Improving Reliability of Distributed VoD Servers

Manuel Billot
Thomson, Irisa

Valérie Issarny
Inria, Irisa

Isabelle Puaut
Insa, Irisa

Michel Banâtre
Inria, Irisa

Abstract

VoD services (Video on Demand) are characterized by QoS (quality of service) requirements which depend on:

- *the quality of video and sound transmitted to the client;*
- *the respect of time constraints associated to video and audio data;*
- *the reliability and the accessibility of the service.*

Much work has been done in order to provide system support aimed at meeting reliability requirements in distributed VoD servers. However, existing proposals focus on the treatment of failures of the network and disk, and do not integrate the consequence of CPU failures. To deal with this issue, we have previously proposed a resource reservation model that integrates reliability requirements. In this paper, we customize the resource manager of a distributed VoD server in order to take advantage of the large set of resources (disks and CPUs) to enhance the server's reliability. The proposed available-resource manager implements a load balancing strategy which provides reliability on demand and guarantees the server's accessibility. This paper details the corresponding algorithm and presents results proving that our load balancing strategy reduces the cost of reliability in terms of resource consumption.

1 Introduction

To be attractive, VoD (Video on Demand) applications have to provide a QoS (Quality of Service) better than the one of TV providers. The QoS of such applications depends on different criteria including the quality of audio and video and the variety of proposed services and offered videos, which depends on economical considerations. It also depends on the availability of the service, which consists in the ability of the server to respond immediately to clients demands (*accessibility*) and the ability to ensure a continuous service in the presence of failure (*reliability*). In this paper, we limit our field of investigation to strategies allowing to provide simultaneously accessibility and reliability. In VoD servers [Kandlur 93, Lougher 93], accessibility and reliability are often conflicting parameters, since the cost of providing reliability lowers accessibility. One of the most difficult issue is therefore to achieve reliability at a reasonable cost.

In this paper, we consider a VoD server distributed over a set of CPUs managing disks, interconnected by a network. This type of server is highly scalable and has a low-cost compared to dedicated architectures [Furht 95]. Furthermore, the multitude of CPUs and disks used to serve different clients may also be exploited to ensure reliability [Berson 94]. We propose a solution to the management of the VoD server's resources so as to provide accessibility together with reliability. Our proposal supports failures of disks but also of CPUs of the distributed VoD server. Our design goals were to implement a service which is able to offer tunable reliabilities depending on the server needs, the different degrees of reliability resulting in various levels of hardware costs. This proposal builds on our previous one called *available-resource* [Billot 96] that allows to consider reliability as a parameter of QoS in multimedia applications. Thus, the provided reliability results from a negotiation between the applications and the underlying system.

Section 2 introduces our proposal aimed at enhancing reliability of VoD servers. Section 3 describes our implementation of an available-resource manager which allows to optimize both reliability and accessibility of the VoD service. Finally, section 4 addresses the implementation of a VoD server based on our proposal and the evaluation of the service's provided accessibility compared to the one provided by RAIDs.

2 A proposal to ensure reliability of VoD servers

Subsection 2.1 presents a short overview of the methods currently used in distributed VoD servers to provide both real-time and accessibility guarantees. Then, subsection 2.2 briefly describes a previously proposed solution allowing to meet real-time constraints of multimedia applications while providing reliability on demand. Thereafter, subsection 2.3 introduces the notion of available-resource manager, which allows to optimize accessibility of VoD servers.

2.1 VoD servers

We consider a VoD server based on the MPEG2 standard [(MPEG) 93] which imposes to provide the multiplexed flow of video and audio data with a constant rate (see figure 1). The decoder used by the client is synchronized on

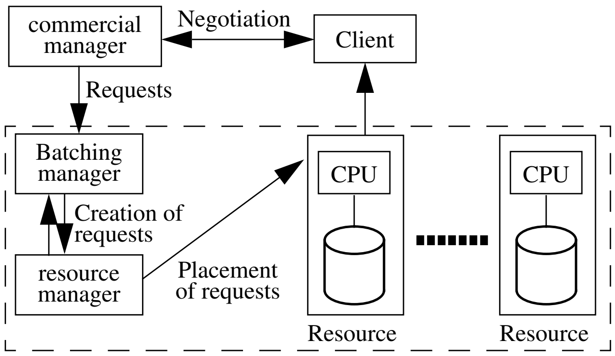

Figure 1. Architecture of a distributed VoD server.

a Constant Byte Rate (CBR). Therefore, any data arriving too late at the decoder input is considered as useless. In the above VoD server, real-time constraints are usually met by means of resource reservation [Anderson 93], where a ressource is a CPU and the managed disks. The resource reservation bounds the number of requests that are simultaneously accepted. Servers have to optimize accessibility denoted by the number of simultaneously accepted clients while meeting QoS constraints imposed by the MPEG2 standard. The optimization of the accessibility of the service relies on batching [Aggarwal 96] and on load balancing strategies. These strategies are briefly sketched hereafter.

The load balancing strategy used for improving accessibility of VoD servers requires the collaboration between two major inter-dependent policies: the *video placement policy* and the *request placement policy*. The video placement policy consists in choosing the placement of the videos on the different disks and possibly in managing the replication of videos. The request placement policy consists in dynamically assigning client requests to the CPUs managing the different disks storing the requested video, according to load balancing consideration [Wolf 94]. This policy is implemented by a *resource manager* (see figure 1). Existing video placement policies rely on the replication of different video streams according to their requesting probabilities [Keeton 93, Little 93]. More precisely, the number of copies of each video stream is an increasing function of its requesting probability. The distribution of the copies is generally daily updated in order to tackle the variation of requesting probabilities. The accessibility of a video is given by the number of simultaneously acceptable requests and is the sum of the numbers of acceptable requests on each resource storing the video. In order to preserve the ratio between the accessibilities of the different videos, the request placement policy has to balance the load among the set of resources. In VoD applications, several client requests for the same video may arrive at the server in a small time-interval. In this case, the server can handle these requests as a single one by multicasting the video stream to the corresponding set of clients [Aggarwal 96]. We suppose that this strategy, qualified as batching, is handled by a *batching manager* (see figure 1) independent of the resource manager. Therefore, the resource manager handles requests of the batching manager.

To summarize, a resource reservation policy ensures that the service's real-time constraints are met, whereas policies managing batching and placement of videos and requests, together allow to maximize the service's accessibility. These methods do not take into account reliability requirements. In this paper, we propose a method offering reliability on demand to clients while preserving accessibility. This method extends the concept of available-resource [Billot 96], by selecting dynamically the elementary resources that are composed for implementing reliability mechanisms.

2.2 Available-resources for ensuring reliability

From the standpoint of reliability management, our proposal assumes the following passive replication scheme between two resources storing the requested video [Lee 90]. In the absence of failure, the *primary resource* sends data to clients with a fixed data rate. In the case of failure of the primary resource, the *secondary resource* approximates the breaking time of the video and takes over data transmission. The approximation of the breaking time may introduce erroneous recovery which can lead the service to either not display a one-second sequence or to display it twice. However, this does not reduce the understanding of the video. This method has been implemented in the VoD server used to evaluate our proposal (see subsection 4). However, notice that although any passive replication method could be supported.

We have seen in the previous subsection that a single request issued by the batching manager may handle a set of clients. For economical reasons, the server may have to guarantee a greater reliability to requests managing the most numerous clients. Therefore, the reliability has to be tunable depending on the batching manager's needs. We have previously proposed a method allowing to consider reliability as a QoS criterion [Billot 96]. In that framework, reliability can be negotiated between the applications and the underlying system. This proposal is based on the resource reservation principle [Anderson 93] and introduces the concept of *available-resource*. An available-resource offers reliability on demand to multimedia applications by implementing a replication mechanism. An available-resource is statically composed of physically independent elementary resources, which ensures service continuation in spite of failure occurrences. The negotiation is conducted according to the application's needs in terms of processing capacity and reliability, and to the resources' capacity. In addition to transparently implementing a replication scheme, an available-resource handles reservation of the processing capacity of its constituting elementary resources based on the capacity required at the level of the available-resource. The above method of resource reservation for reliability applies directly to a VoD server. Let us consider a VoD server managing a large set of resources (see figure 1). Remind that each resource is composed of a CPU and the controlled disks. Resources are physically independent and can be combined so as to build available-resources. Secondary resources being unused while the associated primary resource is un-faulty, we have shown that the processing capacity reserved for a secondary resource can be shared between different available-resources made up of a common secondary resource [Billot 96]. This leads to lower the actual reliability of applications in the presence of simultaneous failures of primaries having a common secondary. On the other hand, this also leads to lower the processing capacity reserved for reliability purpose and hence to optimize resource consumption.

This paper introduces an enhancement of the available-resource notion for distributed VoD servers. It allows to dynamically select and group the elementary resources building up available-resources upon each request placement. As a distributed VoD server provides a large set of resources, the selection, which is made so as to optimize both load balancing and the offered reliability, results in an increased accessibility. The basics of the available-resource manager in charge of the dynamic selection of elementary resources is described in the next subsection. Then, we detail the mechanisms implementing the corresponding selection and allocation of resources in section 3.

2.3 Managing available-resources

The available-resource manager has to satisfy the demanded reliability for each request. As we mentioned previously, the number of clients served by a single request may be a major criterion of the requested reliability. The available-resource manager has also to provide a low cost service, while ensuring the requested reliability. In this case, the cost is evaluated in terms of the accessibility decrease implied by the request's processing. The available-resource manager offers reliability on demand according to the number of allocated resources (the client can use several physically-independent resources, in order to increase reliability) and the use of shared reservation (we have proposed a method (see section 2.2) allowing to share a reservation of a secondary resource).

In the following, we consider that all the videos require the same constant data rate. This condition corresponds to the MPEG2 standard. However, the proposed method can be generalized to variable data rates. The negotiation protocol decomposes as follows, upon a client's request:

- The request of the batching manager describes the video name and the requested reliability;
- The available-resource manager evaluates its ability to satisfy the request by considering the resources' load:
  - If it can fulfill the request's requirements, the request is accepted;
  - If the available-resource manager can not accept the request, the negotiation is broken;
  - If the available-resource manager can accept the request with a lower reliability, this reliability is returned to the batching manager; the batching manager then decides if it accepts the request with the proposed lower reliability.

The above protocol requires to describe the reliability requested by the batching manager and the one provided by the available-resource manager. The *reliability* criterion used in the negotiation is defined as the probability of a resource to operate correctly, which is computed according to the resource's failure rate and failure duration. For instance, the reliability of an elementary resource is given by the resource's $MTBF$ (Mean Time Between Failure) and $MTTR$ (Mean Time To Repair). Let us denote R as the reliability of the considered elementary resource; then $R = \frac{MTBF}{MTBF+MTTR}$. Let us now define R_1 and R_2 as being the reliabilities of two elementary resources. The reliability R provided by the available-resource made of these two resources can then be defined as $R = R_1 + (1 - R_1) * R_2$ assuming non-shared reservation. The evaluation of reliability in the presence of shared reservation is described in section 3.2.1.

The proposed available-resource manager is designed to provide reliability on demand to the batching manager. It has also to implement a load balancing strategy aimed at optimizing accessibility (see section 2.1). We propose in the next section an implementation of the available-resource manager that offers both reliability on demand and accessibility guarantees.

3 Implementing an available-resource manager providing high accessibility

We consider a VoD server implemented on top of a cluster of resources that can be reserved. The unit of processing capacity of a resource that is used to accommodate a request is qualified as a *slot*. Each resource is therefore able to handle a bounded number of slots. We have shown in the previous section that the available-resource manager is able to adjust the provided reliability by choosing the number of elementary resources and by sharing the reservation of secondary resources. The available-resource manager can therefore reserve several slots for a single request to provide the requested reliability. However the number of slots have to be set so as to minimize the allocation cost in terms of accessibility of the proposed service. The overall cost of a request depends on the two following parameters:

- The *number of reserved slots*: whereas in existing VoD servers a request always requires a single slot, our method can reserve several slots, where each is on a different resource. In order to optimize the cost of a request, the algorithm has to reduce the number of allocated slots and has to privilege shared reservation;
- The *relative accessibility of each video*: the ratio between the accessibilities of the different videos (see subsection 2.1) has to be preserved.

Given the above requirements, we propose an algorithm for request placement. Each resource is described by means of the set of videos which are stored on the resource and the maximal load in terms of slots, accepted by the resource. In order to simplify the description of the proposed method, we assume that all the elementary resources have the same specific reliability value. In addition, we implement a 2-resilient passive replication scheme (see subsection 2.2), which therefore bounds the provided reliability. However, the algorithm can be generalized to consider other resiliencies. As an example, let us take the treatment of a request. The available-resource manager achieves a *primary reservation* on a resource, which executes the *primary process* (see subsection 3.1). If the required reliability is greater than the specific reliability of the primary resource, the available-resource manager makes a *secondary reservation* on another resource on which it places a *secondary process* (see subsection 3.2). Notice that the processing capacity of the secondary resource may possibly be shared.

3.1 Placement of primary processes

The maximal load of each resource is characterized by three integer values: the number of slots allocated to primary processes ($primary$), slots allocated to secondary processes ($secondary$), and unallocated slots ($free$). These values evolve each time a request is placed and are used by the available-resource manager to evaluate the current load of the resource. During the placement of a primary process, the allocation always takes one slot (i.e., $primary := primary + 1$). The cost of this placement is therefore evaluated by taking into account the relative accessibility of each video based on the work of [Wolf 94]. Briefly stated, in order to maximize the relative accessibility of each video, the available-resource manager balances the load between the resources by maintaining as close as possible the *accessibility ratios* between the current load (i.e., $primary + secondary$) and the maximal load (i.e., $primary + secondary + free$) for all the resources. The available-resource manager computes the accessibility ratio for each of the resources storing the video, and the lowest ratio determines the placement having the lowest cost.

3.2 Placement of secondary processes

From the standpoint of the secondary process, the cost of its placement is evaluated according to two criteria: the number of allocated slots and the relative accessibility of each video. The number of allocated slots depends on the chosen placement. More precisely, each resource handles a set of secondary slots whose cardinality may be lower than the number of secondary processes placed on the resource thanks to reservation sharing. It follows that the set of secondary slots already allocated can suffice to provide the reliability requested by an additional incoming request. In such a case, the number of secondary slots does not change, that is to say, the *difference of secondary load* is null. In the following, we introduce the algorithm proposed to evaluate the difference of secondary load. This algorithm is executed for each resource storing the video except the one handling the primary. Among this set of resources, the algorithm chooses the one whose difference of secondary load is the lowest. If several resources ensure the same difference of secondary load, the algorithm chooses one among those providing the best load balancing as sketched for the placement of primary processes.

3.2.1 Evaluating the difference of secondary load

Let us consider the placement of a new secondary process on the resource R; $secondary$ represents the number of secondary slots allocated on R. This value is initially set to null and incremented in order to guarantee the reliabilities requested by the set of requests that R handles as a secondary. Let us denote as $req(R_j, R)$ the set of requests handled by R_j as a primary resource and by R as a secondary one. Let us now consider the cases upon which some requests of $req(R_j, R)$ may be stopped due to the failure of some resources except R, in addition to the failure of R_j. This may happen due to the single failure of R_j if all the secondary slots of R are reserved by available-resources whose primary was on R_j and if shared reservation was used. This may also happen due to the failure of some resource R_i prior to R_j. In this case, shared reservation was used and secondary slots are already used by requests of $req(R_i, R)$ that were served by R_i. The proposed method does not consider more than two simultaneous failures, since the probabilities are negligible compared to the probability of two simultaneous failures. Let us denote:

- $P(i_1, ..., i_n)$ as the probability of each sequence of failures (where $0 \leq n \leq 2$ and n equals the number of failed sites and i_j is the j^{th} site that failed among $i_1, ..., i_n$);
- $pr(i_1, ..., i_n)$ as the *ratio of continued requests*, which is the probability for a request $req(R_j, R)$ to be continued in spite of failure of $i_1, ..., i_n$.

The algorithm computes the reliability guaranteed to the requests of $req(R_j, R)$ by taking into account the different failures that may affect their processing . We identify three situations:

- The resource R_j operates correctly: the probability that R_j operates is $P() = Rel$ where Rel denotes the reliability of elementary resources. The ratio of continued requests of $req(R_j, R)$ is $pr() = 1$ since the primary resource operates.
- The resource R_j is failing and another resource R_i failed before R_j: let us denote $Prec(i, j)$ as the *probability of precedence* of R_i failure before R_j failure. In the case of resources having the same MTBF and MTTR, $Prec(i, j) = 0.5$. We get:

$$P(i, j) = (1 - Rel) * (1 - Rel) * Prec(i, j)$$

$$pr(i, j) = min(\frac{max(secondary - req(R_i, R), 0)}{req(R_j, R)}, 1)$$

- The resource R_j is failing and no other resource failed before: We get:

$$P(j) = 1 - Rel - \sum_{i \neq j} P(i, j)$$

$$pr(j) = min(\frac{secondary}{req(R_j, R)}, 1)$$

Then, the reliability guaranteed to $req(R_j, R)$ is denoted as F:

$$F = P() * pr() + P(j) * pr(j) + \sum_{i \neq j} (P(i, j) * pr(i, j))$$

The evaluation of reliability for each set of requests $req(R_k, R)$ allows to verify if the allocated secondary load ($secondary$) is sufficient to provide the requested reliabilities. If one of the reliabilities is not guaranteed, $secondary$ is incremented up to a value that guarantees all the requested reliabilities. This algorithm evaluates the secondary load to allocate on a resource R in order to support a new secondary process. The difference of secondary load is therefore the difference between the secondary load allocated after and before the placement of the secondary process.

The above algorithm allows to evaluate the difference of secondary load in the case of the placement of a secondary on the resource R. This algorithm is executed for each possible secondary placement, hence allowing to determine the best one. We present in the following section an example of its use for placing two requests on a given architecture.

3.3 Example of request placement

We consider a server made of three resources, each having a null current load and the same maximal load. The resources are named R_i ($0 \leq i \leq 2$) and have a reliability of 0.9. Let us now assume two successive requests requiring 0.98 as reliability and the same video that is stored on all the resources. Since the reliability of the first request is greater than the reliability proposed by a single resource, the request requires the allocation of a primary resource and of a secondary one. The three resources having the same load, any of them can be equally selected as a primary, we choose R_0. The allocations are represented in the following table:

resource	primary slots	secondary slots
0	1	0
1	0	0
2	0	0

We can not select R_0 as a secondary resource. Based on the above, the different opportunities for the secondary placement are presented in the following table, the reliability values being computed by the algorithm introduced previously:

solution			evaluation	
(prim./ sec.)	sec. slots	difference of sec. load	(prim./ sec.)	guaranteed reliability
(0/1)	0	1	(0/1)	0.990
(0/2)	0	1	(0/2)	0.990

The value of *guaranteed reliability* corresponds to the reliability guaranteed to the set of requests having the same primary and secondary resources. This value is computed considering the allocation of (*secondary slots+difference of secondary load*) on the secondary resource. In each of the considered cases, the placement of the secondary process requires the allocation of one unit of secondary load (*difference of secondary load=1*), we choose to place the secondary process on R_1. Since R_2 is idle, the primary process of the second request is allocated on R_2. We get:

resource	primary slots	secondary slots
0	1	0
1	0	1
2	1	0

The placement of the secondary process requires the following evaluations:

solution			evaluation	
(prim./ sec.)	sec. slots	difference of sec. load	(prim./ sec.)	guaranteed reliability
(2/0)	0	1	(2/0)	0.990
(2/1)	1	0	(0/1) (2/1)	0.985 0.985

Hence, we do not need any further allocation to guarantee the requested reliability if we choose solution (2) (*difference of secondary load=0*):

resource	primary slots	secondary slots
0	1	0
1	0	1
2	1	0

The presented example shows that the allocation of three slots allows to guarantee the reliability requested by the two clients and to perform load balancing.

In the following section, we present an implementation of a VoD service using the proposed available-resource manager. Then, we demonstrate that this method provides reliability on demand and allows to maximize the server's accessibility.

4 Implementation and evaluation of the method

Subsection 4.1 describes our prototype of a VoD service that has been used for simulation experiments. Then, subsection 4.2 addresses the cost of providing reliability using our method, which is compared to the one of RAIDs in subsection 4.3.

4.1 VoD service

In our prototype, the VoD server executes on top of pentium-based PCs running Chorus V3r5 [Rozier 92] (see figure 2). Furthermore, we have developed clients that

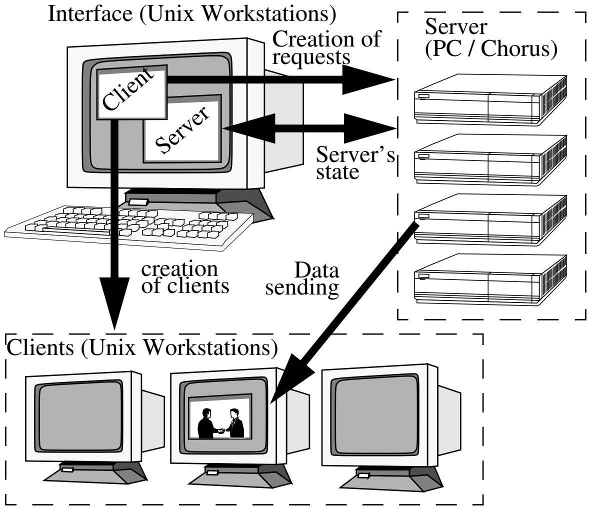

Figure 2. Architecture of the VoD service.

conform to the industry-standard [(MPEG) 93, Furht 95]. However, as we have no hardware available, these are emulated by software. This prototype of the VoD server allows us to emphasize the simplicity of the method proposed to enhance reliability: the implementation of the replication mechanism and of the resource manager, together took one-man-month. A set of tools have been implemented to observe the server behavior: they allow us to create clients on a set of Unix stations and to send these clients' requests to the server. The graphical interface of the server further displays placements of primary and secondary processes, failures of resources, and the recovery processing.

This prototype allowed us to evaluate the simplicity of implementation of this proposal. However, this prototype does not allow us to consider various architectures. Therefore, we evaluate our proposal, in the following subsection, by simulating larger architectures.

4.2 Cost of reliability

The simulation allows us to evaluate the cost of reliability in terms of resource consumption, including the impact of shared reservation. The evaluation of this cost depends on parameters like the video placement policy, the architecture of the server, and the variety of client's requests. In this paper, we concentrate on the evaluation of the cost according to the requested reliability. We define the cost noted (C) as being the number of reserved slots per request. Given the above requirements, we provide two cost references:

- When all the requests do not use any replication mechanism, the resource manager reserves a unique slot ($C = 1$) per request.

- When all the requests use a replication mechanism without shared reservation, the resource manager always reserves two slots ($C = 2$) per request.

The two above references define a scale of cost, which is used to evaluate the cost of each request. We evaluate the impact of varying reliability upon the accessibility of the server. The accessibility also depends on the number of copies of the video stored by the server. In the following, we give results for resources storing copies of a single video. The number of copies ranges from two to six. Each resource is able to handle concurrently at most 30 requests and its reliability is fixed to 0.90. We consider a set of similar requests (identical video and reliability). The evaluated cost results from the number of accepted requests for a given number of copies. As an example, let us consider the cost for requests having a reliability of 0.93 to a server made of three resources storing the same video. The server handles a set of clients requesting the same reliability (0.93) until a request is denied by the resource manager. In this case, the resource manager accepts $n = 75$ requests. As the number of slots available on this server is $s = 3 * 30$, the average cost is therefore $C = \frac{s}{n} = 1.20$.

The results of our proposal are given by the curves of figure 3. The different curves correspond to the number of available video copies (2, 4 and 6), each resource having 30 slots. The curve for two copies shows the advantages of shared reservation: it provides a wide range of reliabilities leading to different costs. Our approach offers intermediate solutions between usual extreme solutions that are the absence of replication and replication without shared reservation. For similar reliabilities, the cost obtained with four and six copies is better than the one obtained with two copies. This means that the algorithm takes advantage of shared reservation and therefore increases the accessibility of the server while providing reliability on demand.

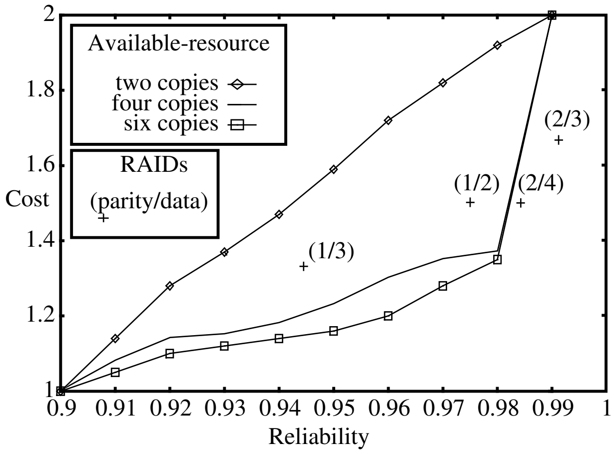

Figure 3. Cost of reliability.

4.3 Comparing our approach with RAID

The work that is the most related to our, deals with specializing RAIDs (Reliable Array of Inexpensive Disks) [Chen 94]. A RAID is based on a set of inexpensive disks which are considered as a single disk. The possibly high number of disks also reduces the reliability of the RAID, since the failure of a single disk may cause the failure of the whole RAID. To increase reliability, RAIDs store parity data which allow to accomodate disk failures. The storage capacity of RAIDs is therefore divided into effective storage and parity data. In fact, the diversity of RAIDs architectures allows to change the number of parity disks and the number of data disks. Hence, similarly to our proposal, these changes provide various reliabilities of different cost. The comparison of this solution with our remains interesting since both have a common goal which is to provide a range of reliabilities of different cost, even if they are concerned with different resources. To compare our solution with RAIDs, we propose to evaluate RAIDs using the same scale. As an example, we evaluate the cost of a RAID managing one parity disk and two data disks. In such a RAID, two of the three disks are effectively used to store data, the other one being used to store redundant data. The cost of reliability is then $C = \frac{3}{2} = 1.5$. The reliability of the above RAID noted (R) depends on the reliability of a disk noted (R_1). The RAID being able to accomodate a single failure, its reliability is $R = R_1^3 + 3 * (1 - R_1) * R_1^2$. Let us define ($R_1 = 0.9$), the cost of various RAID architectures are presented in figure 3. These architectures are denoted by (p/d), where p is the number of parity disks and d is the number of data disks. The cost of our proposal obtained with four and six copies is better than the one of the RAIDs (1/3) and (1/2) providing the same reliabilities. Our placement policy performs really good for reliabilities between 0.90 and 0.98 by taking advantage of shared reservation. We believe that even better performances could be obtained for reliabilities between 0.98 and 1 by implementing 3-resilient redundancy. Also notice that our proposal is able to provide different reliabilities to concurrent requests while a RAID architecture provides the same reliability to all the clients.

5 Conclusion

We argue that one approach to improving reliability in distributed VoD servers without sacrificing accessibility lies in managing together reliability and accessibility. The use of RAIDs achieves this compromise in the case of disk reliability. In this paper, we have proposed a new method whose goal is to achieve this compromise in the case of both CPU and disk reliability. Our method is based on a request placement policy which takes into account both requests' reliability and accessibility needs. We showed that the multitude of CPUs and disks used to serve different clients can also serve ensuring low-cost reliability. This method allows the batching manager to negotiate the required reliability, which is then guaranteed during the whole execution of the request. Results given in section 4 demonstrate that our approach allows to accommodate both reliability on demand and a high accessibility of the server. The accessibility obtained with our method was compared to the one of the RAIDs providing equivalent reliability. We proved that our method is more efficient while providing various reliabilities.

We are currently working on extending the treatment of reliability as a parameter of QoS in multimedia applications. So far, we have used a 2-resilient passive replication method for guaranteeing reliability in order to maximize accessibility. However, one may consider the use of various replication methods depending on the QoS criteria that are favored. For instance, if one wants to favor reliability over accessibility by making fully transparent any failure occurrence, the active replication method should be implemented. Towards that goal, we are examining the choice of the replication mechanism to be implemented with respect to the service's QoS.

References

[Aggarwal 96] C.C. Aggarwal, J.L. Wolf, and P.S. Yu. On optimal batching policies for video-on-demand storage servers. Technical report, IBM research division, T. J. Watson Research Center, March 1996.

[Anderson 93] D.P. Anderson. Metascheduling for continuous media. *ACM Transactions on Computer System*, 3(11), August 1993.

[Berson 94] S. Berson, L. Golubchik, and R.R. Muntz. A fault tolerant design of a multimedia server. Technical report, University of California, Los Angeles, January 1994.

[Billot 96] M. Billot, V. Issarny, I. Puaut, and M. Banâtre. A proposal for ensuring high availability of distributed multimedia applications. In 15^{th} *IEEE symposium on reliable distributed systems*, October 1996.

[Chen 94] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID high performance, reliable secondary storage. *ACM computing survey*, 26(2), June 1994.

[Furht 95] B. Furht, D. Kalra, F.L. Kitson, A.A. Rodriguez, and W.E. Wall. Design issues for interactive television systems. *Computer*, 28(5), May 1995.

[Kandlur 93] D. D. Kandlur, M. Chen, and Z. Shae. Design of a multimedia storage server. Technical report, IBM research center, T. J. Watson Research Center, September 1993.

[Keeton 93] K. Keeton and R.H. Katz. The evaluation of video layout strategies on a high-bandwith file server. In *International conference on network and operating system support for digital audio and video*, November 1993.

[Lee 90] P.A. Lee and T. Anderson. *Fault tolerance - Principles and practice*. Springer - Verlag, 1990.

[Little 93] T.D.C. Little and D. Venkatesh. Probabilistic assigment of movies to storage devices in a video-on-demand system. In *International conference on network and operating system support for digital audio and video*, November 1993.

[Lougher 93] P. Lougher and D. Shepherd. The design of a storage server for continuous media. *The computer journal*, 36(1), January 1993.

[(MPEG) 93] SC29/WG11 (MPEG). Coding of audio, picture, multimedia and hypermedia - Part 1: systems. Draft submitted to ISO-IEC/JTC1 SC29, December 1993.

[Rozier 92] M. Rozier, V. Abrassimov, F. Armand, M. Boule, M. Gien, M. Guillemont, F. Herrman, C. Kaiser, S. Langlois, P. Leonard, and W. Neuhauser. Overview of the Chorus distributed operating system. In *Usenix symposium on micro-kernels and other kernels architectures*, April 1992.

[Wolf 94] J.L. Wolf, P.S. Yu, and H. Shachnai. DASD dancing: a disk load balancing optimization scheme for Video-on-Demand computer systems. Technical report, IBM research division, T. J. Watson Research Center, December 1994.