedicated to video data. This system is based on a fully distributed design and provides reliability properties. The goal of our study presented in this paper, is to experimentally evaluate the cost of the CFS design. This cost can be expressed in terms of network utilization and CPU utilization. For various configurations, we analyzed the time spent in various steps of a reading request and the resources used in each step. Thus, we determined the overhead introduced by the CFS management. Results showed that this overhead is acceptable, between 3% and 15%, depending on the configuration. Moreover, throughput results showed that it does not prevent the CFS from reaching good performance and from achieving scalability. However, it is not recommended, for large clusters, to use a striping group size equal to the number of cluster nodes. It is wiser to partition the nodes into smaller striping groups. It avoids contention over the internal network due to message overlapping, and it enables to support one failure per striping group. These results point out the importance of scheduling the client requests to avoid contention on the disks or on the internal network.

We are currently studying performance results for several simultaneous requests using various striping group sizes. In the future, we will also work on optimizing the local filesystem, as it is done in Mitra [GZS+97], in order to reduce the variability of the disk latency.

Références

[Bon01] A. Bonhomme. Scalability issues in a reliable distributed video storage system. Research Report RR2001-21, LIP, ENS Lyon, France, May 2001.

[FSM98] F. Fabbrocino, J.R. Santos, and R. Muntz. An implicitly scalable real-time multimedia storage server. In *Second Workshop on Distributed Interactive Simulation and Real-Time Applications (DIS-RT98)*, pages 92–101, June 1998.

[GB99a] J. Gafsi and E.W. Biersack. Data striping and reliability aspects in distributed video servers. *Cluster Computing : Networks, Software Tools, and Applications*, 2(1) :75–91, February 1999.

[GB99b] J. Gafsi and E.W. Biersack. Performance and reliability study for distributed video servers : Mirroring or parity ? In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS'99)*, Florence, Italy, June 1999.

[GLP98] L. Golubchik, J. Lui, and M. Papadopouli. A survey of approaches to fault tolerant design of VOD servers : techniques, analysis and comparison. *Parallel Computing*, 24(1) :123–155, 1998.

[GVK+95] D.J. Gemmel, H.M. Vin, D.D. Kandlur, P.V. Rangan, and L. Rowe. Multimedia storage servers : a tutorial and survey. *IEEE Computer*, 28(5) :40–49, November 1995.

[GZS+97] S. Ghandeharizadeh, R. Zimmermann, W. Shi, R. Rejaie, D. Ierardi, and A.W Li. Mitra : A scalable continuous media server. *Multimedia Tools and Applications Journal*, 5(1) :79–108, July 1997.

[Lee98] J.Y.B. Lee. Parallel video servers : a tutorial. *IEEE Multimedia*, pages 20–28, June 1998.

[LW00] J.Y.B. Lee and P.C. Wong. Performance analysis of a pull-based parallel video server. *IEEE Transaction on Parallel and Distributed Systems*, 11(12) :1217–1231, December 2000.

[ORS96] B. Ozden, R. Rastogi, and P.J. Shenoy. Disk striping in video server environments. In *Proceedings of the IEEE Conference on Multimedia Systems*, pages 580–589, June 1996.

[PGS88] D. Patterson, G. Gibson, and M. Satyanarayanan. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the 1988 ACM Conference on Management of Data (SIGMOD)*, pages 81–94, Chicago, IL, June 1988.

[SV00] P.J. Shenoy and H.M. Vin. Failure recovery algorithms for multimedia servers. *Multimedia Systems*, 8(1) :1–19, January 2000.

[TMDV96] R. Tewari, R. Mukherjee, D.M. Dias, and H.M. Vin. Design and performance tradeoffs in clustered video servers. In *the IEEE international Conference on Multimedia Computing and Systems (ICMCS'96)*, pages 144–150, May 1996.

[TPBG93] F.A. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID - a disk array management for video files. In *Proceedings of the ACM International Conference on Multimedia*, Anaheim, CA, August 1993.