Scalability Issues in a Reliable 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

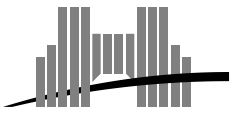
CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

Scalability Issues in a Reliable Distributed Video Storage System

Alice Bonhomme, LIP/ENS-LYON –
46, allee d'Italie, 69007 Lyon, France
E-mail: Alice.Bonhomme@ens-lyon.fr

May 2001

Research Report N° 2001-21



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

Scalability Issues in a Reliable Distributed Video Storage System

Alice Bonhomme, LIP/ENS-LYON – 46, allee d'Italie, 69007 Lyon, France
E-mail: Alice.Bonhomme@ens-lyon.fr

May 2001

Abstract

This article studies the design and the performance of a fault-tolerant distributed storage system dedicated to video. We use a completely distributed approach that permit to reduce the memory cost of the reliability scheme. We experimentally point out the scalability properties of this system.

Keywords: Video server, Scalability, Striping, Fault tolerance, Cluster

Résumé

Cet article étudie la conception et les performances d'un système fiable de stockage distribué dédié à la vidéo. On utilise une approche complètement distribuée qui permet de réduire le coût mémoire de la stratégie de tolérance aux pannes. Nous évaluons expérimentallement les propriétés d'extensibilité de ce système.

Mots-clés: serveur vidéo, tolérance aux pannes, grappes de PC, performances, distribution des données

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 differs significantly 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 present the design of the CFS (Cluster File System), an efficient storage system dedicated to video streams. Our goal is to have a storage system that is 1) efficient and with a good price/performance ratio, 2) with reliability properties.

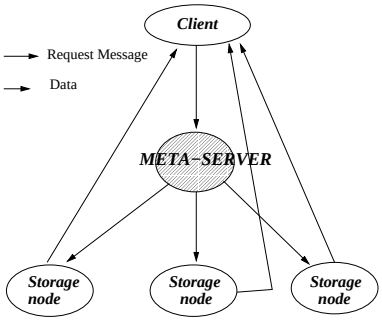


FIG. 1: *The centralized approach*

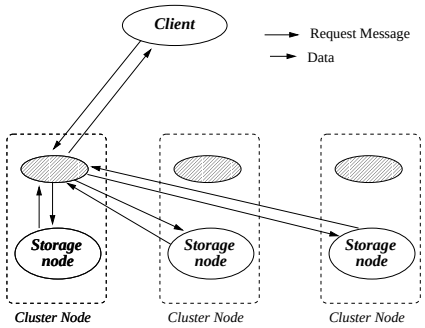


FIG. 2: *The distributed approach*

Point 1 can be addressed by using a material architecture made up of on the shelf components, such as cluster of PCs. One can split the cluster solutions into two main approaches. In the *centralized approach* (Figure 1), the clients are connected to a meta-server which informs the storage nodes about the requests. Then, the data are sent directly from the storage nodes to the clients. This avoids the bottleneck of network connection between the clients and the meta-server. In the *distributed approach* (Figure 2), the meta-server is distributed among the server nodes. Thus, each node of the cluster is at the same time a connection node for the clients and a storage node. The second design goal leads us to the choice between those two approaches.

Point 2, that is reliability, includes two problems : the efficient failure support problem and the NoSPoF (*No Single Point of Failure*) problem. The first problem means that the server should be able to support, under a storage node failure, as many streams as it supports in a normal case. The second problem means that the failure of any component of the server should not cause the crash of the global server. In order to efficiently support the failure of a storage node, reliable strategies [SV00, GLP98] are usually based on data distribution, also called *data striping*. These strategies are similar to the RAID[1] schemes. Hence, in order to tackle the first problem, we have chosen the Streaming RAID approach [TPBG93] which appears to be the best solution in performance terms, even if it has the disadvantage of a high memory requirement. The second problem (i.e. NoSPoF) is addressed by choosing the distributed approach. Because, in the centralized approach, if the meta-server fails, the whole server is no longer available.

In this paper, we present more precisely the CFS system, based on those design choices (cluster of PCs, distributed approach, Streaming RAID strategy). We show that besides reaching the NoSPoF goal, the distributed approach reduces the amount of memory required on each cluster node for reliability purpose. Furthermore, we experimentally investigate the scalability issues of this system, with or without the presence of failure, in order to validate the CFS design. The rest of the paper is organized as follows : Section 2 describes the design of the CFS and the streaming RAID strategy. Section 3 outlines the test methodology. Section 4 analyses the memory cost. Section 5 shows some experimental results. Finally section 6 discusses some related works, and section 7 concludes the paper and discusses some future works.

[1] *Redundant Array of Inexpensive Disks*

2 Design of the CFS storage system

In this section, we present the main issues which the CFS system is dealing with. The basic interface of the CFS is similar to the one of most high performance file systems with asynchronous functions for reading and writing, and other usual functions such as open, close, etc. Nevertheless, in this paper we focus on read operations, the more frequently used accesses within a video-on-demand server. Hence, we restrict the CFS presentation to the retrieval scheme.

2.1 Data placement

For the video data placement, we use a *striping strategy*. A video file is divided into blocks of equal size. These blocks are then distributed among the cluster nodes. Data striping ensures that loads from every client will be evenly distributed across all server nodes. Furthermore, for fault tolerance issues, we have *parity blocks* stored on a node of the server. The set made up of data blocks and the associated parity block is called a *parity group*.

2.2 Data retrieval

Data are retrieved from the storage nodes based on the Streaming RAID approach [TPBG93]. This scheme has been defined for a multi-disk server. In this study, we apply it to our distributed architecture. In most of the retrieval schemes, the data blocks are retrieved one after the other, at the time they are requested. However, when a failure occurs, there is a time overhead to get the redundant block. Within the Streaming RAID scheme, the granularity of data retrieval is the parity group. Thus for one read request, one block will be simultaneously read on each node. So, in the event of a node failure, the missing data can be reconstructed on the fly by a parity computation. There is no overhead needed to get the parity block since it has already been read. Nevertheless, the major disadvantage of this scheme is the large amount of memory required by the system which grows linearly with the block size and the number of nodes. In section 4, we show that the distributed architecture enables to reduce this memory cost by distributing it between the server nodes.

3 Performance methodology

In order to be able to evaluate the performance of the CFS system, we use a test methodology that defines the meaning of "number of streams supported" by a given configuration. This methodology uses a benchmark that simulates a set of clients connected to the nodes of the cluster and playing a video stream at a constant bitrate. The evaluation scheme is based on a principle of cycles. A set of read requests is performed in each cycle and delivered to the distribution network in the subsequent cycle. In our case, we focus on the retrieval performance so we do not send the data, instead, we just check that the retrieval operation did complete on time. This scheme is illustrated in figure 3. Suppose that we experimentally simulate N clients (we use a set of N read requests), if in each cycle all the requests have been completed, then we can declare that the server supports N streams.

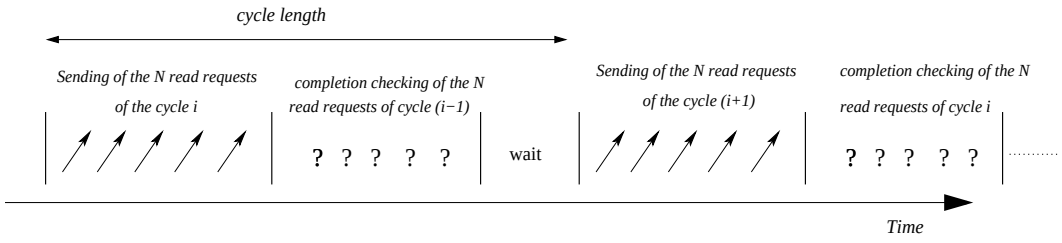


FIG. 3: *Description of a cycle*

Now, we have to determine for each test, how to fix the cycle length and the amount of data to read. Let C be the cycle length, S the block size, $B£$ the bitrate and P the number of server nodes. In order to meet the requirements given by the stream bitrate, the amount of data read in one cycle is directly related to the bitrate and the cycle length. The longer the cycle length is, the larger the block size will be. This relation is given by

$$S = \frac{C \times B}{P}. \tag{1}$$

Now if we consider the fault tolerance strategy, Equation (1) becomes

$$S = \frac{C \times B}{P - 1} \tag{2}$$

because one of the nodes is dedicated to the parity block.

In this study, we focus on 4 Mbit/s streams (MPEG2 video format), so we consider the bitrate parameter as a constant parameter. Reaching good performance for a disk subsystem requires a large block size, hence a long cycle length. However, within a video server context, a short cycle length (in the order of 1 or 2 seconds in our case) is crucial for interactivity reasons[2]. Hence, the choice of the block size will be the result of an acceptable trade-off between those two opposite constraints.

4 Memory cost analysis

In this section, we evaluate the memory cost of the streaming RAID approach associated to a distributed architecture. Let N be the number of streams supported per node on the cluster. Hence, the global server supports $N \times P$ streams. In the case without parity block, the memory necessary on each node is given by

$$\begin{aligned} M &= 2 \times N \times P \times S + (P-1) \times N \times S \\ &= N \times S \times (3 \times P - 1). \end{aligned} \tag{3}$$

The first term of the right hand side of the equation corresponds to the amount of memory necessary to store the data for the streams managed by the local node (NPS for data retrieved from the CFS, and NPS for data sent over the network). The second term is the amount of memory necessary to read blocks for remote streams.

Let us assume that the system is scalable. So, if we add a node to the cluster, the server will support N additional streams. The support of those new streams will only need $3 \times N \times S$ additional memory per node, regardless of the value of P.

Now, if we want the server to be reliable, we can add one node to the server in order to store parity blocks. Thus we have $P + 1$ nodes. In this case, the amount of memory necessary on each node to support N streams is given by

$$\begin{aligned} M &= 2 \times N \times P \times S + P \times N \times S + N \times S \\ &= N \times S \times (3 \times P + 1). \end{aligned} \tag{4}$$

The first two terms of the right hand side of the equation have the same meaning as in Equation (3), the remaining part corresponds to the memory necessary to store the parity blocks retrieved for the local streams. In summary, in order to have a reliable cluster, one need to add one node in the cluster, and it will cost an additional $2 \times N \times S$ amount of memory per node.

5 Experimental results

The 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.

[2] When a client performs a play, fastforward or backward operation, the time before he gets the first data should not be too long.

5.1 Tradeoffs between cycle length and performance

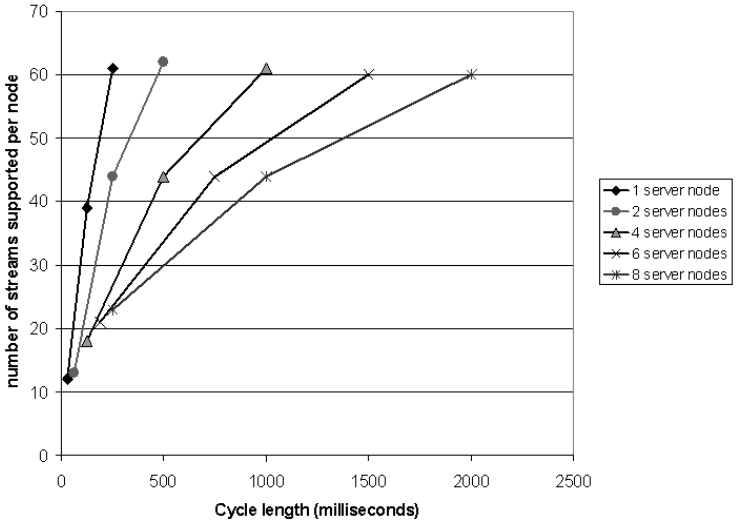


FIG. 4: Evolution of the number of streams per node versus the cycle length

As it is expressed in Equation (1), the cycle length has an important impact on the global performance because the block size is derived from the cycle length. We have measured the number of streams supported per node, in different configurations and with different cycle lengths. The results, plotted in Figure 4, clearly show that the best results are reached using a long cycle length. Furthermore, the more nodes the server has, the longer the cycle length necessary to reach good performance is. This is related to parameter P in Equation (1). In order to maintain a block size that takes advantage of the disk subsystem, the cycle length has to be increased when P increases. Furthermore, the increase of the cycle length permits to have more time for data transmission through the internal network. This transmission time actually increases as the number of server nodes increases.

5.2 Scalability results

Figure 5 shows the scalability properties of the CFS in both configurations, with or without a reliable strategy. We observe that, in both cases, the number of streams increases as the number of server nodes increases. We note that for 8 nodes, the required memory computed with Equation (3) reaches 172 Mbytes, a reasonable value for a video server.

The relation between the two curves is illustrated in Figure 6 in terms of cost of the fault tolerance strategy. This cost is evaluated by comparing the number of streams supported with and without a fault tolerance strategy. The decrease of the cost is mainly due to the fact that assuming a distribution on P nodes, the cycle length is shorter for a non-redundant distribution than for a redundant distribution. From Equation (1) and (2), one can compute that this difference has a fixed value equals to $\frac{S}{B}$. Thus, when P increases, the global cycle length also increases and $\frac{S}{B}$ represents a few percentage of it. For example, with a block size equals to 128 KBytes, the cycle length for a 2-node server equals 250 ms with parity and 500 ms without parity, whereas for a 8-node server, the cycle length equals 1750 ms with parity and 2000 ms without.

6 Related works

A lot of techniques have been investigated in order to achieve scalability for video servers. Those techniques have been implemented within video server prototypes. For example, studies have been conducted

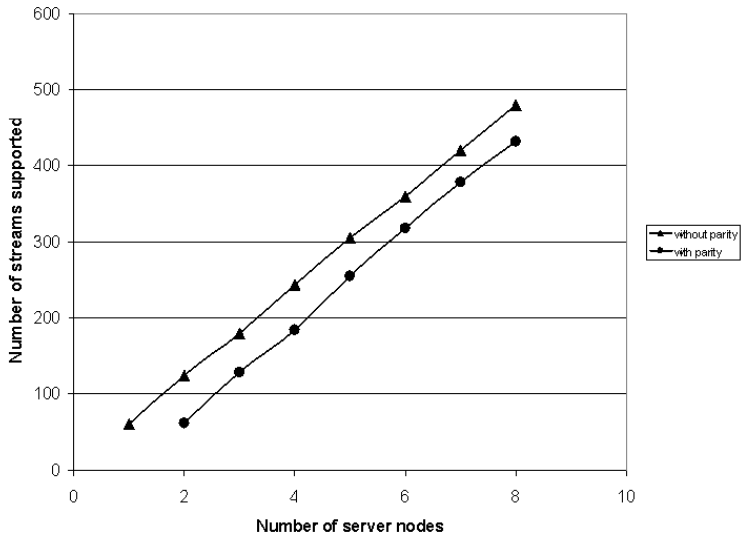


FIG. 5: Evolution of the number of streams globally supported by the server, versus the number of server nodes (the block size is 128 Kbytes)

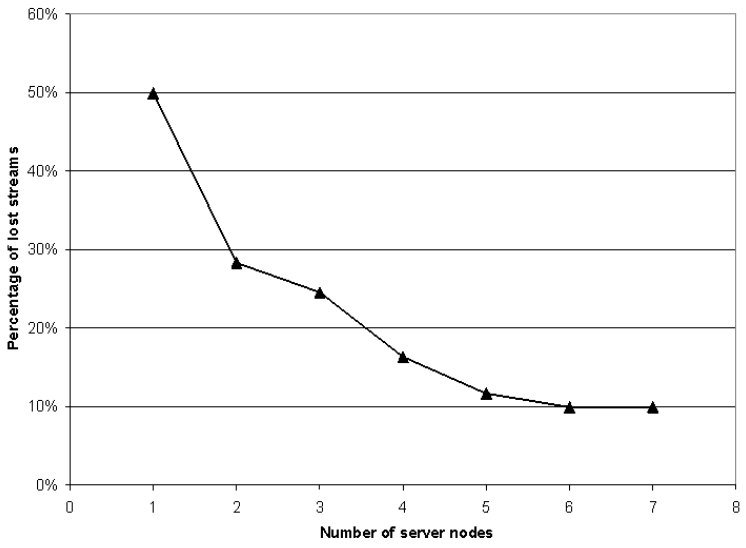


FIG. 6: Cost of the fault-tolerance capability (the block size is 128 Kbytes). This value is computed as the percentage reduction of the number of streams supported, compared to the non-redundant configuration with the same number of nodes

by Biersack et al [GB99], Muntz et al. [FSM98], Ghandeharizadeh et al. [GZS+97], Lee and Wong [LW00]. A comprehensive study of architectural alternatives and the approaches employed by existing systems can be found in [GVK+95, Lee98]. The main difference between the mentioned studies and the architecture studied in this paper lies in the fact that they implement different reliability strategies on top of a centralized approach. Furthermore, none of them permits to support a node failure without any performance degradation.

7 Conclusion

In this paper we have presented the CFS, a reliable distributed storage server dedicated to video streams. This system is based on a distributed approach. All components are distributed among the server nodes, thus, the failure of any component do not cause the whole server crash. We show that this approach allow to use a reliable strategy called Streaming RAID by reducing its memory cost. Finally, we experimentally point out that this design can be scalable. Further works will focus on analyzing this system under different bitrate constraints, especially low bitrates and variable bitrates.

Références

[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.

[GB99] Jamel Gafsi and Ernst 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] Kack Y.B. Lee. Parallel video servers : a tutorial. *IEEE Multimedia*, pages 20–28, June/April 1998.

[LW00] Jack 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.

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

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