# An Effective Scheduling Policy in Large-Scale Video Servers

Yu-Zhan Hou Yu-Jiin Wang Cheng Chen
Department of Computer Science
and Information Engineering,
National Chiao Tung University,
Hsinchu, Taiwan, R.O.C.
TEL: (03) 5712121 ext 31834
Email:cchen@eicpcas.csie.nctu.edu.t

Abstract

*Due to the advances in computer and multimedia techniques, the applications of video-on-demand are visibl and influence people's life worldwide. However, with th growth of service demand, the single-server architecture is limited by its scalability and server-level fault-tolerance. Recently, several multi-server architectures are proposed to alleviate such problems.*

*In this paper, we will propose a scalable architecture for video server management system. Our architecture employs not only distributed storage servers but also distributed subscriber management. Wide striping technique is a cost-effective data placement in multi-server architecture. However, the unpredictable access skew fro users results in performance degradation. Therefore we design a new scheduling policy to fully utilize the potential bandwidth of wide striping by controlling the startup time of each request. With different modes of our policy, we can benefit from minimum delay or load balance. According to our simulation results, our policy efficiently alleviates load imbalance and thus improves system performance. The detailed information about design principles and performance evaluations will be described in the literatu re.*

*Keyword: large-scale video server, job scheduling, request delaying, load balancing, wide striping*

## 1. Introduction

Recently, advances in computing and communication technologies make it possible to provide a wide range of interactive multimedia services in a variety of commercial and entertainment domain [1, 12-13]. One of the most visible applications of multimedia systems is on-demand playback of video in a distributed environment. In the distributed system, video server plays an important role that provides storage for video content and interactive service for users. Thus, designing a hig performance video server for VOD or Near-VOD applications becomes an interest in recently years [1-7]. Basically, a video server contains two main parts. One is so ca lled SMU (Subscriber Management Unit) and the other is VSE (Video Server Engine). SMU is responsible for admission control, resource management, and job scheduling. On the other hand, VSE handles storage management, buffer management, and data retrieval and delivery [12-14]. However, single-server architecture has limitations on scalability and server fault-tolerance [1, 10-13]. For large-scale service demand, these limitations cause performance degradation and serious impact on service quality. In recentl years, several scalable architectures are proposed to solve the problem [1-8]. Multi-server architecture is common in these approaches. Clustered vidco server in [2] offers a generalized model of multi-server architecture. Such clustered architecture consists of multiple nodes interconnected by a switch. The nodes of a cluster can be categorized into delivery nodes and storage nodes. Other similar multi-server architectures such as MARS [3], Tiger Fileserver [4], Tiger Shark [5], Server Array [6-7] have be en proposed to enhance the scalability of video server system. We will adopt a multi-server architecture, which consists of multiple SMU and VSE nodes, to develop a scalable video server system. Our architecture is similar to original clustered server [2 except video data will be directly routed to client from VS nodes. A specific component named Master Server controls all components in the system and admits incoming requests.

In our system, we also design a new scheduling policy, named *delayed-startup policy*, to utilize the full capabilit of wide striping [2, 8-10]. This policy explores the potential capability by controlling the startup time of each request. With different delaying mechanisms, we can obtain different performance gains. One of them yield s minimum delay time and seems preferable for real-time applications. However, this way still results in lower throughput for load imbalance. Therefore, we may employ another delaying mechanism that always postpones the request to the node with the lightest load. This load balancing technique enhances the throughput but causes significant delay. We also analyze the trade-off between system throughput and delay time, and then give a generalized mode benefits from the two factors. We can show that our policy indeed improves the system throughput, leads to more load balance across multiple servers, enhances node utilization, and reduce the cost compared with previous work [8-9].

The remainder of the paper is organized as follows. In Section 2, the design issues of our distributed video server system will be described. I Section 3, we will present the concept and principle of proposed scheduling policy. Related performance gains will be evaluated and analyzed in detail in Sectio 4. Finall , Concluding remarks will be given in Section 5.

## 2. Design of Our Distributed Video Server System

### 2.1. Overview

We have designed and implemented an effective single-server management system for VOD applications [13]. As shown in Fig.1, SMU contains five modules: **Admission Control, Job Schedule, Disk Schedule, Data Placement,** and **CD/HD Swapping**. Admission control module guarantees that the service quality will not be degraded or be contravened by newly incoming request. I order to keep accepted streams for continuous playback, job schedule module is used to control VSE to retrieve video data from disks and then buffer them in a cycle fashion. Meanwhile, disk schedule module is used t optimize the performance of data reading from disks by rearranging the order of retrieval commands from job schedule module. In VSE, there may be multiple disks

equipped as storage device to store hundreds of contents. Data placement. module will control how to place the contents into storage space. And the CD/HD swapping module response to update the contents from CD-ROM into disks.

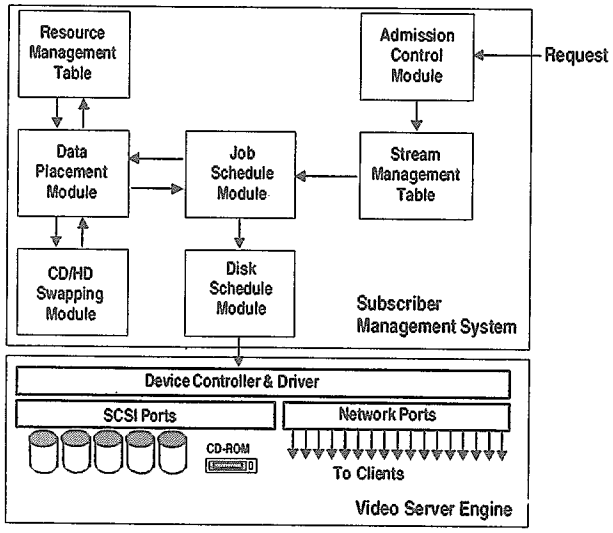



Figure 1. Basic Configuration of Our Single Video

The original design of VSE aims at small -scale video-on-demand service. However, as the requirement of service grows up, the performance of single video server is limited. So we re -design the original video server configuration, and a distributed architecture of video server cluster is proposed to meet the requirement of scalability.

The new architecture of our distributed system is shown in Fig.2. Each SMU communicates with other SM and VSE through Ethernet. All VSE nodes are interconnected with a high-speed network for exchangin video data, and the high-speed network is in turn connected to outward service network. There are three primary entities in our design. The **Master Server** is used for admittin and dispatching user requests to slave SM nodes. Slave **SMU** handles the service requests of admitted users, and **VSE** performs storage management, buffer management, and data retrieval.

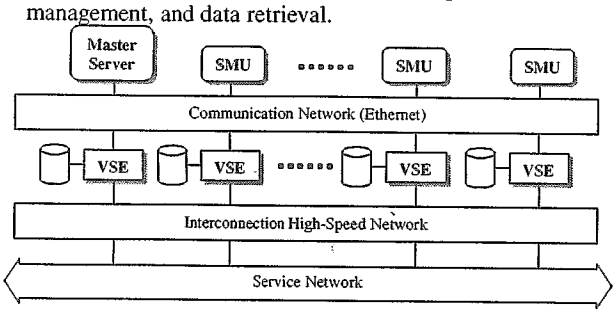



Figure 2. Architecture of Our Distributed Video Server System

Master Server's primary task is to admit user request. And then dispatches them to the slave SMU according t some scheduling rule with load balancing principle. Master Server also plays a global resource manager that keeps status of all nodes and system timer. The global resource table contains node information, data allocation, etc. If some node crashes, then Master Server must handle the recovery of service by fault tolerance mechanism. The slave SMU nodes accept the user requests from Master Server and then schedule the requests immediately. Because, SMU node performs the data retrieval schedulin of admitted streams, it will have a copy of data allocation and schedule information in its local resource table. The retrieval commands are sent to interconnected VSE nodes, and then the ready video data are transmitted to client end through service network. VSE can be viewed as a storage node which controls the data storage and retrieval. It may equip multiple storage devices and a large volume of buffer for storing video data. Because the performance of VSE node is determined by the I/O bandwidth of storage device, a good disk scheduling algorithm is necessary to optimize reading data from disks. In our distributed model, the access requestes may come from multiple SMU nodes, so it is better to put disk scheduling module into VSE node for effective management. Before performing the data access, buffer management should allocate sufficient buffer and arrange the allocation efficiently.

## 2.2. Design Issues

The bottlenecks of server scalability in our distribu ted system may come from Master Server, interconnection networks, and load imbalance caused by access skew of requests. If the request queue of Master Server is not controlled well, the performance will drop down proportional to the system load. To overcome such problems, we apply different priorities for request processing. The admitted and scheduled streams, which are sensitive to delay, have higher priority to process. And the new and non-scheduled streams are with lower priority. If the queue is too long to service in time, request delaying is necessary to guarantee quality of service. The clustered architecture is inherently constrained by interconnected network and ATM switch is the common solution in lots of related work [2-4]. The performance of distributed vide server will be limited due to unbalanced load distribution or unbalanced inherent capability of nodes. Against the problem of load imbalance, we have to design effective data-placement methods and job-scheduling techniques to help load balancing at static and dynamic time.

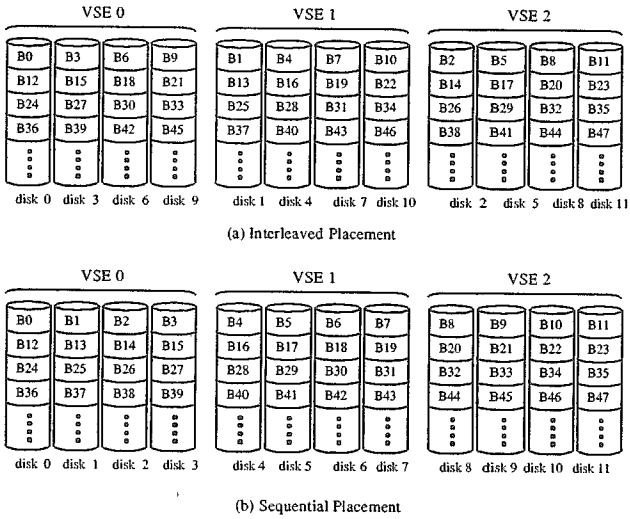



Figure 3. Different Data Placement Schemes of Wide Striping

We take advantage of data striping technique to allocate video clips on disks. If we adopt wide striping rule [2, 8-10] which distributes striped video blocks over all storage nodes, then higher bandwidth and load balancin will be achieved. In our storage subsystem, there may be multiple storage nodes (VSE). Each VSE may possess

multiple disks. We briefly introduce two placement schemes now. The first is *interleaved placement* and the other is *sequential placement*, which are shown in Fig.3 (a) and (b) respectively [10, 15]. Interleaved placement allocates data blocks across VSE nodes in interleaved fashion while sequential placement puts the next block on the contiguous logical disk. While the numbe r of disks is large, similar effect of access skew in large stripe factor arises among VSE nodes. Interleaved placement ma reduce this skew. However, communicative overhead ma be reduced with sequential placement.

Based on our data allocation model, the data retrieval process works periodically to keep up with playback of servicing streams. In our design, there exists a global schedule in Master Server. We can observe that the load pattern is not uniformly distributed among disks and the overloaded disks may cause QOS decay. If request is tolerant for some waiting time, we can avoid the overloaded condition by delaying request [9, 15]. However, previous work only notices to avoid overloaded condition but ignores the effect of load balance. In fact, load balance in such case also makes sense in terms of syste performance or fault-tolerance.

## 3. Delayed-Startup Scheduling Policy

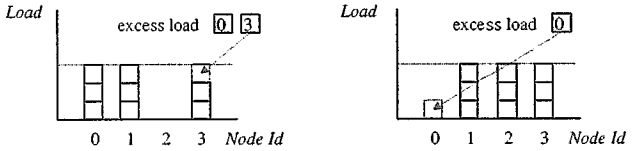
With a good scheduling policy, we can balance the load of data retrieval at run -time, therefore to guarantee high performance, enhance quality of service, reduce buffer size, and strengthen fault-tolerance of system. Now we start to introduce the basic concept of a new scheduling policy in our system environment, and then present the principle of the proposed policy in some detail.

### 3.1. Job Scheduling Model

Our job scheduling adopts round scheduling, where time is divided as equivalent service rounds ( $\tau$) and each storage node can service many requests during a service round. Therefore, each service round can be further divided into several equivalent *time slots*, and each time slot is allocated to only one stream for avoiding resource contention. If we have $N$ disks in our system and each one equips the service capacity of $M$ requests in a service round, then a global schedule table with two-dimension array ($N \times M$) is needed to record the allocation of time slots. The maximum capacity of system is therefore $NM$ streams. For the sake of random access of request, the load distribution is not uniform. Some nodes may carry load more than $M$, and some nodes are underflow (i.e. load < $M$). The schedule model must satisfy two characteristics of general scheduling problem [3-4, 8-9]: (1) *real-time requirement*: every request should be scheduled before deadline so as to satisfy real-time data retrieval; (2) *conflict-free schedule*: in each time slot, no two or more streams request the same storage node. If each node is protected from overloading, then all the scheduled requests will be serviced in time. Meanwhile, the schedule is conflict-free since each row of schedule table belongs to unique storage node. Therefore, our schedule model meets the two requirements mentioned above.

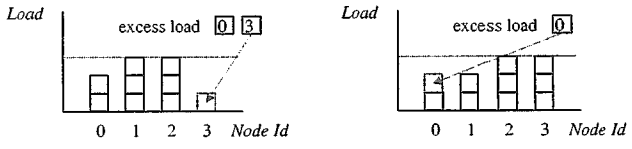
### 3.2. Principle of Delayed-Startup Scheduling

Now, we want to take advantage of the deterministic feature of data striping to exploit the potentia l capability. The migration of load distribution is periodical during data retrieval process. If we allow requests to delay several rounds, the unused time slots would be allocated to excess requests. Without losing generality, we identif $r_i$ as the request which attempts to access node $i$, and $s_i$ represents one available slot of node $i$ similarly. Whenever, there exists a request without correspondin $s_i$, we name it as *excess request*. For example, assumed $N = 4$, $M = 3$, and the incoming 10 requests attempting to access storage nodes is {0, 0, 3, 2, 3, 2, 3, 0, 3, 0}, we get two excess requests, $r_0$ and $r_3$. During the load migration, the two excess requests can be held and allocated at round ( $i+1$) and round ($i+3$) respectively as shown in Fig.4. By this way, no r equest needs rejection until all available slots are allocated. Hence, the full capability of storage subsystem can be utilized.




(a) Load distribution at round ($i+1$) (b) Load distribution at round ($i+3$)

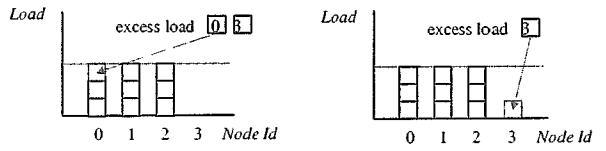
Figure 4. Excess Load Allocation




(a) Load distribution at round ($i+2$) (b) Load distribution at round ($i+3$)

Figure 5. Alternative Excess Load Allocation

In general, there will be many candidates of available slots for choice, so the scheduling problem can be considered as a problem of how to allocate available slots efficiently. Different time slot allocation yields different delay time. Fig.5 shows another allocation different from Fig.4. The excess requests, $r_3$ and $r_0$, are delayed to round ($i+2$) and round ($i+3$) respectively, and the total delay time is 5 rounds. The first allocation has smaller delay time while the second one reflects a balanced load distribution. To guarantee the service quality, request delaying is onl applied to startup time of streams. We name such policy as *delayed-startup* scheduling, and its main principle is to fully utilize the potential capability of data striping by delaying requests. For exploring the delayed -startup scheduling policy, there are some assumptions and notations have to make clear first. We assume the video data is encoded by unique bit rate ($r_{play}$) and the striping block size ($B$) is also unique. These two values determine the length of service round ($\tau = B/r_{play}$). And the length of each video is long enough to cover all storage nodes. Based on this concept and principle, we can develop further three kinds of scheduling modes in the following sections.




(a) Schedule request 0 at round ($i+2$) (b) Schedule request 3 at round ($i+2$)

Figure 6. Allocation in Minimum Delay Mode

### 3.3. Minimum Delay Mode

The main goal of this mode is trying to minimize the total delay of all excess requests. Here, we propose our algorithm, which finds one allocation pattern with minimum delay time. The formal algorithm is presented in Algorithm 1. The worst case of finding available node consumes O($N$) and searching for available slot requires O($M$). Therefore, the complexity of this algorithm is O($|R|(N+M)$), where $|R|$ means the number of excess requests. For the same example in Fig.4, we can directl apply the algorithm to schedule all the excess requests at round ($i$), and we obtain the result that $r_0$ is delayed by rounds and $r_3$ is delayed by 2 rounds as shown in Fig.6. The total delay time is 4 rounds. Theorem 1 will address the correctness of this algorithm.

**Algorithm 1:** Minimum Delay Time Allocation
**Input:** schedule table: $T_{N\times M}$
load table: $L_N$
excess request set: R
number of nodes: N
capability of node: M
**Output:** schedule table $T'_{N\times M}$ with minimum delay time
**Program:**
**for** ∀ r ∈ R **do**
/* find nearest available slot in $T_{N\times M}$ */
**for** i = 0 to N-1 **do**
/* find underflow node */
**if** (L[(r-i) mod N] < M) **then break**;
**end for**
**for** j = 0 to M-1 **do**
**if** (T[(r-i) mod N][j] = ∅) **then**
/* schedule the excess request in chosen slot */
T[(r-i) mod N][j] = r;
/* update load table */
L[(r-i) mod N]++; R = R – {r};
**break**;
**end for**
**end for**
❑

**Lemma 1:** Given a set of requests, $R$, and a set of available slots, $S$. For any pair of ($r_i$, $s_j$), where $r_i \in R$, $s_j \in S$, the delay time, $d(r_i, s_j) = (r_i - s_j) \bmod N$.
**Proof:** It is trivial. ❑
**Theorem 1:** Given a set of requests, $R$, and a set of available slots, $S$, and $|R| \le |S|$. The total delay time, $T = \sum_{i=0}^{|R|-1} \min\{d(r_i, s_j) \mid s_j \in S\}$, is minimum. In other words, if each request selects its nearest available slot we will get the minimum total delay.
**Proof:** The set of requests, $R$, and the set of available slots, $S$, can construct a complete bipartite graph, whose edge are given with delay time by Lemma 1. We set an extra vertex as the root and then links to each $r \in R$ with weight 0. Assumed that $|R| = m$, then apply the Prim's algorithm [16] to find the minimum-spanning-tree in the graph, then we obtain the $m$ edges with the minimum total delay. This completes the proof. ❑

The problem of minimum delay mode is that access skew still exists among storage nodes. This may cause performance degradation whether in normal or failure cases Load balance mode will solve this problem in the following.

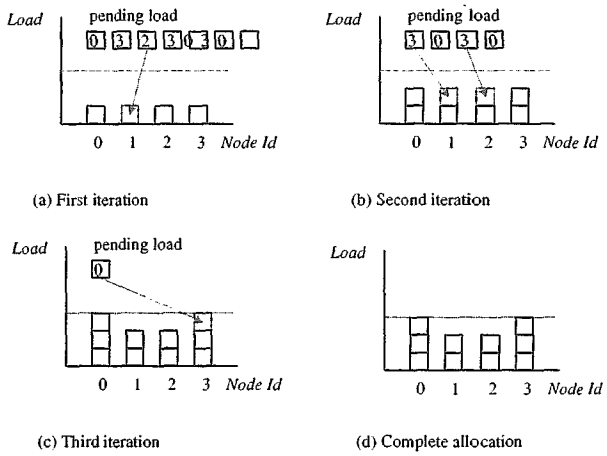



Figure 7. Allocation in Load Balance Mode

### 3.4. Load Balance Mode

With load balance mode, the incoming request is always allocated to the available slot of storage node with the lightest load. We define one iteration is to allocate ever $N$ available slots. With the same example in Fig.4, Fig.7 shows the three iterations of allocation. In the first iteration, the first three requests without delay $r_0$, $r_2$ and $r_3$, are scheduled, and the remaining available slot, $s_1$, is allocated to the $r_2$ for minimum delay. In the second iteration, the $r_0$ and $r_3$ are allocated without delay and the $s_1$ and $s_2$, are assigned to the two $r_3$'s respectively. The third iteration allocates the $r_0$ and delays the final $r_0$ to $s_3$. The total delay time is 5(=1+1+2+1) rounds.

The formal description of this method is presented in Algorithm 2 and the complexity of this algorithm is O($|R|^2$). The proof of Algorithm 2 is trivial. Since load is balanced during the process of assigning slots, the outcome is a balanced condition.

**Algorithm 2:** Load Balance Allocation
**Input:** schedule table: $T_{N\times M}$
pending request set: R
number of nodes: N
capability of node: M
**Output:** schedule table $T'_{N\times M}$ with load balance
**Program:**

```
iteration = |R|/N;
for k = 0 to iteration-1 do
  /* allocate N available slots */
  for i = 0 to N-1 do
    if (T[i][k] = ∅) then
      /* find minimum-delay request */
      r_min = min { (r-i) mod N | ∀ r ∈ R};
      /* schedule selected request */
      T[i][k] = r_min; R = R − {r_min};
  end for
end for
if (R≠∅) then /* handle final iteration */
  for ∀ r ∈ R do
    for i = 0 to N-1 do
      /* find nearest available slot */
      if (T[(r-i) mod N][iteration] = ∅) then
        /* schedule selected request */
        T[r-i][iteration] = r; R = R − {r};
      break;
    end for
  end for
```

❑

It is intuitive that load balance mode suffers longer delay time by forcing requests to delay for load balance consideration. In later section, we will evaluate the dela time of this mode, and the result is so significant that we can not ignore its effect.

### 3.5. Generalized Mode

Here, we give a generalized mode to benefit from these two modes. The idea is inherited from load balance mode except limiting the distance for requests to look for available slots. We first introduce the concept of *round-constrain* (*RC*).

While we perform delayed-startup policy with load balance mode, we may find the available slot with the lightest load. Therefore, the distance for each request t explore the available slot ranges from 0 to ( *SF*-1), where *SF* means striping factor. However, when *SF* is large, the delay time may be significant. So we define round-constrain (*RC*) to limit the distance for Algorithm 2 to explore the proper slot. For each incoming request, we always find the available slot for load balance within *RC* rounds. If all the available slots within *RC* run out, the distance of exploring available slot will be extended t $2\times RC$ and so on. When the distance is extended to *N*, it will degenerate to load balance mode. Now, we give an example to explain the procedure concretely.

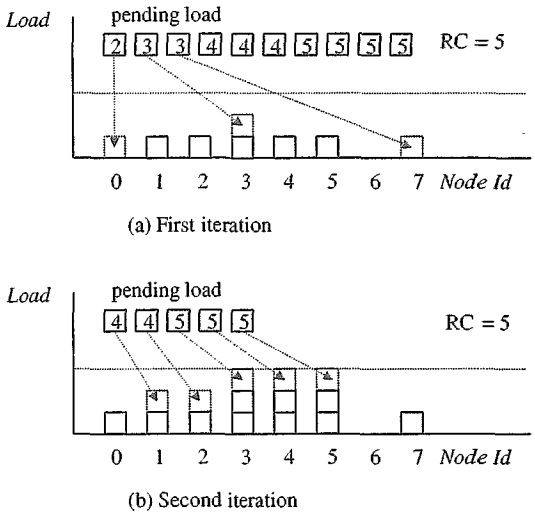



Figure 8. Example of Allocation in Generalized Mode

Assumed that $N = 8$, $M = 3$, $RC = 5$ and pending request set is {1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5}. Fig.8 shows the iterations of allocation. In the first iteration, one of $r_3$ should be allocated to $s_6$ if load balance mode is applied. However, the delay time will be 5 rounds which exceeds the maximum delay constrain of 4 rounds (=*RC*-1). Thus we find the lightest loaded node within the round-constrain. The chosen one is $s_3$, so no delay is required for this pending request. Similarly, in the second iteration, one of $r_5$ should be allocated to $s_0$ with load balance consideration. But the delay time also exceeds (*RC*-1). We allocate the request to $s_5$ that yields no delay similarly. The total delay time in the example is 14 rounds, which is half of the one caused by load balance mode (=29 rounds) and close to 10 rounds of minimum delay. The whole algorithm is presented formally in Algorithm 3 and the complexity of worst case in the algorithm is $O(|R|^2)$.

**Algorithm 3:** Generalized Allocation
**Input:** schedule table: $T_{N\times M}$
load table: $L_N$
pending request set: R
round-constrain: RC
number of nodes: N
capability of node: M
**Output:** schedule table $T'_{N\times M}$ with load balance
**Program:**

```
iteration = |R|/N;
group = (N-1)/RC + 1;
for k = 0 to iteration do
  /* allocate non-delayed request */
  for i = 0 to N-1 do /* for each available slot */
    for ∀ r ∈ R do
      /* find request without delay */
      if ( r = i and T[i][k] = ∅) then
        /* schedule selected request */
        T[i][k] = r; R = R – {r};
        break;
    end for
  end for
  /* allocate delayed request */
  for ∀ r ∈ R do
    for j = 0 to group-1 do
      /* find lightest node within round-constrain */
      s_min = min {(L[i+j*RC] | 0 ≤ i < RC and
                L[i+j*RC] ≤ M)};
      if ( s_min ≠ ∅ ) then
        for i = 0 to M-1 do
          if (T[s_min][i] = ∅) then
            /* schedule selected request */
            T[s_min][i] = r;
            /* update load table */
            L[s_min] ++; R = R – {r};
        end for
    end for
  end for
end for
```

❑

## 4. Simulation and Performance Evaluation

### 4.1. Overview of Our Simulation Environment

We have designed and implemented a discrete -event simulation package to evaluate the performance of the architecture used in our system as well as the proposed scheduling policy. Our simulation parameters are summarized in Table 1. The video data is encoded by MPEG-2 with bit rate of 3Mbps and block size is default t 94KB, thus the service round is about 250ms [14]. The number of video is default to 100. Average content length is 8192 blocks, which implies the number of maximal concurrent accesses is 8192. Request generation is modeled as Poisson arrival process with Zipf-like distribution of video selection [12-13]. The process of simulation will be executed for at least $V_{size}$ rounds to achieve a stead condition. We adjust the request arrival rate to model different system load, and collect related statistics in the steady condition.

| Simulation Parameter | Symbol | Default |
|---|---|---|
| Stream Number | $n_{stream}$ | 1000 |
| Load Factor | $\alpha$ | 50% ~ 100% |
| Disk Number | N | 8 ~ 128 |
| Disk Capability | M | 10 |
| Average Content Length | $V_{size}$ | 8192 |
| Parameter of Zipf Distributed | $\theta$ | 0.271 |
| Video Number | $V_{num}$ | 100 |
| SMU Capability | $C_{smu}$ | 100 (req/round) |
| VSE Capability | $C_{vse}$ | 60 (req/round) |
| High-Speed Network Capability | $C_{network}$ | 500 (req/round) |

Table 1. Simulation Parameters

To model different scales of our system, the disk number varies from 8 to 128. Each node should have an upper bound on computability. We assume VSE node is able to handle 60 requests per round, and each SMU works well within the moderate request arrival rate of 100 requests per round. Since the load of Mas ter Server will be dispatched to slave SMU nodes evenly, Master Server will not be a bottleneck unless a burst of incoming requests. However, we still constrain the upper bound of capability in Master Server. Another capability limitation comes from interconnection networks. During the process of simulation, the traffic of data that flows through the network should not exceed the bound. If it indeed happens, then we choose to delay the excess requests or reject them. Similar criterion is applied to previous capability limitations.

### 4.2. Evaluations of Proposed Architecture

First, we show the necessity of partitioning a video stream into sub clips and then distributing them across server nodes in the following example. Given 16 VS nodes where each one equips 8 disks, and we want to store 100 videos on these storage nodes. Assumed $P(V_0) > P(V_1) >...> P(V_{99})$ without losing generality, where $P(V_i)$ stands for popularity of video, $V_i$. Applying Round & Round placement in [13], $V_i$ is placed on node ($i$ mod 4). Due to popularity skew, hotter videos are constrained by the capability of single node, so we suggest that partitionin and distributing video clips across all server nodes, i.e. wide striping. We may have two ways to implement wide striping scheme. Fig.9 shows the corresponding reject probability of interleaved and sequential placements respectively with FCFS scheduling policy. Compared t Round & Round placement, these two wide stripin schemes indeed gain higher performance. From the results of our evaluation for wide striping, sequential placement performs better than interleaved placement. In the following discussion, we employ sequential placement as our default data placement scheme.

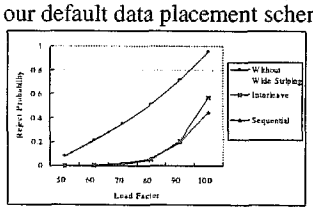



Figure 9. Reject Probability of Various Data Placement Schemes

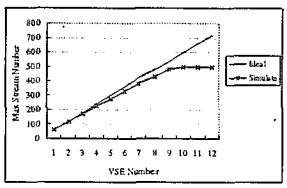



Figure 10. Scalability with Different Number of VSE

Given different request arrival rates, we can evaluat e the node utilization of VSE and deduce the maximu throughput. Here, we assume the total capability of SMU nodes is always sufficient to handle the incoming requests. Fig.10 shows that the growth of system throughput is almost linear until it saturates for the bottleneck of network

bandwidth.

According to these aforementioned evaluations, we can observe that even various data placement schemes are applied, these static load balance techniques will not explore the full capability of resource yet and the performance degrades as system scales up. Some dynamic load balancing strategy must be considered to guarantee the performance.

### 4.3. Evaluations of Delayed-Startup Policy

Now, we apply our delayed -startup policy in our system and evaluate its performance gains. Fig.11 shows the throughput of minimum delay mode (MD) and load balance mode (LB). Compared with FCFS policy, The throughput is enhanced with delayed-startup policy, and the LB mode performs very close to the ideal case.

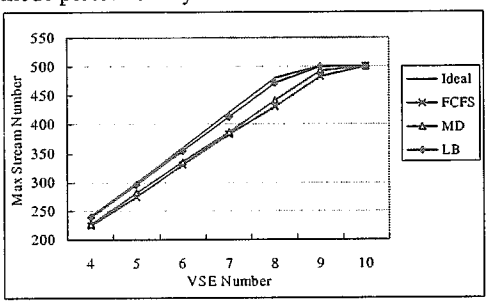



Figure 11. Throughput of Different Scheduling Policies

Fig.12 shows the reject probability with various load factors, where VSE number is 4 and each one equips 8 disks. Delayed-startup policy indeed reduces the reject probability especially in the condition of heavy load. With load balance mode, we obtain the lowest reject probability. However, the cost of higher throughput is significant dela time as shown in Fig.13. To reduce the total delay time, we may apply the generalized mode with proper round-constrain (*RC*). For lower average delay time, w may choose smaller *RC*. However, if lower reject probability is on demand, larger *RC* is preferred. While system load is heavy, the reject probability goes high. This implies that we can adjust RC dynamically according to the system load and thus reduce the request delay time. For a large striping factor, the variance of request delay is large as well. We may employ extra buffer to pre-fetch some data blocks for each stream and amortize the total delay time. Such buffering technique also reduces request response latency and enhances the QOS. Fig. 14 shows that extra buffer helps reducing the request delay in delayed -startup policy. However, buffer size could not be too large, or the initial time of pre-fetching blocks will cause longer request delay.

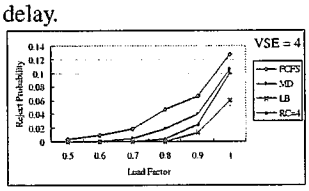



Figure 12. Reject Probability with Various System Load Factors

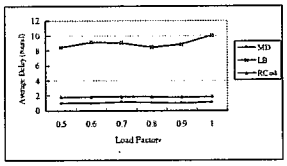



Figure 13. Average Delay of Different Delayed-Startup Modes

Compared with other scheduling methods, our proposed policy offers higher performance if sufficient delay time is allowed. Fig.15 shows the reject probabilit of various scheduling policies. Though the method in [9] solves link conflict problem, but its request-delaying algorithm (Peg-and-Hole) only avoids overloaded condition and ignores load balance. And in [8], the scheduling scheme (Greedy) employs FCFS policy and request delaying is not allowed. Both methods suffer higher reject probability due to load imbalance.

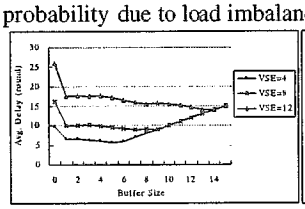



Figure 14. Average Delay with Pre-fetch Buffering

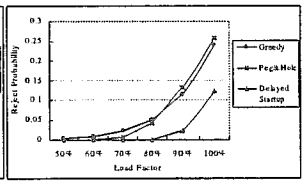



Figure 15. Reject Probability of Different Scheduling Policies

We conclude that load imbalance makes a critical factor of performance in a large -scale video server environment. Wide striping is an effective static load balancing technique, but still suffers load imbalance. So we proposed a dynamic load balancing technique, delayed-startup scheduling, to explore the full capability of wide striping.

## 5. Concluding Remarks

In this paper, we have designed a scalable video server system with multi -server architecture. The critical iss ues of data allocation and retrieval are also addressed in the contents. In aspect of data allocation, we suggest *sequential placement* of wide striping for higher performance. And wide striping is cost -effective since the concurrent accesses can be scaled up without replication. On the other hand, the proposed scheduling policy, named *delayed-startup policy*, optimizes the node utilization or the delay time with two different modes. We also provide a generalized form that compromises various preferences for performance and average the system throughput, leads to more load balance across multiple servers, enhances node utilization, and reduce the cost of server system by simulation evaluations.


## Acknowledgement

The paper was supported by NSC 87-2622-E009-008


## References


[1] Y.B. Lee, "Parallel Video Servers:A Tutorial," *IEE Multimedia*, No. 5, Issue 2, pp.20-28, April-June 1998.

[2] R. Tewari, R. Mukherjee, D. Dias and . Vin, " Design and Performance Tradeoffs in Clustered Video Servers", *Proceedings of the 3rd IEEE International Conference on Multimedia Computing and Systems*, pp. 144-150, June 1996.

[3] M. Buddhikot and G. Parulkar, "Efficient Data Layout, Scheduling and Playout Control in MARS," *Multimedia Systems*, Vol.5, No. 3, pp.199-212, 1997.

[4] W. Bolosky et al."The Tiger Video Fileserver," *Proc. 6th International Workshop on Network and Operating System Support for Digital Audio and Video*, April 1996.

[5] R. L. Haskin, "Tiger Shark-A Scalable File System for Multimedia", *IBM Journal of Research Development*, Vol. 42, No. 2, pp. 185-198, March 1998.

[6] C. Bernjardt and E. Biersack, "The Server Array: A Scalable Video Server Architecture," *Hig -Speed Network for Multimedia Applications*, Kluwer Press, Boston, 1996.

[7] Y.B. Lee and P.C. Wong, "A Server Arr ay Approach for Video-On-Demand Service on Local Area Networks," *IEEE INFOCOM'96*, IEEE Computer Society Press, Los Alamitos, Calif, Vol. 1, pp.27-34, March 1996.

[8] A. Reddy. "Scheduling and Data Distribution in Multiprocessor Video Server," *Proc. Seco nd IEE International Conf. On Multimedia Computing and*

*Systems*, pp.256-263, 1995.

[9] M. Wu, W. Shu and K. Samuthiram, "Optimal Scheduling for Normal and Interactive Operations in Parallel Video Servers," *Proc. of Computer Software and Applications Conf.*, pp.290-295, Aug. 1997.

[10] P. Shenoy and H. Vin, "Efficient Striping Techniques for Multimedia File Servers," *Proc. of the 7th International Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 2 -36, May 1997.

[11] M.Y. Hsu, *A Simulation Model for Hierarchical Storage Systems of Video Servers* , Master Thesis, Department of Computer Science and Information Engineering, NCTU, June 1996.

[12] S.J. Cheng, *An Effective Data Placement Scheme for Supporting VCR Functionality in A Video Server*, Master Thesis, Department of Computer Science and Information Engineering, NCTU, June 1998.

[13] J.K. Chen, *Design and Implementation of An Effectiv Platform of Video Server Management System*, Master Thesis, Department of Computer Science and Information Engineering, NCTU, June 1998.

[14] *Video Server User Guide*, Institute of Mentor Data System, 1998.

[15] R. Tewari, D. Dias, R. Mukherjee and H. Vin, "Real-time Issues for Clustered Multimedia Servers," IBM *Research Report*, RC20020, April 1995.

[16] H. Cormen, E. Leiserson and L. Rivest, *Introduction to Algorithms*, pp.504-510. McGraw-Hill Book Company, 1989.