A Server-less Architecture for Building Scalable, Reliable, and Cost-Effective Video-on-demand Systems

Raymond W. T. Leung and Jack Y. B. Lee
Department of Information Engineering
The Chinese University of Hong Kong

ABSTRACT

Video-on-demand (VoD) systems have traditionally been built around the client-server architecture, where a video server stores compressed video for delivery to clients connected by a network. With increasing demand for large-scale VoD systems, researchers have spent considerable effort in designing scalable, reliable, and cost-effective video servers. Nevertheless, a video server can only have finite capacity. As the system scales up, the server will need to be upgraded and this can become very expensive as the system scales beyond thousands of users. In this study, we investigate a radically different architecture where the bottleneck – video server, is eliminated altogether. Specifically, this peer-to-peer, server-less architecture relies on the client machines for distributed data storage and delivery. A client initiating a new streaming session will first locate other clients where the requested stream is stored, and then requests delivery of the stream directly from those clients instead of from a central server. This fully distributed architecture is inherently scalable as the storage and delivery capacity grows with the number of clients in the system. Additionally, we develop fault-tolerance algorithms for the system so that stream delivery can be maintained even if some of the clients fail.

I. 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 [6-10] and distributed server [3-5] 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 large-scale, 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 systems 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. First, one needs a placement policy to determine how to distribute the video data among the nodes. The goals are to increase load balance among the nodes, to increase system-level reliability, to increase the scalability of the system, and so on. Second, one needs an algorithm to schedule the retrieval and transmission of video data for a video streaming session. Third, as video data are distributed among many nodes, the system will need to provide a directory service for a node to locate and request data for a new stream. Fourth, as nodes in the system are more loosely controlled than dedicated video servers, the system will need to adapt to changes in the system configurations such as addition of new nodes and deletion of existing nodes.

In this study, we address the first two of the above-mentioned challenges and leave the challenge of designing a directory service and system adaptation for future work. In the rest of the study, we assume that a node always knows the location of the data it needs and the system configuration never changes. The rest of the paper is organized as follows: Section II reviews related works on large-scale video-on-demand systems; Section III presents the proposed server-less VoD architecture; Section IV evaluates performance of the architecture using numerical results; Section V discusses the scalability issue; and Section VI summarizes the paper.

II. Related Works

In this section, we first review in Section II-A a number of previous related works and then compare them with our approach in Section II-B.

A. Previous Works

Current VoD architectures can be classified into centralized [1-2] and distributed [3-12] architectures. 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*. [2] 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 [3-5] or parallel [6-10] server architectures 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*. [3] studied replication assignment and update frequency in relation to the desired data availability, consistency, and QoS requirements. Serpanos, *et al*. [5] proposed a MMPacking video assignment algorithm based on video popularity to achieve load and storage balance. In a related work by Chuang, *et al*. [4], performance differences between caching and replication are studied with respect to quality-of-service (QoS) requirement, storage requirement and traffic profile. Another approach is the use of parallel server architectures studied by Lee [6-7], Bernhardt, *et al*. [8], Wu and Shu [9], and Tewari, *et al*. [11] that employ data striping at the server level. 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 [10-12], 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 [14] and Gnutella [15]. These P2P systems are primary designed to function as a large distributed storage system [16-17]. In a P2P system, a user shares data with a group of other users and searches for the desired data by submitting the query to neighbors or to a directory server. Once the desired data are located, the user downloads the data directly from the other user's computer. As the data are selectively replicated among user nodes, this structure allows sharing of data 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 data to achieve load balance and fault tolerance [17]. As the 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 data-sharing platform have outweighed these limitations.

B. Contributions of this Study

First, compared to the traditional client-server architecture, our approach decentralizes and distributes the server functions to the clients. This server-less architecture not only eliminates the primary bottleneck in the system, but also is inherently scalable.

Second, compared to the current P2P systems such as Naptser and Gnutella, the server-less architecture investigated in this study serves completely different applications, i.e. video-on-demand versus file sharing. In particular, 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 the current P2P systems.

To the best of the authors' knowledge, our study is the first to address issues in building server-less VoD systems with a goal to provide a service comparable to or even better than existing client-server-based VoD architectures. In particular, we address the issue of data placement policy and presented a striping-based algorithm for video data placement. New retrieval and transmission schedulers are then developed for this data placement policy. To address the issue of reliability, we incorporate the use of data and node redundancies into the system architecture and extend the scheduling algorithms to account for fault tolerant operation. Through numerical results, we show that building VoD systems using the presented server-less architecture is feasible and it can achieve performance comparable to client-server-based systems despite not having dedicated high-end video servers.

III. ARCHITECTURE

In this section, we present details of the server-less architecture, including the data placement policy, the retrieval and transmission schedulers, and the fault tolerant mechanism. A server-less VoD system comprises a pool of user nodes connected by a network as shown in Fig. 1. Each node has its individual 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.

This clustering architecture serves three purposes. First, the system designer can group nodes according to their geographical locations or network topology to localize network traffic. Second, by dividing nodes into clusters, one can limit any systematic failures such as software bugs to a single cluster and prevent bringing down the whole system. Third, by controlling the cluster size one can balance storage overhead (due to data replication across clusters) against processing complexity (e.g. directory service, synchronization, etc.) when one scales up the system. In this study, we focus on the architecture and performance of a single cluster and leave the clustering problem to future works.

A. Data Placement Policy

As discussed in Section II-A, distributed systems often 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. These blocks are then distributed to all nodes in the cluster in a round-robin manner as shown in Fig. 2. This node-level striping scheme avoids data replication while at the same time divides the storage requirement equally among all nodes in the cluster[1]. Note that we have assumed, for simplicity, that striping is done across all nodes in the cluster. In general one can also use partial striping where the striping group comprises only a subset of the 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.

B. 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 [21]. In our design, we employ the Grouped Sweeping Scheme (GSS) proposed by Yu, *et al*. [20] to schedule a node's disk 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 (c.f. Section IV-C).

In GSS, streams are divided into g groups in which retrievals within a group are scheduled using SCAN. The groups are served in a round-robin manner as shown in Fig. 3 for a GSS with two groups.

---

[1] Strictly speaking small storage imbalance can still exist if one always starts placing the first block of a video title to the same node (e.g. node 0). This is because the total number of units of a video title may not be exact multiples of the number of nodes in the system and therefore the last striping group may have fewer strip units than the number of nodes in the system, leading to a storage requirement difference of one stripe unit. This

We call the period of serving a group a micro round and the period of serving all groups once a macro round. If one set g=1 and g=N then GSS reduces to SCAN (higher throughput, larger buffer requirement) and FCFS (lower throughput, smaller buffer requirement) respectively. Intermediate value of g can be used to tradeoff disk efficiency with buffer requirement.

Both micro and macro rounds have fixed duration, denoted by T_g and T_f respectively. Given a data block size of Q bytes, up to N/g data blocks will be retrieved in a micro round using the SCAN scheduler. These retrieved data blocks will then be transmitted at a rate of R_v/N for a duration of T_f seconds, which precisely equals to g micro rounds. Therefore the g groups are effectively staggered in time and this reduces the combined buffer requirement [20].

Knowing the video bitrate, we can compute the micro and macro round duration for configuring the GSS scheduler as follows:

$$T_f = gT_g = \frac{NQ}{R_v} \tag{1}$$

C. Fault Tolerance

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.

When a node fails, all data stored in that particular node becomes unavailable. In communications terminology this is called data erasure. To recover from data erasures, erasure-correcting codes such as the Reed-Solomon Erasure Correcting (RSE) Code [18-19] can be used. 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.

By extending the striping-based placement policy in Section III-A 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

---

storage imbalance can easily be solved by varying the striping order for different video titles and so we will ignore this minor complication in the rest of the paper.

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

$$Q_r = Q\left(\frac{N}{N-h}\right) \tag{2}$$

bytes to maintain the same data size of a striping group.

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

$$R_r = R_v\left(\frac{N}{N-h}\right) \tag{3}$$

to maintain the same video bitrate.

IV. Performance Evaluation

In this section, we evaluate the system requirements and performance of the server-less VoD architecture presented in Section III. We constructed a performance mode for the designs and algorithms in Section III to compute numerical results for evaluation. Due to space limitation, derivations of the performance model are omitted.

A. Storage Capacity

Under the striping-based placement policy, the storage requirement is equally shared by all the nodes in the cluster. Hence, the more nodes in a cluster, the less storage is required on each node to store the same amount of video data. Therefore the storage requirement imposes a lower limit on the scale of a cluster.

For example, assuming a video bitrate of 150 KBps and a video length of 2 hours, the total storage for 100 videos is 102.9 GB. Given that today's harddisks have at least tens of gigabytes of storage capacity, even if each node only allocates 1 GB for video storage, the minimum cluster size needed is still only 108 nodes including redundancy of 5 nodes to achieve a system mean time to failure (MTTF) of 1000 hours.

B. Network Capacity

One way to connect nodes together to form a cluster is to use a switch-based network such as switched Ethernet. Today's medium-range Ethernet switches typically has switching capacity of 32 Gbps or more while carrier-class switches easily exceeds 192 Gbps. Assuming a video bitrate of 150 KBps and ignoring protocol overhead, a node will consume 149.25 KBps both upstream and downstream without redundancy, and 154.66 KBps with a redundancy of 7 nodes for a cluster size of

200 nodes. For the cluster size larger than 200 nodes, a node will consume less than 154.66 KBps in the case of redundancy because the system can achieve the same system reliability with less redundancy. Suppose we use 154.66 KBps as the upper bound on bandwidth consumption for cluster size larger than 200 nodes. Then with a 32 Gbps switch running at 60% utilization, this translates into a maximum cluster size of 8124 nodes. Upgrading it to a 192 Gbps switch will extend the cluster size limit to 48744 nodes.

On the other hand, the server-less architecture does require more bandwidth for the last-mile access network. In particular, the architecture requires more upstream bandwidth than traditional client-server-based VoD systems, which require little to no upstream bandwidth. This requirement will rule out some access network with asymmetric bandwidth such as ADSL or cable modem. Nonetheless, the emerging Ethernet-based access network running at 10Mbps or even 100Mbps full-duplex will provide more than sufficient bandwidth to accommodate a server-less VoD system.

C. Disk Access Bandwidth

A node obviously will have finite resources, including memory and disk access bandwidth. We first investigate the disk access bandwidth issue as it determines the configuration of the disk scheduler (GSS), which in turn affects the buffer requirement.

The disk scheduler has two configurable parameters, namely block size Q and the number of groups g in GSS. Increasing the block size or decreasing the number of groups in GSS results in higher disk efficiency, at the expense of larger buffer requirement and longer system response time. Therefore in terms of buffer requirement and system response time, it is desirable to reduce the block size Q and to increase the number of groups g in GSS.

We first formulate the disk model. The time for serving a request retrieval is composed of four components, namely fixed overhead denoted by α, seek time denoted by t_{seek}, rotational latency denoted by $t_{latency}$ and a transfer time equal to Q/r, where r is the disk transfer rate:

$$t_{request}(Q) = \alpha + t_{seek} + t_{latency} + \frac{Q}{r} \tag{4}$$

Under GSS, retrieval requests are divided into g groups, each served in a micro round using the SCAN scheduler. Under the worst-case scenario, the maximum time to retrieve k requests in a micro round can be computed from

$$t_{round}(k,Q) = k\alpha + t_{seek}^{\max}(k) + k\left(W^{-1} + \frac{Q}{r_{\min}}\right) \tag{5}$$

where $t_{seek}^{\max}(k)$ is the worst-case combined seek time for k requests, W^1 is the time for one complete round of disk platter rotation, and r_{min} is the minimum disk transfer rate[2].

To ensure correct operation, retrievals for these k requests must complete within the duration of a micro round [22]. Therefore the following condition must be satisfied:

$$t_{round}\left(\frac{N}{g}, Q\right) \leq \frac{T_f}{g} \tag{6}$$

Note that for simplicity, we have assumed that the disk is reserved for use by the VoD application and is not shared with other applications such as web browser or messaging application. The model can be extended to support other applications by reserving disk bandwidth in each micro round. Further investigation will be needed to study the issue in more detail.

Now to determine the parameters Q and g, we take an off-the-shelve harddisk – Quantum Atlas 10K [23] for evaluation using parameters listed in Table 1. Using a block size of 4KB and setting g=N, the computed retrieval time is 0.017 seconds while the micro-round time is 0.027 seconds. Hence the condition in (6) is satisfied even if we reduce GSS to the least efficient special case of FCFS with g=N. Moreover, under FCFS the same result will hold for any cluster size because both the micro round length and the retrieval time are linearly proportional to the cluster size N (c.f. (5) and (6)).

D. Buffer Requirement

Fig. 4 plots the buffer requirements versus the cluster size. Redundancy is included to maintain the system MTTF at 1000 hours. Based on results from the previous section, we set g=N, effectively reducing the GSS scheduler to FCFS. With a block size of 4KB, the total buffer requirement is 2.376 MB at the cluster size of 200 nodes. Given the current cost of memory, this buffer requirement, while not insignificant, should be practical for STBs and ordinary PCs.

E. System Response Time

Fig. 5 plots the system response time, scheduling delay, and prefetch delay, versus the number of nodes in the cluster at 90% system utilization. The prefetch delay increases with the increase in the cluster size while the scheduling delay levels off for cluster size larger than 100 nodes. Comparing scheduling delay with prefetch delay, we can easily see that the response time is dominated by prefetch delay. At a cluster size of 200 nodes, the system response time is a reasonable 5.615 seconds.

---

[2] Modern disks commonly use disk zoning, i.e., varying track size, to increase storage capacity. This results in varying disk transfer rate depending on the zone. r_{min} refers to the lowest transfer rate among all zones.

V. Cluster Scalability

Results from the previous section reveal several characteristics of a server-less VoD system. First, within the system parameters used, the system scalability is not limited by network or disk bandwidth. The disk in particular has more than enough bandwidth even when used with the least efficient FCFS scheduler. This property is in sharp contrast to conventional VoD systems built around the client-server architecture, where maximizing I/O efficiency is of primary importance.

Second, the primary limit on the scalability of a cluster is, surprisingly, the system response time. This is a result of the striping-based placement policy, where a striping group must be completely received before it can be erasure-corrected for playback. The size of the striping group, and consequently the prefetch delay, increases linearly with the number of nodes in the cluster. While one can further reduce the block size to reduce the response time, protocol overhead will become significant and so ultimately limits the size of a cluster.

Nevertheless, we do not need to increase the cluster size unless storage capacity is insufficient. As results from Section IV-A show that the lower limit on the cluster size is 108 nodes, one can divide a large system into multiple clusters of say, 200 nodes each, and maintain the system response time to a reasonable 5.615 seconds. As these clusters are autonomous and independent from one another, one can continue forming new clusters to expand the scale of the system.

VI. Summary

In this study, we presented a server-less VoD architecture and proposed designs for data placement policy, retrieval and transmission schedulers, and fault tolerance mechanism. Based on an off-the-shelve harddisk model, we found that a cluster's scalability is surprisingly not limited by I/O performance, but rather limited by the system's response time. Nevertheless, one can expand the system scale simply by forming multiple clusters and there is no limit to which the system can grow.

This study is only the first step taken to establish some fundamental properties of a server-less VoD architecture. There are many other challenges that need more investigations. For example, nodes in a cluster are assumed to be clock-synchronized. This simplifies the analysis but clearly is not the case in practice. Distributed clock-synchronization protocols exist [13] but clock jitter among nodes will never be zero. Thus the impact of clock jitter on the system's performance will need to be quantified.

On the other hand, there are also many other possibilities in the design of a server-less VoD system. For example, this study assumes that clusters are autonomous and independent. Relaxing this assumption to allow clusters to communicate with one another will open many more design choices

for data placement policy, fault tolerant mechanism, and so on. Within a cluster, one can also use different striping policies for different video titles. For example, it may be desirable to use smaller striping group for more popular video titles to increase their availability and also at the same time to reduce response time. Again, more investigations are needed to quantify the performance gain and the tradeoffs.

REFERENCES

[1] A. Krikelis, "Scalable multimedia servers", *IEEE Concurrency*, vol.6(4), Oct.-Dec. 1998, pp.8–10.

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

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

[4] John C. I. Chuang, Marvin A. Sirbu, "Distributed Network Storage with Quality-of-Service Guarantees", http://www.ini.cmu.edu/~sirbu/pubs/99251/chuang.htm.

[5] D. N. Serpanos; L. Georgiadis; 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.

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

[7] J. Y. B. Lee, "Concurrent push-A scheduling algorithm for push-based parallel video servers", *IEEE Transactions on Circuits and Systems for Video Technology*, vol.9(3), April 1999, pp.467–477.

[8] C. Bernhardt and E. Biersack, "The server array: A scalable video server architecture", *High-Speed Networks for Multimedia Applications*. Norwell, MA: Kluwer, 1996.

[9] M. Wu and W. Shu, "Scheduling for large-scale parallel video servers", in *Proc. 6th Symp. Frontiers of Massively Parallel Computation*, Oct. 1996, pp.126–133.

[10] J. Y. B. Lee, "Supporting server-level fault tolerance in concurrent-push-based parallel video servers", *IEEE Transactions on Circuits and Systems for Video Technology*, vol.11(1), Jan. 2001, pp.25–39.

[11] R. Tewari, D. M. Dias, R. Mukherjee, and H. M. Vin, "High availability in clustered multimedia servers" in *Proc. 12th Int. Conf. Data Engineering*, 1996, pp.645–654.

[12] J. Gafsi, E. W. Biersack, "Modeling and performance comparison of reliability strategies for distributed video servers", *IEEE Transactions on Parallel and Distributed Systems*, vol.11(4), April 2000, pp.412–430.

[13] D. Mills, "Internet time synchronization: The network time protocol", *IEEE Trans. Commun.*, vol.39, Oct. 1991, pp.1482–1493.

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

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

[16] M. Parameswaran, A. Susarla, and A. B. Whinston, "P2P networking: an information sharing alternative", *Computer*, vol.34(7), July 2001, pp.31–38.

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

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

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

[20] 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.

[21] A. L. N. Reddy, J. C. Wyllie, "I/O issues in a multimedia system", *Computer*, vol.27(3), March 1994, pp.69–74.

[22] 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.

[23] G. Ganger and J. Schindler, "Database of Validated Disk Parameters for DiskSim", http://www.ece.cmu.edu/~ganger/disksim/diskspecs.html.

Table 1. Parameters used in numerical analysis in Section IV.

Parameters	Symbol	Value
Node mean time to failure	$1/\lambda$	1000 hours
Node mean time to repair	$1/\mu$	10 hours
Video block size	Q	4096 Bytes
Video bitrate	R_v	150 KB/s
Fixed overhead (Head switching time)	α	0.176ms
Rotational latency	W^{-1}	5.99 ms
Minimum disk transfer rate	r_{min}	18.68 MB/s

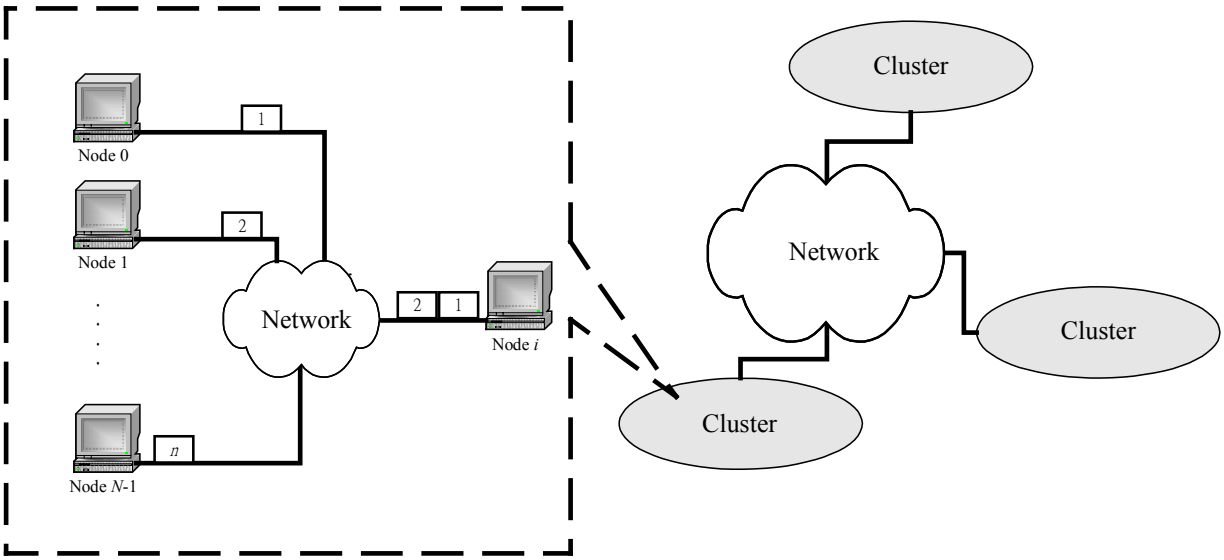

Fig. 1. Architecture of a server-less VoD system.

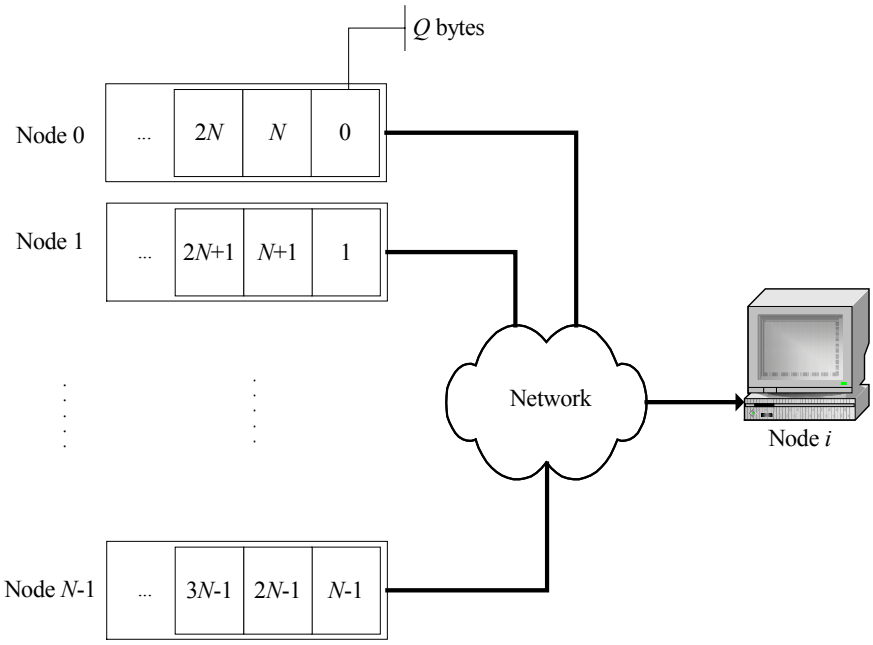

Fig. 2. A striping-based data placement policy.

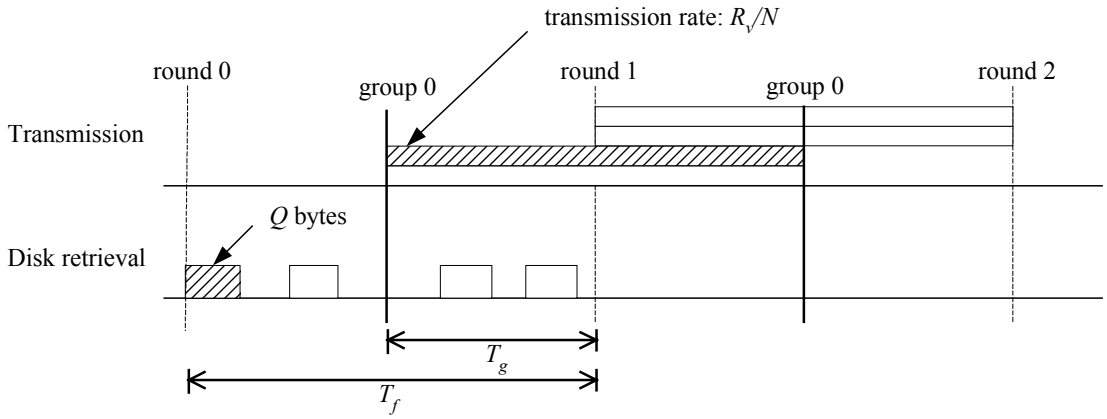

Fig. 3. The Grouped Sweeping Scheme (shown with two groups).

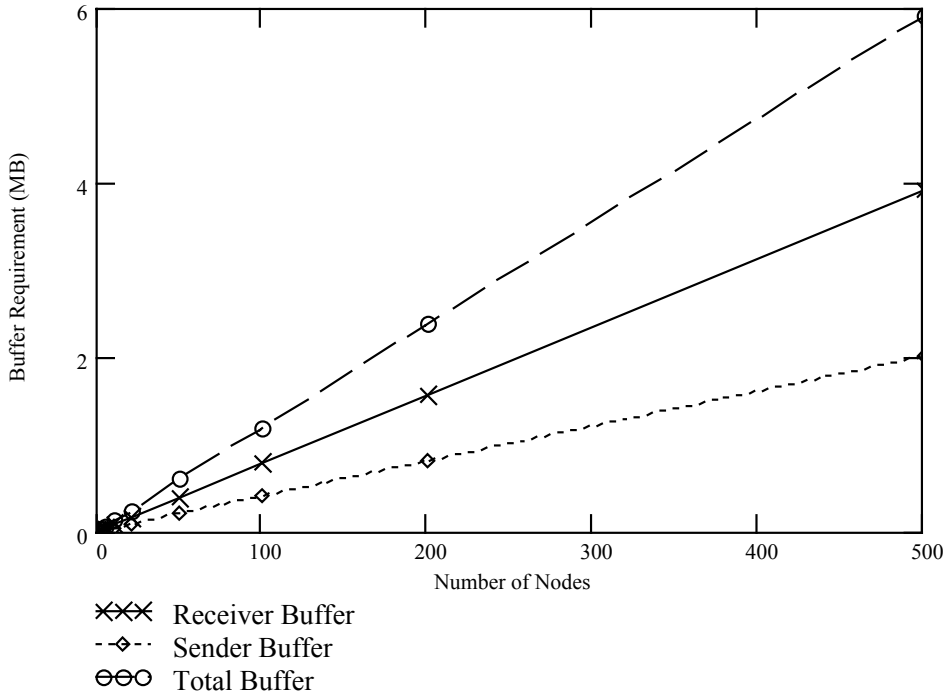

Fig. 4. Buffer requirements versus cluster size (Q=4KB, g=N).

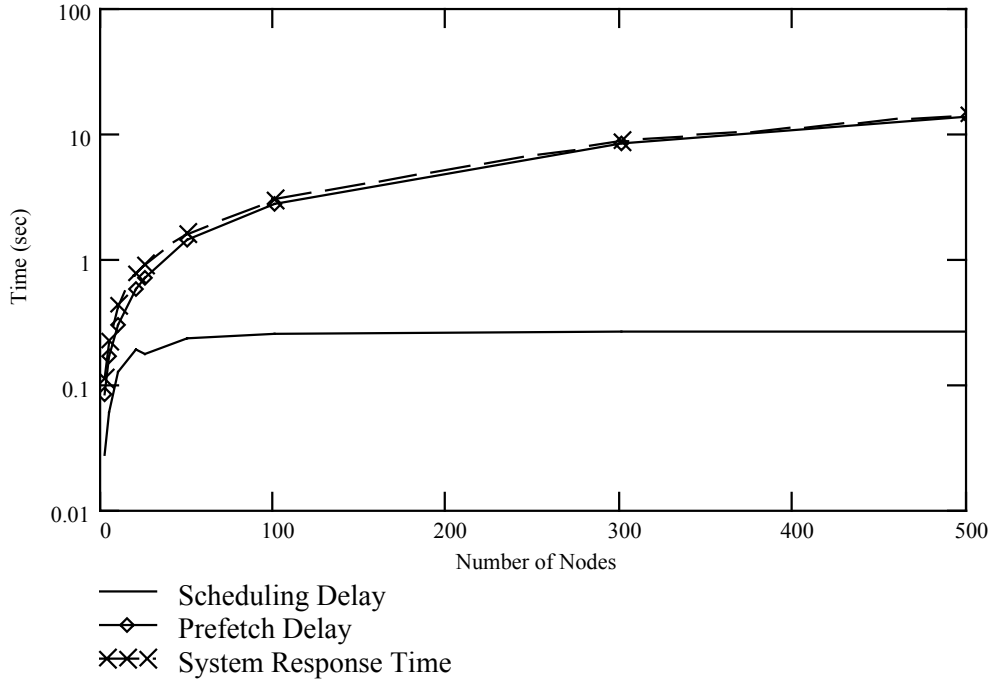

Fig. 5. System response time, scheduling delay, and prefetch delay versus cluster size (Q=4KB, g=N, 90% utilization).