Scheduling Video Streams in a Large-scale Video-On-Demand Server

Jinsung Cho and Heonshik Shin

Department of Computer Engineering, Seoul National University
Seoul 151-742, Korea

Abstract

This paper addresses the design problems concerning a large-scale, parallel video-on-demand server that consists of multiple clusters of nodes connected by a high performance interconnection network. In order to efficiently control the flow of video streams, we propose two scheduling algorithms for data retrieval and communication. First, we present a disk scheduling algorithm called round scheduling which fully utilizes disk bandwidth, minimizing the disk idle time while the server retrieves data blocks. Second, a communication scheduling algorithm is developed to guarantee conflict-free communication over the multistage interconnection network that is topologically equivalent to the Omega network. We also show some simulation results on the server configuration. Analysis of tradeoffs between the server utilization and the startup latency helps to determine the proper number and size of server clusters for a set of given nodes.

Keywords: multimedia, video-on-demand server, server cluster, round scheduling, communication scheduling

1 Introduction

Recent advances in computer technology and demands of video, audio, and text integration services have provided driving forces behind the emergence of various multimedia applications including video-on-demand(VOD) services. The realization of VOD services requires the development of the servers that support efficient mechanisms for storing and delivering video streams to many clients. The fundamental problem in developing such servers is that the delivery and playback of video streams must be performed at real-time rates. Many prototype VOD servers have been developed[2–4]. They have focused on the effectiveness of resource utilization such as disk and memory, while guaranteeing the continuity of streams.

Several telephone companies are planning to provide VOD services to the public over telephone lines[1]. For the purpose of these large-scale service systems, the server should store thousands of movies and serve tens of thousands of concurrent clients. Assuming that MPEG-1 video streams require a playback rate of 0.5 MBps, a 100-minute long movie requires 3 GB, and two thousand movies require a capacity of 6 TB, or 1,200 disks of 5 GB. If we exploit the parallelism of disks in such a system and assume that the effective bandwidth of a disk is 6 MB/s, 14,400 clients can be serviced simultaneously. The large-scale server consists of storage nodes which store and provide video data, and network nodes which deliver data to clients in a timely fashion. Communication between nodes demands high bandwidth interconnections.

In this paper, we address the problems in designing a large-scale VOD server which consists of a large number of nodes connected by a high performance interconnection network. The design problems narrow down to how to cluster such nodes and how to distribute and schedule movies, in the server. In order to efficiently control the flow of video streams, we propose two scheduling algorithms for data retrieval and communication. First, we present a disk scheduling algorithm, *round scheduling*, which fully utilizes the disk bandwidth of storage nodes. Second, we develop a communication scheduling algorithm over Omega interconnection network. Exploiting our communication scheduling algorithm, any pair of nodes in the server can communicate with each other without conflict. Our algorithm results in the balanced communication pattern generated by the server. We also show some analytical results on the configuration of a server for given nodes.

Recent works[1,2] have tackled the problems of designing and implementing VOD servers which consist of multiple nodes. Reddy[1] has proposed a simple movie distribution and movie scheduling algorithm in a multiprocessor video server. The proposed solution minimizes the contention for links over the interconnection network, but cannot always guarantee conflict-free communication. He does not consider the effective management of the disk bandwidth and assumes the homogeneous set of streams which contains the same playback rate streams. In general, however, clients are likely to request heterogeneous streams of which the playback rates are different from each other. On the other hand, in [2], assuming a fiber optic cross-point switch in which there is no scheduling problem, they have evaluated the effect of parameters used in disk scheduling. We now aim at these problems in this paper. In addition, their work[1,2] bounds the scalability of the server in terms of the number of nodes. We analyze the configuration of a server for a large number of nodes.

The remainder of this paper is organized as follows: Section 2 describes our system model and architecture for a large-scale VOD server. In Section 3, we present our disk and communication scheduling algorithms which fully utilize the server resources. Some numerical results for the performance analysis of

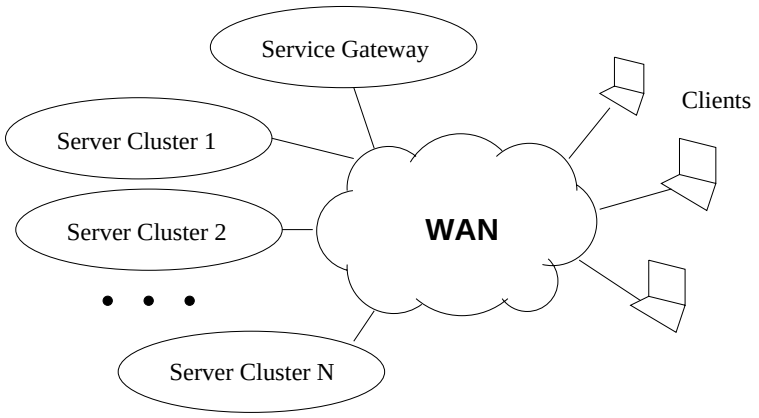


Fig. 1. The configuration of VOD system. *Server nodes are grouped into clusters which serve clients with movie streams independently.*

our server are presented in Section 4, and Section 5 concludes this paper.

2 System Model

In many parallel/distributed VOD servers[2,5], each video stream is divided into logical blocks, which are then distributed among multiple storage nodes (referred to as *data-striping*). Data-striping implicitly achieves higher disk bandwidth and load balancing[2]. In a small-scale distributed server, however, a video stream is stored and serviced in a single storage node[5] (referred to as *no-striping*).

Although the data-striping technique has the aforementioned advantages, it has the following disadvantages: First, a distributed scheduling among storage nodes is required. This imposes clock synchronization problems among all nodes in the server. Second, start-up latency is relatively larger than the no-striping case on account of scheduling problems. Start-up latency means the time elapsed since a request is made to initiate a new stream until the stream is serviced. Third, data-striping lacks scalability. If disks or storage nodes are added to the server, the whole data must be redistributed among the disks. Finally, the popularity of video streams cannot be considered. Since there is no improvement in performance when more than one copy of the same movie is placed on a disk[7], replicating popular movies in a server fails to increase the number of clients that can be serviced simultaneously. Hence, a hybrid technique of data-striping and no-striping is required.

For a given set of nodes, we divide it into server clusters as shown in Fig. 1. Considering the advantages of data-striping, video streams are striped across all the storage nodes in a server cluster. Video streams are allocated and replicated among server clusters with respect to their popularity. Server clusters

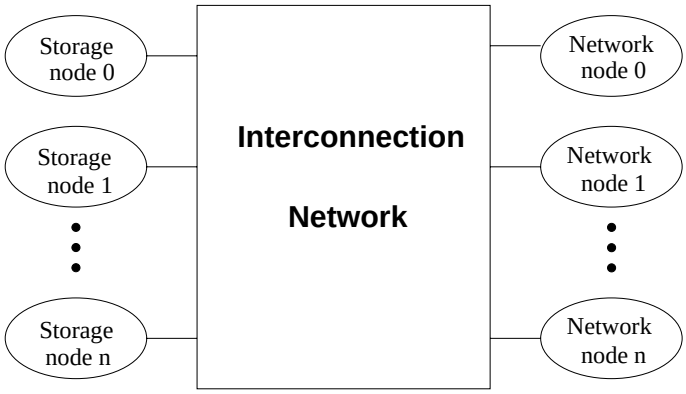


Fig. 2. The architecture of a server cluster. *Each stream is striped across all storage nodes and delivered to a client via network node.*

provide movie streams for clients independently. A series of the movies stored in server clusters is serviced at a service gateway. Considering the load balance among server clusters, the gateway provides clients with the address of the server cluster that stores the requested movie. For a given set of nodes, the appropriate number of server clusters or the number of nodes within a server cluster (the size of a server cluster) will be described in Section 4. If the size of a server cluster equals the number of nodes in the server, video streams are striped across the server, that is, data-striping occurs in the server. On the other hand, if the size of a server cluster is equal to one, no-striping scheme is employed in the server.

Fig. 2 shows the architecture of a server cluster which consists of *storage nodes* and *network nodes*. Storage nodes are responsible for storing video data and delivering the required bandwidth to this data while network nodes are for delivering data blocks from storage nodes to clients. Each request stream would originate at one of the network nodes in the server cluster. This network node should deliver the video stream without violating the continuity requirement of the stream. We consider Omega interconnection network[10] as the communication network between storage nodes and network nodes. However, our scheduling algorithm is applicable to other multistage interconnection networks which are topologically equivalent to Omega network[11]. Although storage and network functions can reside on the same node by connecting node i to input i and output i of the network, we will treat storage nodes and network nodes separately in the rest of this paper. The next section focuses on scheduling algorithms in a server cluster.

3 Scheduling Algorithms

Our VOD server architecture[1] imposes the following problems: data distribution and disk scheduling at storage nodes, communication scheduling between storage nodes and network nodes, and admission control for deterministic service guarantee. In the following subsections we discuss these issues.

3.1 Data distribution

Data distribution refers to distributing the blocks of movies across the storage nodes. This involves the order in which the blocks are striped across the storage nodes. Data organization determines the bandwidth available to a movie, load balance across the storage nodes, and communication patterns. Successive blocks of a video object may be allocated to storage nodes either using a round-robin or a random placement algorithm[2]. With random placement, successive blocks are placed on storage nodes using a random permutation. Although the random placement technique adapts to incremental growth, it may require more meta-data and cause the load of storage nodes to be unbalanced. On the other hand, the round-robin placement scheme places successive blocks of a video stream on consecutive storage nodes (see Fig. 3) and allows the streams to access storage nodes deterministically. However, this could cause large start-up latency if start blocks of movies are placed at the same storage nodes. Distributing the starting points of movies across the storage nodes decreases the average start-up latency.

We now analyze the worst case start-up latency. As an illustration, suppose three clients request movies A, B, and C respectively and that storage nodes 0, 1, and 2 serve the first blocks of A, B, and C, respectively, as shown in Fig. 3. After the playback time T_{play} of a disk block, storage nodes 1, 2, and 3 schedule the next block. If the fourth client requests movie D at this moment, the schedule for block D.0 will be delayed until storage node 3 is idle; so the loads are balanced across the storage nodes. Hence, the scheduling penalty is $3T_{play}$. Supposing there are n storage nodes, the worst case start-up latency will be modeled as follows:

$$(n-1)T_{play} + T_{read}(1) + T_{comm} + T_{net} \;\leq\; T_{latency}^{max}, \tag{1}$$

where $T_{read}(k)$ denotes the time to read k disk blocks, T_{comm} is the time to deliver a block from storage node to network node, and T_{net} is the time to

[1] For the description of algorithms in this section, the terms server cluster and server are used interchangeably.

$node0$	$node1$	$node2$	$node3$
$A.0$	$A.1$	$A.2$	$A.3$
$A.4$	$A.5$	$A.6$	$A.7$
...	...	...	...
$B.3$	$B.0$	$B.1$	$B.2$
$B.7$	$B.4$	$B.5$	$B.6$
...	...	...	...
$C.2$	$C.3$	$C.0$	$C.1$
$C.6$	$C.7$	$C.4$	$C.5$
...	...	...	...
$D.1$	$D.2$	$D.3$	$D.0$
$D.5$	$D.6$	$D.7$	$D.4$
...	...	...	...

Fig. 3. An example of data distribution. *$B.3$ denotes the 3rd block of B stream. The start block of each stream is distributed across storage nodes for load balancing.*

deliver a block to clients.

By rewriting Eq. (1), the number of nodes in a server may be bounded to

$$n \leq (T_{latency}^{max} + T_{play} - T_{read}(1) - T_{comm} - T_{net})/T_{play}. \quad (2)$$

3.2 Disk scheduling

The performance of VOD servers is limited by their relatively low disk bandwidth. This section describes a scheduling technique that fully utilizes the disk bandwidth of storage nodes while satisfying the continuity requirement of video streams.

We first consider the process involved in serving a single client. A disk block must be retrieved for the client every T_{play} seconds. From the standpoint of a storage node, it must retrieve a disk block every $n \times T_{play}$ seconds, where n is the number of storage nodes. Thus, the retrieval of a disk block must be completed within $n \times T_{play}$.

We now consider s client requests, r_1, r_2, $\cdots$, r_s. In each storage node, s retrievals for $r_1, r_2, \cdots, r_s$ constitutes a *round*. That is, in a storage node, B_j^1,

round:	(1)	(2)	(3)	(4)	(5)	(6)	(7)	$\cdots$
r_1 :	B_1^1	B_2^1	B_3^1	B_4^1	B_5^1	B_6^1	B_7^1	$\cdots$
r_2 :	B_1^2		B_2^2		B_3^2		B_4^2	$\cdots$
r_3 :	B_1^3			B_2^3			B_3^3	$\cdots$
r_4 :	B_1^4	B_2^4		B_3^4	B_4^4		B_5^4	$\cdots$
time:	0	1	2	3	4	5	6	$\cdots$

Fig. 4. A scenario of the simple round scheduling in a storage node. r_i *denotes the* i^{th} *client request and* B_j^i *denotes* j^{th} *disk block of* r_i *in a storage node. The disk bandwidth is not fully utilized.*

$B_j^2, \cdots, B_j^s$ are retrieved in the jth round, where B_j^i denotes the jth disk block of r_i in the storage node. Each disk block, B_j^i, must be retrieved within its deadline, $n \times T_{play}^i$. The deadlines of B_j^i, $1 \leq i \leq s$, will be met if the period of a round is given by the shortest playback time of s disk blocks, $n \times T_{play}^{min}$, where $T_{play}^{min} = min_{1 \leq i \leq s}(T_{play}^i)$. In order to service s clients without violating the continuity of streams, the following admission control criteria must be satisfied.

$$T_{read}(s) \ \leq \ n \times T_{play}^{min} \tag{3}$$

When clients access heterogeneous streams, T_{play}^i's of client request r_i, $1 \leq i \leq s$, are different from each other with respect to the playback rate of video streams. If $T_{play}^1 = T_{play}^2 = \cdots = T_{play}^s = T_{play}$ then every $n \times T_{play}$ seconds, a block for each client request is retrieved and consumed; thus not accumulated. If $T_{play}^i > T_{play}^{min}$, however, data accumulation will occur for r_i. To avoid this untoward effect, we opt to schedule the blocks to be retrieved in each round. Such a procedure is called *round scheduling*. Fig. 4 shows a simple scenario where $n \times T_{play}^1 = 1$, $n \times T_{play}^2 = 2$, $n \times T_{play}^3 = 3$, $n \times T_{play}^4 = 1.5$.

As shown in this example, B_1^1, B_1^2, B_1^3, B_1^4 are retrieved in the first round; B_2^1 and B_2^4 in the second round. As only two blocks are retrieved in the second round, there exists disk idle time. Hence, the client requests that have failed the admission control can be serviced during the idle time. For instance, if we assume that four clients can be simultaneously serviced or that four blocks can be retrieved in a round, and if r_5 with $n \times T_{play}^5 = 1.5$ and r_6 with $n \times T_{play}^6 = 2$ arrive, then the admission control for these two requests will fail. Requests r_5 and r_6, however, can be serviced during the disk idle time. Fig. 5 shows a schedule for this scenario. Further analysis of these cases results in the following theorem. Let $Q = \{i | i = 1, 2, \cdots;$ index for client requests$\}$ and $k_i = T_{play}^{min} / T_{play}^i$.

round:	(1)	(2)	(3)	(4)	(5)	(6)	(7)	$\cdots$
r_1 :	B_1^1	B_2^1	B_3^1	B_4^1	B_5^1	B_6^1	B_7^1	$\cdots$
r_2 :	B_1^2		B_2^2		B_3^2		B_4^2	$\cdots$
r_3 :	B_1^3			B_2^3			B_3^3	$\cdots$
r_4 :	B_1^4	B_2^4		B_3^4	B_4^4		B_5^4	$\cdots$
r_5 :		B_1^5	B_2^5		B_3^5	B_4^5		$\cdots$
r_6 :		B_1^6		B_2^6		B_3^6		$\cdots$
time:	0	1	2	3	4	5	6	$\cdots$

Fig. 5. A scenario of the efficient round scheduling in a storage node. *Requests r_5 and r_6 can be serviced during the disk idle time of Fig. 4.*

Theorem 1 *It is possible to service a new request r_a even when the admission control fails, if the following relationships hold:*

$$\sum_{i \in Q} k_i \quad \leq \quad 1 \quad \textit{and} \quad a \in Q. \tag{4}$$

Proof. k_i for r_i is the number of blocks retrieved in a round and $k_i \leq 1$. If there exists Q such that $\sum_{i \in Q} k_i \leq 1$ then the blocks for $\{r_i | i \in Q\}$ can be retrieved. That is, for r_i, $i \in Q$, k_i/k_{min} blocks are retrieved once every $1/k_{min}$ rounds, where $k_{min} = \min_{i \in Q}(k_i)$. When the values of k_i/k_{min} is not a whole number, the values are toggled between $\lceil k_i/k_{min} \rceil$ and $\lfloor k_i/k_{min} \rfloor$, so that k_i/k_{min} blocks are retrieved once every $1/k_{min}$ rounds on the average. Thus r_a can be serviced. $\square$

Let us apply Theorem 1 to r_5 and r_6 of Fig. 5. Since $k_2 = 1/2$, $k_3 = 1/3$, $k_5 = 2/3$, $k_6 = 1/2$, there exist Q_1 and Q_2 for r_5 and r_6, respectively, such that $Q_1 = \{3, 5\}$, $Q_2 = \{2, 6\}$. Hence r_2 and r_6 are serviced every other round. A block for r_3 and two blocks for r_5 are retrieved in three rounds, and these three rounds are repeated.

On the other hand, $T_{read}(s)$ in Eq. (3) should be optimized by disk head scheduling[9]. CSCAN[8] optimizes the disk seek time within two consecutive rounds. For the purpose of deterministic service guarantees, $T_{read}(s)$ may be bounded to $T_{seek\_max} + s \cdot b/R$, where b is block size and R is data transfer rate. This analysis can be also applicable when storage nodes have multiple disks, or a disk array.

In consequence, Eq. (3) and Theorem 1 can be integrated as our final admission control criteria.

3.3 Communication scheduling

In Section 3.2, we described the guaranteed retrieval of disk blocks in storage nodes. These blocks must also be transmitted to network nodes in a deterministic fashion over an interconnection network. For further discussion we choose the Omega network[12] as a candidate network in the server. This multistage interconnection network has the property that each data block to be sent through the network involves a unique path between source and destination. Thus, for a given set of blocks it may not simultaneously transmit the messages because some of the blocks may conflict with one another. To reslove such conflicts we may need to partition a set S of conflicting data blocks into k subsets, $S_1, \cdots, S_k$, such that each subset is conflict-free[10].

First, we consider communication patterns in the server. Our VOD server distributes video streams across all storage nodes and proceeds in periodic rounds at storage and network nodes. During a round, each storage node must transmit to network nodes m data blocks prefetched in the previous round. While delivering video data to network nodes, n storage nodes generate $m \times n$ (s_i, d_j) pairs, where s_i and d_j denote a source and a destination, respectively. In our environment, source means storage node and destination means network node. On clients' arrival at network nodes, requests are evenly distributed and then sent to storage nodes for service; so the amount of data blocks that a network node receives in return from storage nodes is the same as that of other network nodes. Therefore, in $m \times n$ (s_i, d_j) pairs, each value of s_i and d_j occurs m times exactly whereas the value of s_i and d_j ranges from 0 to $n-1$.

We now describe our communication scheduling algorithm. A round is divided into kn slots, and a data block is transmitted within a slot. For source node s_i, m data blocks are scheduled as follows:

$$(s_i, d_i), \cdots, (s_i, d_{n-1}), (s_i, d_0), \cdots, (s_i, d_{i-1}), \cdots$$

It can be shown that all data blocks are transmitted during a round without conflict based on the following theorem.

Theorem 2 *For a set S of $n \times n$ (s_i, d_j), $0 \le i \le n-1$, $0 \le j \le n-1$, pairs of communication paths, S can be partitioned into the following S_k's which are conflict-free:*

$$S_k = \{(s_i, d_{i+k}) \mid 0 \le i \le n-1\},\ 0 \le k \le n-1$$

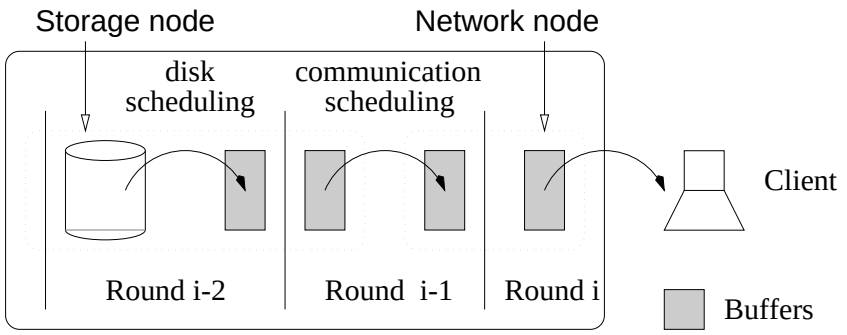


Fig. 6. Data flow in our VOD server. *Two buffers are required in both storage and network node.*

Proof. S can be partitioned into S_k's and it can be obtained from [12] that S_k, $0 \leq k \leq n-1$, is conflict-free. Therefore, the theorem holds. □

In general, considering the link bandwidth of interconnection networks is larger than that of disks, a storage node can transmit more than m blocks to network nodes, or $kn \geq m$. When $kn \geq m$, the admission control criteria we described in Section 3.2 can be applied. If $kn < m$, the network becomes a bottleneck in our VOD server, and a new client who demands that a storage node retrieve more than kn blocks should be rejected. When m equals kn, the utilization of all links in the network reaches 100%.

Since one of the major objectvies of designing VOD servers suggests that they service as many clients as possible, sufficiently large buffers have been assumed for our scheduling algorithms. We now observe the buffer requirement of our VOD server. A data flow in our VOD server is shown in Fig. 6. Since the schedules generated by disk scheduling and communication scheduling are different from each other, we employ the double buffering scheme at both storage node and network node. For this reason 4 buffers per client are required in our VOD server for deterministic service guarantees. The effective management of shared buffers, however, will decrease buffer requirements.

4 Performance Analysis

We now examine the configuration of a large-scale VOD server for given storage nodes. Our simulation model is based on the parameters listed in Table 1 that are considered suitable for the proposed large-scale server: For 640 storage nodes given, data striping across all the 640 storage nodes causes several problems, as described in Section 2. Especially, if we assume the block size to be 256 KB, start-up latency will be larger than 300 seconds from Eq. (1). Hence, we need to group storage nodes into server clusters. While Table 1

Table 1
Parameters used in our analysis

Number of storage nodes	640
Number of stored movies	2,000
Stream rate of a movie	0.5 MBps
Length of a movie	80 $\sim$ 120 min.
Block size	256 KB
Disk bandwidth of storage node	20 MB/s
Link bandwidth of network	20 MB/s

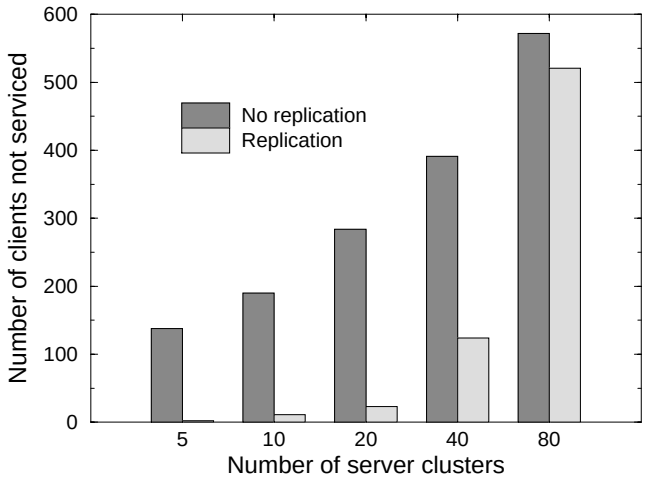
Table 2
The alternatives in the configuration

Number of server clusters	5	10	20	40	80
Number of storage nodes	128	64	32	16	8
Capacity of a server cluster	5,120	2,560	1,280	640	320
Start-up latency(sec)	64	32	16	8	4

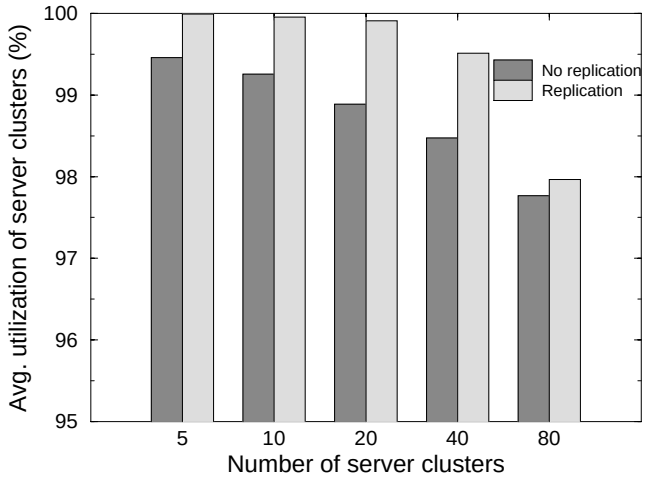
shows the parameters used in our analysis, performance analysis is based on a set of various alternatives for server configuration as listed in Table 2. By capacity of a server cluster in Table 2 we mean the number of clients that can be serviced simultaneously in a server cluster and by start-up lantency the value in the worst case. In normal cases, the values are far smaller. All the alternatives in Table 2 can service 25,600 concurrent clients when the loads are totally balanced across server clusters.

Based on video store rental patterns, it is known that accesses to movies in the server will be highly localized, with a small number of movies receiving most of the accesses[6]. According to Zipf's Law[6] the probability of choosing the nth most popular one from M movies is C/n, where $C = 1/(1 + 1/2 + 1/3 + \cdots + 1/M)$. Thus, replicating popular movies in server clusters can keep the load of server clusters balanced. In this experiment we allocate 1,000 movie titles to server clusters in the round-robin manner according to their ranking and replicate top ranking movies in all the server clusters. For example, when there are 10 server clusters, each server cluster has 100 unreplicated movies and the top 100 replicated movies.

First, we carry out the simulation under the worst case assumption that 25,600 clients request concurrently. Movie requests are localized according to the Zipf's distribution. Simulation results are shown in Fig. 7. The analysis of this graph results in the following assertions: (1) Replicating popular movies shows better performance. It is possible to service 50 to 250 more clients. (2) Until



(a) Number of clients not serviced



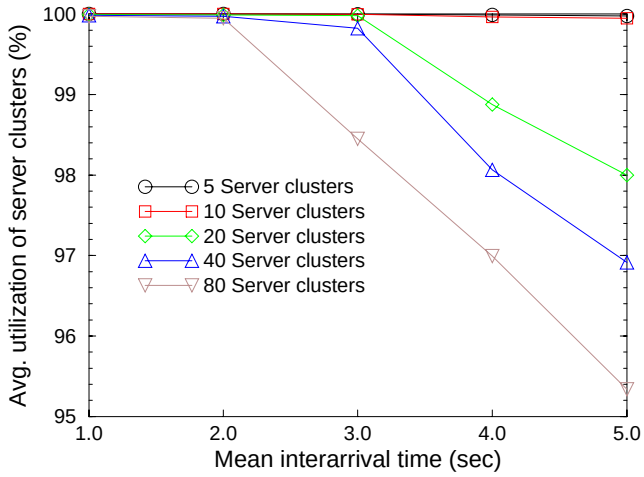
(b) Average utilization of server clusters

Fig. 7. Performance under the worst case assumption

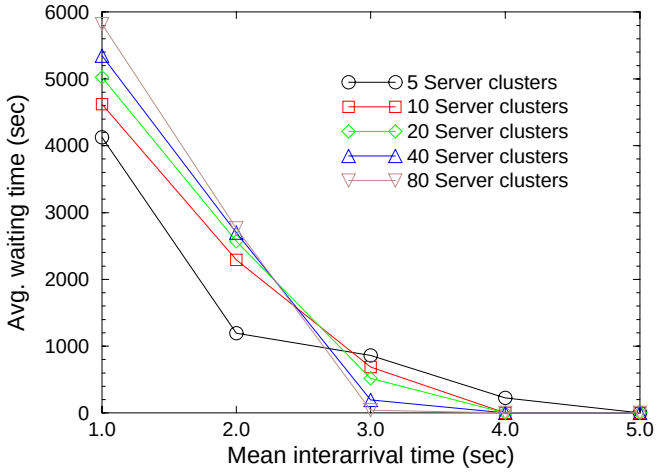
the number of server clusters becomes 20, the average utilization[2] of server clusters is close to 100%. (3) When there are large numbers of server clusters, there exist hot spots among server clusters, those places where there are likely to be far more client requests.

Second, we simulate the real VOD service with replication. It is assumed that clients arrive at the server according to the Poisson distribution with mean interarrival time, $1/\lambda$, and that the running time of each movie is uniformly distributed between 80 and 120 minutes. Fig. 8(a) shows similar results to Fig. 7. When the load becomes smaller in Fig. 8(a), the average utilization of server clusters decreases. This is because there exist hot spots among the server clusters whereas the other server clusters are underutilized. In Fig.

[2] # of clients being serviced/ # of clients can be serviced



(a) Average utilization of server clusters



(b) Average waiting time

Fig. 8. Performance under the average case

8(b), the larger the number of server clusters is, the longer the average waiting time[3] becomes. It also results from hot spots where the waiting time is longer. In addition, since the running time of movies is relatively long, the average waiting time under heavy load $(1/\lambda = 1, 2)$ is long as shown in Fig. 8(b).

In summary, given a large number of nodes, clients experience relatively large start-up time when the number of server clusters is small, or the size of a server cluster is large. On the other hand, when the number of server clusters is large, client requests are not balanced among server clusters, that is, there exist hot spots, even though popular movies are replicated. Consequently, the tradeoff of large versus small clusters provides a basis for the design of most effective server configuration. As to the size of a server cluster, it can be determined

[3] The waiting time means only queueing delay here.

based on the analysis of tradeoffs between the utilization and start-up latency. In the example given above, our simulation reveals that the most appropriate size of a server cluster is 32 storage nodes, since the average utilization of server clusters is close to 100% while the start-up latency is relatively small. This value is feasible by current technologies.

5 Conclusions

In this paper, we have presented the solutions for designing a large-scale VOD server which consists of a large number of nodes connected by a high performance interconnection network. Given hundreds of storage nodes, we divide them into server clusters, which service video streams independently. In a server cluster, two scheduling algorithms are proposed in order to fully utilize the resources: disks and networks. The round scheduling technique retrieves data blocks from disks effectively minimizing the disk idle time, and the communication scheduling guarantees conflict-free communication over the multistage interconnection networks that are topologically equivalent to Omega network.

We also show some analytical results on the configuration of a server for given nodes. As the number of storage nodes in a server cluster becomes smaller, start-up latency decreases, but hot spots develop among the server clusters. We have shown, however, that the replication of movies keeps the utilization of server clusters high. Analysis of tradeoffs between the utilization and the start-up latency help to decide a design strategy for selecting the most appropriate number of server clusters and the size of server clusters for given nodes.

The following problems remain to be addressed in the future: (1) effective buffer management in storage and network nodes and (2) supporting interactive operations (e.g., fast forward, rewind, pause, and resume).

References

[1] A. L. N. Reddy, Scheduling and data distribution in a multiprocessor video server, in: *Proc. International Conference on Multimedia Computing and Systems* (Washington, DC, 1995) 256–263.

[2] R. Tewari, R. Mukherjee, D. M. Dias, and H. M. Vin, Design and performance tradeoffs in clustered video servers, in: *Proc. International Conference on Multimedia Computing and Systems* (Hiroshima, Japan, 1996) 144–150.

[3] S. Ghanderharizadeh and L. Ramos, Continuous retrieval of multimedia data using parallelism, *IEEE Transactions on Knowledge and Data Engineering* **5** (1993) 658–669.

[4] C. S. Freedman and D. J. DeWitt, The SPIFFI scalable video-on-demand system, in: *Proc. 1995 ACM SIGMOD* (San Jose, CA, 1995) 352–363.

[5] A. Heybey, M. Sullivan, and P. England, Calliope: A distributed, scalable multimedia server, in: *Proc. USENIX 1996 Annual Technical Conference* (San Diego, CA, 1996).

[6] A. L. Chervenak, D. A. Patterson, and R. H. Katz, Choosing the best storage system for video service, in: *Proc. ACM Multimedia '95* (San Francisco, CA, 1995) 109–119.

[7] T. D. C. Little and D. Venkatesh, Probabilistic assignment of movies to storage devices in a video-on-demand system, in: *Proc. International Workshop on Network and OS Support for Digital Audio and Video* (Lancaster, U.K., 1993) 213–224.

[8] J. L. Peterson and A. Silberschatz, *Operating System Concepts* (2nd Ed., Addison-Wesley, 1985).

[9] M.-S. Chen, D. D. Kandlur, and P. S. Yu, Optimization of the grouped sweeping scheduling (GSS) with heterogeneous multimedia streams, in: *Proc. ACM Multimedia '93* (Anaheim, CA, 1993) 235–242.

[10] P. J. Bernhard, Bounds on the performance of message routing heuristics, *IEEE Transactions on Computers* **42** (1993) 1253–1256.

[11] C. L. Wu and T. Y. Feng, On a class of multistage interconnection networks, *IEEE Transactions on Computers* **C-29** (1980) 694–702.

[12] D. H. Lawrie, Access and alignment of data in an array processor, *IEEE Transactions on Computers* **C-24** (1975) 1145–1155.