Design and Analysis of a Fault-Tolerant Mechanism for a Server-Less Video-On-Demand System

Jack Y. B. Lee
Department of Information Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
Email: jacklee@computer.org

Raymond W. T. Leung
Department of Information Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
Email: wtleung0@ie.cuhk.edu.hk

Abstract

Video-on-demand (VoD) systems have traditionally been built on the client-server architecture, where a video server stores, retrieves, and transmits video data to video clients for playback. This paper investigates a radically different approach to building VoD systems, one where the server, and hence the primary bottleneck, is completely eliminated. This server-less architecture comprises homogeneous hosts, called nodes, which serve both as client and as mini-server. Video data are distributed over all nodes and these nodes cooperatively stream video data to one another for playback. However, unlike traditional video server that runs on high-end server hardware in a carefully controlled and protected data centre, a node in a server-less system is likely to be far more unreliable. Therefore it is essential that sufficient data and capacity redundancies are incorporated to maintain an acceptable service reliability. This paper presents and analyzes a fault tolerant mechanism based on inter-node striping and erasure correction codes to tackle this challenge. By formulating the system's reliability as a Markov chain model, we obtain insights into the feasible operating region of the system, such as the amount of redundancy required and the node-level reliability that can be tolerated. Numerical results show that a server-less VoD system of 200 nodes can achieve reliability surpassing that of dedicated video server using a redundancy overhead of only 21.2% even though individual nodes are highly unreliable.

1. Introduction

Current video-on-demand (VoD) systems are commonly designed around the client-server architecture. Under this architecture, a client sends a request to the video server for a video title and then the server transmits video data to the client for playback. As the number of user increases, the server will eventually reach its capacity limit. To further increase the system capacity, one can add more servers and distribute the requests to them, such as parallel server [1] and distributed server [2-3] architectures. As each server serves only part of the users rather than all users, the total system capacity is extended.

Nevertheless, the cost of upgrading servers can be substantial, as video servers typically require high-end server hardware with high I/O bandwidth, large memory capacity, as well as storage capacity. Even in the best case, such as parallel server and distributed server architectures that do not require data replication, the server cost will still increase at least linearly with the traffic demand. Moreover, apart from server cost, the distribution network will also need to be upgraded with more bandwidth to carry the vast amount of video traffic to the users. Given the high cost of long-distance backbone networks, it is no wonder why metropolitan-scale VoD services are still uncommon in practice.

In this study, we take a radically different approach to building scalable, reliable, and cost-effective VoD systems. In particular, we turn our attention to an often-neglected element in a VoD system – the client-side device or commonly called the set-top box (STB).

Developments of STB have continued for many years and current STBs not only are low cost, but also are relatively powerful due to the rapid technological development and economy of scale achieved by the personal computer industry. While early generations of STB are very limited in function and capability, the current trend in STB development is towards evolving from a simple video-receiving and decoding device into a home entertainment center with functions like VoD, TV-over-Internet, harddisk-based personal video recorder, messaging center, web browser, CD player, DVD player, digital audio jukebox, or even game console. This evolution not only greatly enhances the usefulness of a STB, but also opens a radically new way to build VoD systems.

Specifically, we take advantage of the increased storage and processing capability of STBs to build a completely distributed VoD system that does not require dedicated server at all. We call this a server-less architecture for obvious reason. In this server-less architecture, all STBs, or called a node in this paper, in the system serve both as a client and as a mini-server. Video data are distributed among the nodes and multiple nodes work together to serve video streaming requests from other nodes. The beauty of this architecture is that the system is inherently scalable, i.e., when new users are added to the system, they add both streaming load and streaming capacity to the system. Moreover, network costs can also be reduced because the nodes are likely to be clustered together, reducing the need for costly long-distance network backbone.

However, building a server-less VoD system is not without challenges. In a previous study [4], we investigated the issues of data placement, scheduling, and streaming. The architecture has been shown to be feasible with today's hardware platforms. In this study, we investigate another critical issue of the server-less architecture – reliability.

Unlike a client-server VoD system, video data are distributed in a server-less VoD system. While this data placement policy eliminates the storage overhead in replication, such as the case in current peer-to-peer (P2P) systems [5-6], the system cannot function if any of the nodes fails. To tackle this problem, we develop a fault tolerant mechanism based on the use of erasure correcting codes to enable the system to sustain node failures. With this mechanism, we derive the system's mean-time-to-failure (MTTF) using a Markov chain model to find out the amount of redundancy required to achieve a given system MTTF, and to investigate how low the reliability of STB can go before the redundancy overhead becomes too excessive. Numerical results show that a server-less VoD system of 200 nodes can achieve reliability surpassing that of dedicated video server (MTTF of 100,000 hours) using a redundancy overhead of only 21.2% even when individual nodes are highly unreliable (MTTF of 256 hours).

The rest of the paper is organized as follows: Section 2 reviews some previous related works; Section 3 presents an overview of the server-less VoD architecture; Section 4 presents the fault tolerant mechanism; Section 5 presents a model for the system reliability; Section 6 evaluates the system reliability using numerical results; and Section 7 concludes the paper.

2. Related works

Current VoD architectures can be classified into centralized and distributed architectures [7]. In centralized server architectures, only the central server serves user requests and so it becomes the system's primary bottleneck. By contrast, requests are shared by multiple servers in a distributed server architecture such that capacity can be scaled up by adding more servers.

Serpanos, *et al*. [7] compared the performance of centralized and distributed architectures for video servers. They concluded that in general, a centralized architecture is preferable in terms of performance and management, but at the expense of higher cost. To improve cost effectiveness, distributed or parallel server architectures [1-3] are commonly employed. For example, one can replicate video data to multiple servers and equally divide requests between them. To further reduce the storage overhead due to replication, replication can be limited to the more popular video titles.

For example, On, *et al*. [2] studied replication assignment and update frequency in relation to the desired data availability, consistency, and QoS requirements. Serpanos, *et al*. [3] proposed a MMPacking video assignment algorithm based on video popularity to achieve load and storage balance. Another approach is the use of parallel server architectures (see Lee [1] for a review of parallel server architectures) that employ server-level data striping. Compared to replication and caching, parallel server architectures eliminate the need for data replication and are inherently load balanced. Moreover, one can introduce data and hardware redundancies into the system to achieve server-level fault tolerance, making the system even more reliable than central-server designs.

Another area related to our study is the peer-to-peer (P2P) concept popularized by software systems such as Napster [5] and Gnutella [6]. These P2P systems are primary designed to function as a large distributed storage system [8-9]. In a P2P system, a user shares files with a group of other users and can search for the desired files by submitting a query to neighbors or to a directory server. Once the desired files are located, the user then downloads the data directly from the other user's computer. As the data are selectively replicated among user nodes, this structure allows sharing of files by a large community at low cost, as a dedicated server is no longer needed. The main challenge comes from the complexity in distributing replicated files to achieve load balance and fault tolerance [9]. As users in a P2P system have varying network bandwidth and processing capability, quality-of-service cannot be guaranteed and slow or even broken connections are not uncommon. Nevertheless, the ease of setting up and participating in a P2P system and the need for a decentralized file-sharing platform have outweighed these limitations.

Compared to traditional client-server architecture, the server-less architecture distributes the server functions to the clients. This approach not only eliminates the primary bottleneck in the system, but is also inherently scalable. Compared to current P2P systems such as Naptser and Gnutella, the server-less architecture employs distributed data storage rather than extensive file replication to improve availability. This difference is essential as video data consume significantly more storage than MP3 audio files. Moreover, unlike file sharing, VoD applications have stringent performance requirements that are essential to the correct operation of the system. Consequently, the server-less VoD architecture requires completely different data placement policy, retrieval scheduling, transmission scheduling, and fault tolerance mechanism compared to current P2P systems.

3. System architecture

In this section, we present details of the server-less architecture, including the data placement policy and the retrieval and transmission schedulers. A server-less VoD system comprises a pool of user nodes connected by a network as shown in Fig. 1. Each node has its own CPU, memory and disk storage. Inside each node there is a mini video server software that serves a portion of each video title to other nodes in the system. Unlike conventional video server, this mini server serves a much lower aggregate bandwidth and therefore can readily be implemented in today's STBs and PCs. For large systems, the nodes can be further divided into clusters where each cluster forms an autonomous system that is independent from other clusters.

3.1. Data placement policy

As discussed in Section 2, existing distributed systems commonly employ data replication and caching to improve scalability. However, unlike video servers where storage capacity is usually large, a node in the form of a STB or a PC will have relatively limited storage capacity. Therefore, instead of replication, we propose the use of striping as the data placement policy for the architecture.

Specifically, each video title is divided into fixed-size striping units (or called blocks) of Q bytes each. And we assume there exist an archive server that distributes the striping blocks to all nodes in the cluster in a round-robin manner. This node-level striping scheme avoids data replication while at the same time divides the storage requirement equally among all nodes in the cluster.

To initiate a video streaming session, a client node will first locate the set of server nodes carrying blocks of the desired video title, the striping policy and other parameters (format, bitrate, etc.) through the directory service. These server nodes will then be notified to start transmitting the video blocks to the client node. The notification can be performed directly by the client node or indirectly by the directory service, of which the exact mechanism involved is beyond the scope of this study.

3.2. Retrieval and transmission scheduling

Let N be the number of nodes in the cluster and assume all video titles are constant-bit-rate (CBR) and share the same bitrate R_v. For a server node in a cluster, it may have to retrieve video data for up to N video streams, of which N–1 of them are transmitted while the remaining one played back locally. Note that as a video stream is served by N nodes concurrently, each node only needs to serve a bitrate of R_v/N for each video stream.

Many existing video server designs employ round-based schedulers such as SCAN and its variants [10-11]. In our design, we employ the Grouped Sweeping Scheme (GSS) proposed by Yu, *et al.* [12] to schedule a node's disk

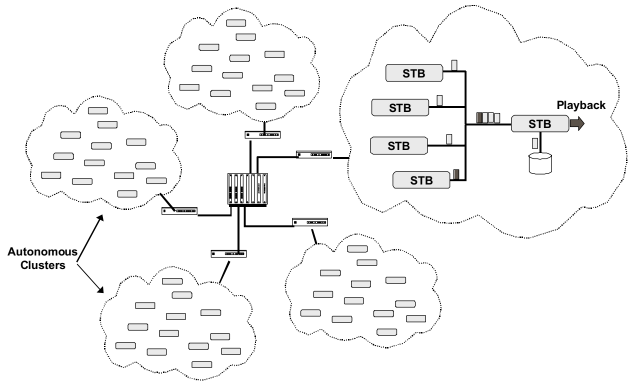

Fig. 1. Architecture of the server-less video-on-demand system.

retrieval and network transmission. Compared to the more common SCAN scheduler that maximizes throughput at the expense of buffer overhead, GSS allows one to control the tradeoff between disk efficiency and buffer requirement. This is a crucial feature as disk throughput may not be the bottleneck in a server-less VoD system. Interested readers are referred to Lee and Leung [4] for more details on the scheduling algorithms.

4. Fault tolerant mechanism

In a server-less VoD system, fault tolerance becomes an essential capability as reliability of STBs and PCs will be significantly lower than dedicated video servers located in a data centre run by professional operators around the clock. Moreover, given the relatively large number of nodes, the system needs to expect and prepare to recover not from a single failure, but from multiple simultaneously failures as well. The following sections present a fault tolerant mechanism to extend the architecture described in Section 3 to sustain node failures.

4.1. Capacity and data redundancy

When a node fails, all data stored in that particular node becomes unavailable. In communications terminology this is called erasure as opposed to error. To recover from data erasures, erasure-correcting codes such as the Reed-Solomon Erasure Correcting (RSE) Code [13,14] can be used. Compared to replication, Weatherspoon *et al*. [15] showed that erasure-resilient systems require an order of magnitude less redundancy overhead to achieve similar reliability. Thus we employ erasure-correcting codes to sustain node failures in the server-less architecture.

Specifically, a (n, h)-RSE codeword comprises n symbols of which (n–h) of them are message symbols (i.e. data) and the remaining h are redundant symbols. One can recover all (n–h) message symbols as long as *any* (n–h) out of the n symbols are correctly received. Let N be the number of nodes in the system. Then by extending the striping-based placement policy in Section 3.1 with a (N, h)-RSE code, the system will have sufficient redundant data for a client node to recover all video data with up to *h* node failures in the cluster. To accommodate the RSE-code, we need to modify the placement policy, the schedulers, and the client node's buffering algorithm. For the placement policy, an additional encoding step will be needed to compute the *h* redundant blocks for each group of (*N*–*h*) video data blocks. Moreover, as now only (*N*–*h*) of the stored data are playable data, we will need to increase the strip unit size from *Q* bytes to

$$Q_r = Q\left(\frac{N}{N-h}\right) \tag{1}$$

bytes to maintain the same data size of a striping group. After the encoding step, the archive server then distributes these *N* blocks (including *h* redundant blocks) to all nodes.

For the disk scheduler, the retrieval unit will be increased from *Q* bytes to Q_r bytes. Transmission rate will also increase from R_v to

$$R_r = R_v\left(\frac{N}{N-h}\right) \tag{2}$$

to maintain the same video bitrate.

4.2. Redundant data transmission policy

Clearly, to support fault tolerant the system has to bear overheads associated with the use of data and capacity redundancy. Apart from the extra storage required to store the redundant data, there will also be overheads in disk retrievals as well as network transmissions, especially when there is no failure in the system.

So far, we have assumed that all redundant data are transmitted at all times, regardless of whether there is any failure in the system. We call this static redundant data transmission for obvious reason. With a (*n*, *h*)-RSE code, the network overhead incurred by redundant data will amount to *h*/(*n*–*h*).

Another possibility would be to transmit redundant data only when needed – dynamic redundant data transmission. Specifically, the system can initially transmit only *k* out of the *h* redundant data blocks in a group to the receiver. Assume there are *f* failed nodes in the system, then the system can survive up to (*k*–*f*) more simultaneous node failures. For every new failure, the system will activate the transmission of another previously dormant redundant block, until all redundant data blocks are transmitted. The network overhead in this case will be reduced to *k*/(*n*–*h*), where $k \le h$.

The downside of dynamic transmission, however, is that the system will need to actively detect node failures and reconfigure the redundant data transmission process once a new node failure occurs. This detection and reconfiguration process obviously takes time, and the system will fail if *k* or more nodes fail before the reconfiguration completes. Therefore these tradeoffs must be carefully evaluated against the resource savings to conclude the suitability of the dynamic transmission approach. We will assume static transmission in this paper and leave the investigation of the dynamic transmission approach to future works.

5. Reliability modeling

In this section we formulate a model for measuring the system reliability with static redundant data transmission. We use mean time to failure (MTTF), defined as the average time between two consecutive occurrences of system failure, as the measure of the system's reliability. Using a (*N*, *h*)-RSE code, the system fails when more than *h* out of the *N* nodes in the system fail simultaneously. We assume nodes fail independently with a MTTF exponentially distributed with a mean λ. A node, once failed, will be repaired immediately and independently. The repair time is also exponentially distributed with a mean μ. Our goal is to derive the relationship between *h* and the system MTTF.

We model the system using a continuous-time Markov chain model as shown in Fig. 2. Let state *i* be the system state where *i* out of the *N* nodes have failed. For example, state 0 means all nodes are operational, state 1 means one of them has failed, and so on. We assume that once a system fails, it will be shutdown and all failed nodes repaired before it is restarted.

Let λ_i be the rate at which the system transits from state *i* to state *i*+1 and μ_i be the rate at which the system transits from state *i* to *i*–1. We can obtain λ_i and μ_i from

$$\lambda_i = (N-i)\lambda \tag{3}$$

$$\mu_i = i\mu \tag{4}$$

Let T_i be the expected time the system takes to reach state *h*+1 from state *i*. Starting from state 0, the expected time to reach state *h*+1 is equal to the expected time to reach state 1 from state 0, i.e., equal to $1/\lambda_0$, plus the expected time to reach state *h*+1 from state 1, i.e., T_1 by definition. Thus we can obtain the following relation:

$$T_0 = \frac{1}{\lambda_0} + T_1 \tag{5}$$

Next we consider state 1. The system may either transit to state 2 with probability $(\lambda_1/(\lambda_1+\mu_1))$, or transit to state 0 with probability $(\mu_1/(\lambda_1+\mu_1))$. The combined transition rate is equal to $(\lambda_1+\mu_1)$ and so the expected time until transition occurs is equal to $1/(\lambda_1+\mu_1)$. Using argument similar to the case of state 0, we can obtain another relation:

$$T_1 = \frac{1}{\lambda_1+\mu_1} + \frac{\lambda_1}{\lambda_1+\mu_1}T_2 + \frac{\mu_1}{\lambda_1+\mu_1}T_0 \tag{6}$$

Repeating this procedure for T_2 to T_{h+1}. We can obtain the following set of equations:

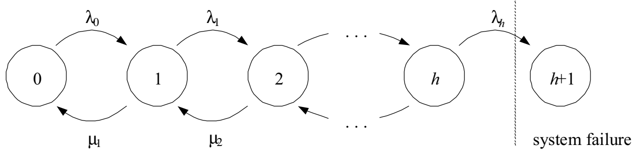

Fig. 2. A Markov chain model.

$$\begin{aligned}
T_1 &= \frac{1}{\lambda_1+\mu_1} + \frac{\lambda_1}{\lambda_1+\mu_1}T_2 + \frac{\mu_1}{\lambda_1+\mu_1}T_0 \\
&\vdots \\
T_h &= \frac{1}{\lambda_h+\mu_h} + \frac{\lambda_h}{\lambda_h+\mu_h}T_{h+1} + \frac{\mu_h}{\lambda_h+\mu_h}T_{h-1} \\
T_{h+1} &= 0
\end{aligned} \tag{7}$$

Substitute (5) into (6), we can eliminate the term T_0 and express T_1 in terms of T_2 only. Repeating this substitution recursively, we can simplify the above equations into the following equations:

$$\begin{aligned}
T_1 &= \frac{1}{\lambda_1} + \frac{\mu_1}{\lambda_1\lambda_0} + T_2 \\
T_2 &= \frac{1}{\lambda_2} + \frac{\mu_2}{\lambda_2\lambda_1} + \frac{\mu_2\mu_1}{\lambda_2\lambda_1\lambda_0} + T_3 \\
&\vdots \\
T_i &= \left(\sum_{j=0}^{i} \frac{\prod_{k=0}^{j-1}\mu_{i-k}}{\prod_{k=0}^{j}\lambda_{i-k}} \right) + T_{i+1} \\
&\vdots \\
T_{h+1} &= 0
\end{aligned} \tag{8}$$

As T_{h+1} is zero, we can obtain the value of T_0 by backward substitution:

$$T_0 = \sum_{i=0}^{h} \left(\sum_{j=0}^{i} \frac{\prod_{k=0}^{j-1}\mu_{i-k}}{\prod_{k=0}^{j}\lambda_{i-k}} \right) \tag{9}$$

of which is the MTTF of the system.

6. Reliability evaluation

High-end video servers operated in a controlled environment such as a data centre typically have MTTF of at least 10,000 hours, or more if hardware redundancies are available (e.g. RAID disk, redundant power supply, etc.). Client-side devices such as STBs and PCs on the other hand, will be far less reliable. Although hardware reliability of STBs and PCs are not too far behind servers, the operating environment is far more hostile. For example, a STB may be shutdown due to power surge, power outage, or simply because the user unplugs the STB to move it to another location.

On the other hand, these non-hardware-induced failures are also fixed more quickly than hardware failures. For example, a STB inadvertently unplugged will stop working and the user will likely fix it immediately if the user is using the device; or fix it the next time the user uses the device. In any case, these frequent breakdowns do not require maintenance service by a professional and thus can be corrected more quickly.

As no data exists for the failure rate and repair rate of STBs, we first investigate the sensitivity of the system's MTTF with respect to the failure rate. The results are computed using the formula derived in Section 5 and the results are plotted in Fig. 3. We can observe that the redundancy overhead increases exponentially when the node MTTF falls below 200 hours, or around eight days, while the node MTTR is 10 hours at a system size of 200 nodes. Compared to consumer electronic devices such as a VCR or a satellite TV receiver, this node-level MTTF should be achievable.

Fig. 4 plots the redundancy overhead versus node mean-time-to-repair (MTTR) while the node MTTF is 256 hours at a system size of 200 nodes. Unlike the MTTF case, we observe that the redundancy overhead increases nearly linearly with longer node MTTR. However with a conservative node MTTR of 32 hours, the redundancy overhead is modest (e.g. 26.6% for achieving system MTTF of 100,000 hours).

To investigate how well the system scales, we assume conservatively a node MTTF of 256 hours and a node MTTR of 24 hours, and then plot in Fig. 5 the redundancy overheads for system sizes ranging from 50 to 500 nodes. There are two observations from the results. First, the redundancy overhead is relatively modest, e.g. 21.2% and 23.5% for a 200-node system achieving system MTTF of 100,000 hours and 1,000,000 hours respectively. While not insignificant, the redundancy over is still significantly lower than replication. The second observation is that the proportion of redundant data required decreases with increase in the number of nodes in the system. This suggests that the redundancy overhead will not become a limiting factor when one scales up a system to more nodes.

7. Conclusions

In this study, we investigated the reliability issue in a server-less VoD system. Unlike conventional VoD systems built around the client-server architecture, individual nodes in a server-less VoD system are far more likely to fail due to non-hardware-induced failures. Using a Markov chain model, we evaluated the architecture's reliability and studied its sensitivity to the nodes' MTTF and MTTR. With the conservatively estimated parameters, we found that one can achieve system reliability surpassing dedicated

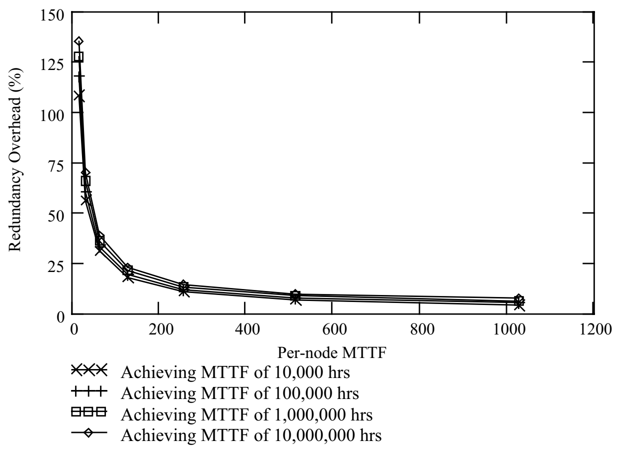

Fig. 3. Redundancy versus node reliability.

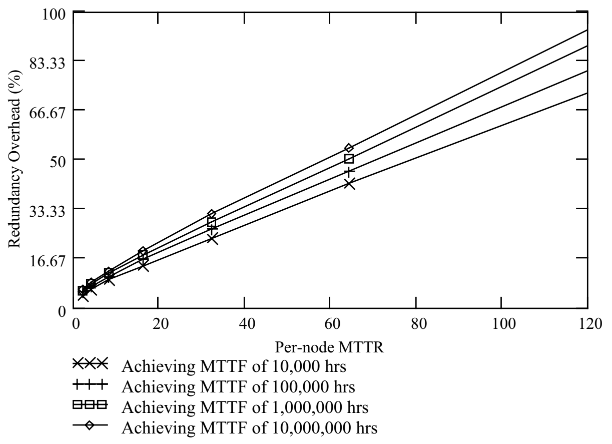

Fig. 4. Redundancy versus node MTTR.

high-end video servers with less than 25% redundancy overhead. This result is encouraging as the amount of redundancy required is significantly lower than the more common replication approaches. Nevertheless, there are still many open issues that warrant further investigations. Two examples are the dynamic redundant data transmission approach discussed in Section 4.2, and automatic video data rebuild for nodes recovering from hardware failure.

8. Acknowledgements

This research is partially funded by research grants (Direct Grant, Earmarked Grants CUHK6095/99E) from the HKSAR Research Grant Council and the AoE-IT, a research grant from the HKSAR University Grants Council.

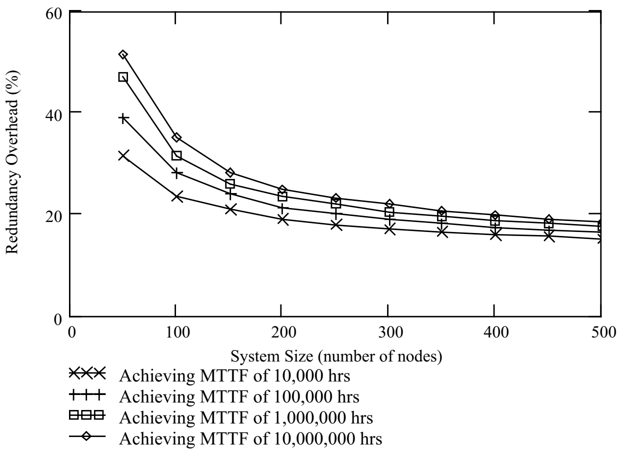

Fig. 5. Redundancy versus system size.

References

[1] J. Y. B. Lee, "Parallel Video Servers: A Tutorial", *IEEE Multimedia*, vol.5(2), April-June 1998, pp.20–28.

[2] G. On, M. Zink, M. Liepert, C. Griwodz, J.B. Schmitt, and R. Steinmetz, "Replication for a Distributed Multimedia System" *Proceedings of the Eighth International Conference on Parallel and Distributed Systems*, 2001, pp.37–42.

[3] D. N. Serpanos, L. Georgiadis, and T. Bouloutas, "MMPacking: A Load and Storage Balancing Algorithm for Distributed Multimedia Servers", *IEEE Transactions on Circuits and Systems for Video Technology*, vol.8(1), Feb 1998, pp.13–17.

[4] J. Y. B. Lee and R. W. T. Leung, "Study of a Server-less Architecture for Video-on-Demand Applications", *Proceedings of the IEEE International Conference on Multimedia and Expo 2002*, Lausanne, Switzerland, Aug 2002.

[5] Napster. http://www.napster.com.

[6] Gnutella. http://gnutella.wego.com.

[7] D. N. Serpanos and A. Bouloutas, "Centralized versus Distributed Multimedia Servers", *IEEE Transactions on Circuits and Systems for Video Technology*, vol.10(8), Dec 2000, pp.1438–1449.

[8] M. Parameswaran, A. Susarla, and A. B. Whinston, "P2P Networking: An Information Sharing Alternative", *Computer*, vol.34(7), July 2001, pp.31–38.

[9] G. Fox, "Peer-to-peer networks", *Computing in Science & Engineering*, vol.3(3), May-June 2001, pp.75–77.

[10] A. L. N. Reddy and J. C. Wyllie, "I/O Issues in a Multimedia System", *Computer*, vol.27(3), March 1994, pp.69–74.

[11] D. J. Gemmell, H. M. Vin, D. D. Kandlur, P. V. Rangan, L. A. Rowe, "Multimedia Storage Servers: A Tutorial", *Computer*, vol.28(5), May 1995, pp.40–49.

[12] P. S. Yu, M. S. Chen, and D. D. Kandlur, "Grouped Sweeping Scheduling for DASD-based Multimedia Storage Management", *ACM Multimedia Systems*, vol.1(2), 1993, pp.99–109.

[13] S. B. Wicker, *Error Control Systems for Digital Communication and Storage*, Englewood Cliffs, NJ: Prentice-Hall, 1995, pp.227–234.

[14] A. J. McAuley, "Reliable Broadband Communication Using a Burst Erasure Correcting Code", *Proceedings of the ACM SIGCOMM 90*, Philadelphia, PA, September 1990, pp. 287–306.

[15] H. Weatherspoon and J. D. Kubiatowicz, "Erasure Coding vs. Replication: A Quantitative Comparison", *Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, Mar 2002.