Journal of Parallel and Distributed Computing **58**, 425–445 (1999)
Article ID jpdc.1999.1560, available online at http://www.idealibrary.com on IDEAL®

Real-Time Communication Scheduling in a Multicomputer Video Server

A. L. Narasimha Reddy

Department of Electrical Engineering, Texas A & M University, 214 Zachry, College Station, Texas 77843-3128
E-mail: reddy@ee.tamu.edu

and

Eli Upfal

Department of Computer Science, Brown University, P.O. Box 1910, Providence, Rhode Island 02912

Received March 17, 1997; revised October 10, 1997; accepted April 27, 1999

In this paper, we address the problem of scheduling communication over the interconnection network of a distributed-memory multicomputer video server. We show that this problem is closely related to the problem of data distribution and movie scheduling in such a system. A solution is proposed in this paper that addresses these three issues at once. The movies are distributed evenly over all nodes of the multicomputer. The proposed solution minimizes the contention for links over the switch. The proposed solution makes movie scheduling very simple—if the first block of the movie is scheduled, the rest of the movie is automatically scheduled. Moreover, if the first block of the movie stream is scheduled without network contention, the proposed solution guarantees that there will be no network contention during the entire duration of playback of that movie. We show that the proposed approach to communication scheduling is optimal in utilizing the network resources. Extensive simulation results are presented to show the effectiveness of the proposed approach.

1. INTRODUCTION

Several telephone companies and cable operators are planning to install large video servers that would serve video streams to customers over telephone lines or cable lines. These projects envision supporting several thousands of customers with the help of one or several large video servers. These projects aim to store movies in a compressed digital format and route the compressed movie to the home where it can be uncompressed and displayed. These projects aim to compete with the local

0743-7315/99 $30.00

video rental stores with better service; offering the ability to watch any movie at any time (avoiding the situation of all the copies of the desired movie rented out already) and offering a wider selection of movies. Providing a wide selection of movies requires that a large number of movies be available in digital form. Currently, with MPEG-1 compression, a movie of roughly 90 minute duration takes about 1 GB worth of storage. A video server storing about 1000 movies (a typical video rental store carries more) would then have to spend about \$250, 000 just for storing the movies on disk at a cost of \$0.25/MB. This requirement of large amounts of storage implies that the service providers need to centralize the resources and provide service to a large number of customers to amortize costs. Hence, the requirement to build large video servers that would store a large number of movies in a single system and be able to service a large number of customers.

Multicomputer systems may be suitable candidates for supporting such large amounts of real-time I/O bandwidth required in these large video servers. Several problems need to be addressed for providing the required real-time I/O bandwidth in such a multicomputer system. In this paper, we outline some of the problems and their solutions particular to a video server based on multicomputers. In this paper, we will use the term multicomputer system to describe a system that may be variously known as a multicomputer or a clusterd system without a single address space.

2. THE PROBLEM

We will assume that the multicomputer video server is organized as shown in Fig. 1. A number of nodes act as *storage nodes*. Storage nodes are responsible for storing video data either in memory, disk, tape, or some other medium and delivering the required I/O bandwidth to this data. The system also has *network nodes*. These network nodes are responsible for requesting appropriate data blocks from storage nodes and routing them to the customers. Both these functions can reside on the same multicomputer node, i.e., a node can be a storage node, a network node, or both at the same time. Each request stream would originate at one of the several network nodes in the system and this network node would be responsible for obtaining the required data for this stream from the various storage nodes in the system.

To obtain high I/O bandwidth, data has to be striped across a number of nodes. If a movie is completely stored on a single disk, the number of streams requesting that movie will be limited by the disk bandwidth. As shown earlier by [1], a 3.5-inch 2-GB IBM disk can support up to 20 streams. A popular movie may receive more than 20 requests over the length of the playback time of that movie. To enable serving a larger number of streams of a single movie, each movie has to be striped across a number of nodes. As we increase the number of nodes for striping, we increase the bandwidth for a single movie. If all the movies are striped across all the nodes, we also improve the load balancing across the system since every node in the system has to participate in providing access to each movie. Hence, we assume that all the movies are striped across all the nodes in the system. Even when the movies are stored in semiconductor memory, the required communication and memory bandwidths may require that the movie be striped across the

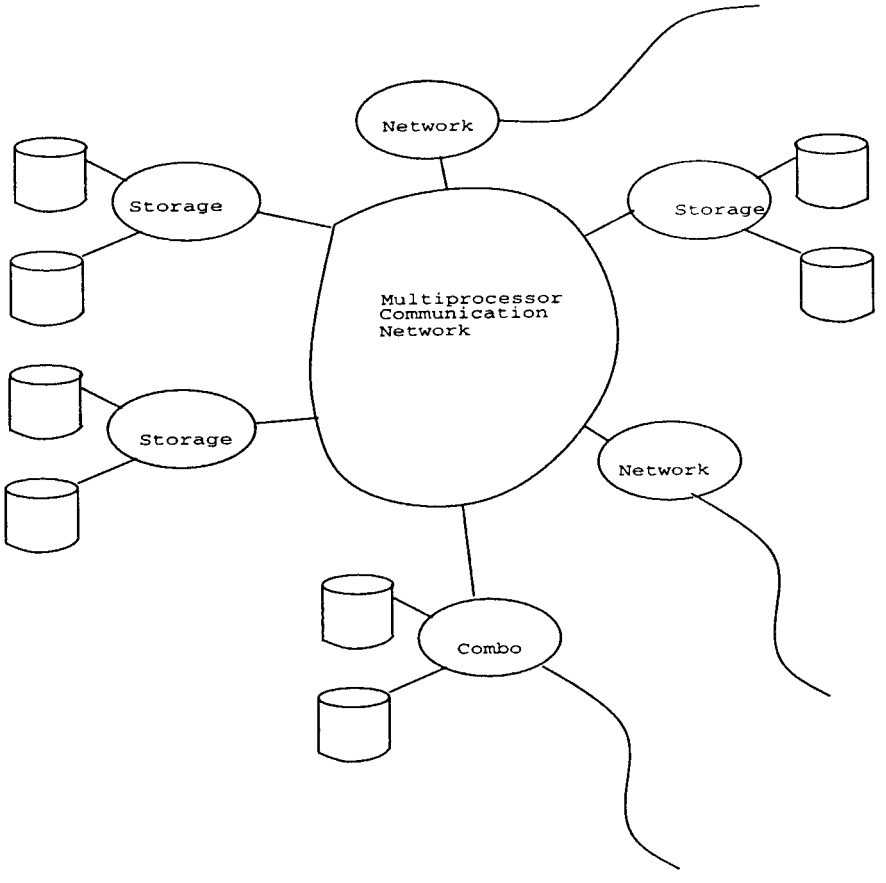


FIG. 1. System model of a multicomputer video server.

memories of different nodes of the system. For the rest of the paper, we will assume that the movies are striped across all the storage nodes. The unit of striping across the storage nodes is called a block. In our earlier studies on disk scheduling [1], we found that 64–256 Kbytes is a suitable disk block size for delivering high real-time bandwidth from the disk subsystem.

As a result of this, a network node that is responsible for delivering a movie stream to the user may have to communicate with all storage nodes in the system during the playback of that movie. This results in a point to point communication from all storage nodes to the network node (possibly multiple times, depending on the striping block size, the number of nodes in the system, and the length of the movie) during the playback of the movie. Each network node will be responsible for a number of movie streams. Hence, the resulting communication pattern is random point-to-point communication among the nodes of the system. It is possible to achieve some locality by striping the movies among a small set of nodes and the restriction that network nodes for a movie be among this smaller set of storage nodes.

We observe that the delivery of video data to the consumer requires three components of service: (1) reading of the request block from the disk to a buffer at the

storage node, (2) transmission of the block from the storage node to the network node over the multicomputer network, and (3) transmission of the block from the network node to the consumer's desktop. Since the last component of service would depend on the delivery medium (telephone wires, cable, or LAN), we will not address that issue here and will limit our attention to the service that has to be provided by the video server system, components (1) and (2).

A video server has to supply data blocks of the movie at regular periods to the consumer. If data is not transfered at regular intervals, the consumer may experience glitches in the delivery of the movie. To ensure glitch-free service, the video server has to guarantee finishing the three components of service in a fixed amount of time. Guaranteeing delay bounds in service component (1) is addressed by appropriate disk scheduling [1–4]. The problem of ensuring delay bounds in service component (2) is addressed in this paper.

When multiple transmissions take place simultaneously over the network, the delays experienced by individual transmissions depend on the contention experienced in the network. Worst case assumptions of contention are typically made to obtain guaranteeable delay bounds in a network. Our approach to this problem is to carefully schedule individual tranmissions over the network so as to minimize (or eliminate) the contention in the network. This approach enables us to guarantee tighter delay bounds on transmissions over the multicomputer network of the video server.

For the rest of the paper, we will assume that every node in the system is both a storage node and a network node at the same time, i.e., a combination node. We will use a multicomputer system with an Omega interconnection network as an example multicomputer system.

Movie distribution (or data distribution/organization) is the problem of 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 the communication patterns observed in the network. *Movie scheduling* is the problem of scheduling a storage node and a network node such that the required blocks of a movie stream arrive at the network node in time. At any given point in time, a node can be involved in sending one block of data and receiving one block of data. Movie scheduling is concerned with scheduling the transfer of blocks of a movie between storage nodes and the network node for that movie stream. *Communication scheduling* is a direct consequence of the movie scheduling problem. When two transfers are scheduled to take place between two different sets of source and destination pairs, the communication may not happen simultaneously between these pairs because of contention in the network.

Figure 2 shows a 16-node Omega network [5] built out of 4×4 switches. Figure 2 assumes unidirectional links going from left to right and, hence, each node has a sending port on one side and the receiving port on ther side in Fig. 2. We will assume that each node has a sending port and a receiving port and that a node can participate in a send and a recieve operation simultaneously. Communication cannot take place simultaneously between nodes 1 and 3 and nodes 9 and 2 in Fig. 2. Can movies be scheduled such that there is no contention at the source, at the destination and in the network? The communication scheduling problem deals with

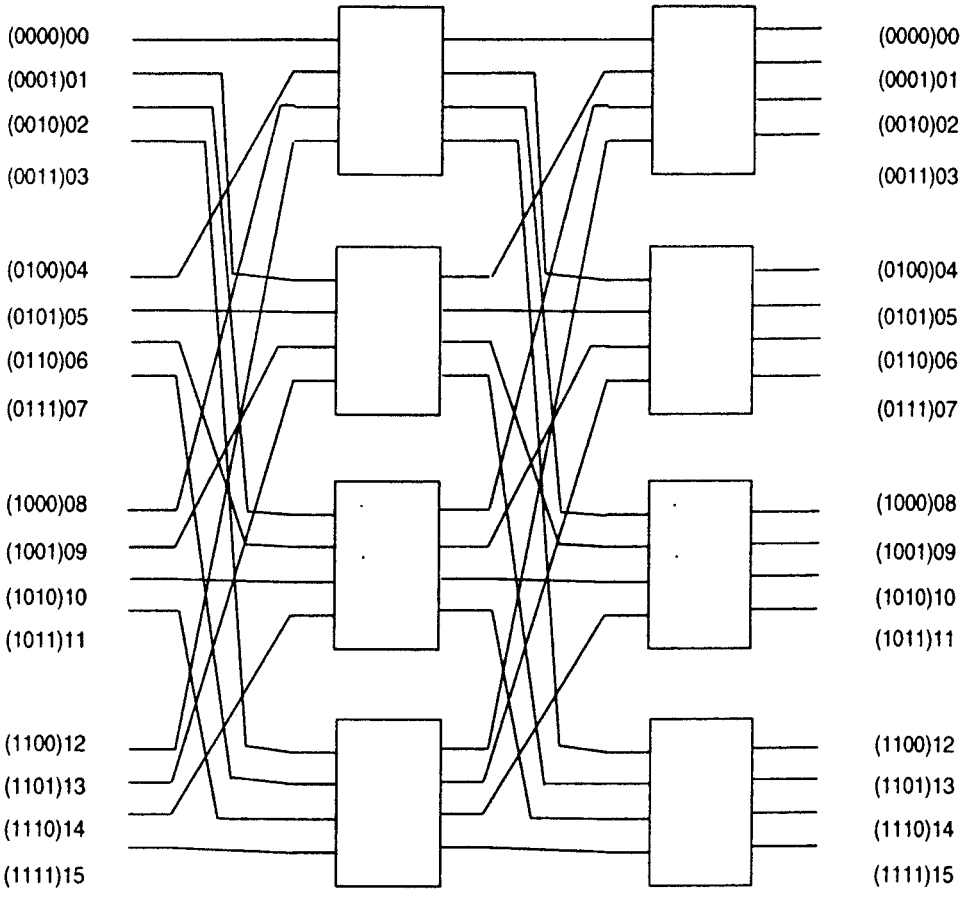


FIG. 2. A 16-node Omega network.

this issue of scheduling the network resources for minimizing the communication delays. This problem of scheduling communication over the multicomputer interconnection network is the focus of this paper. If the nodes in the multicomputer system are interconnected by a complete crossbar network, there is no communication scheduling problem since any pair of nodes in the system can communicate without a conflict in the network. The disk scheduling problem is dealt with at each node separately and we will assume that the system load is such that disk bandwidth is not a problem.

Deadline scheduling [6] is known to be an optimal scheduling strategy when the tasks are preemptable with zero cost and the task completion times are known in advance. Deadline scheduling is shown to be optimal even when the tasks are not preemptable [7]. But, both these studies assume that the task completion times are known in advance. The block transfer time in the network is dependent on whether there is any contention for the switch links and this contention varies, based on the network load. Also, a network transfer requires multiple resources (links, input and output ports) unlike the assumption of requiring only one resource in these studies. Hence, these results cannot be directly applied to our problem.

Recent work [2, 1, 3, 4, 8] has looked at disk scheduling in a video server. File systems for handling continuous media have been proposed [9–13]. Multicomputer based video servers are studied in [14–16]. Related work in multicomputer communication includes estimation of delays in the network [17–19], studies on network hotspots [20], and many application specific algorithms that exploit the network topology, for example, [21]. Other approaches to supporting multimedia network traffic include best-effort approaches [22, 23].

3. SCHEDULING PROCESS

3.1. Basic Approach

We will assume that time is divided into a number of *slots*. The length of a slot is roughly equal to the time taken to transfer a block of movie over the multicomputer network from a storage node to a network node (we will say more later on how to choose the size of a slot). Each storage node starts transferring a block to a network node at the beginning of a slot and this transfer is expected to finish by the end of the slot. It is not necessary for the transfer to finish strictly within the slot but for ease of presentation, we will assume that a block transfer completes within a slot.

The time taken for the playback of a block is called a *frame*. The length of the frame depends on the block size and the stream rate. For a block size of 256 Kbytes and a stream rate of 200 Kbytes/s, the length of a frame equals $256/200 = 1.28$ s. In this paper, we consider constant bit rate (CBR) streams that require a constant data rate. A block takes a slot for transfer from storage node to network node and takes a frame for playback at the client. Hence, a stream requires data transfer service across the multicomputer network for a slot in every frame. If we can provide guarantees that a block of data can be provided to the client every frame, with double buffering at the client, the client can playback the movie continuously. We will assume that the frame is an integral multiple (F) of the slot size. The slot size can be increased suitably to make this possible.

Now consider scheduling the communication required by a single request stream. We will assume that a single network node will coordinate the delivery of this request stream to the client. Let us assume that the data required by this stream k in frame j is stored in storage node s_j^k. Then the data transfer required by this stream can be represented by a sequence of pairs of the form (n^k, s_0^k), (n^k, s_1^k), and so on, where each pair represents the (destination, source) pairs involved in the communication in each frame and n^k is the network node involved in the delivery of this stream. For a single stream, the resulting traffic pattern is many (storage nodes) to one (network node). Each stream requires similar network service from the system. Hence, the network service for all the streams has a many to many traffic pattern. Data distribution among the storage nodes determines the exact traffic pattern. We will assume in this paper that a node can simultaneously participate in a send and receive operation.

The schedule in the system can then be represented by a table as shown in Fig. 3. Each row in the table represents a network node, each column represents a time

slot and each entry in the table represents the individual stream getting service and the storage node transmitting the required block to the network node n^k. Each stream's entries are separated by a frame (or F slots). If a stream has an entry in time slot j requiring service from storage node s_i and network node n^k, then that stream will have an entry in time slot $j+F$ requiring service from storage node s_{i+1} and network node n^k, where s_{i+1} is the storage node storing the next block of data for this stream. The order of storage nodes s_i, s_{i+1} is determined by the data distribution. To avoid scheduling conflicts at the nodes, a storage node cannot be scheduled twice in the same column of this table since that represents two transmissions in the same time slot. Conflicts in the network can be avoided if the pairs of nodes involved in communication in a slot (the entries in a column of the table) can be connected by edge disjoint paths in the network.

Now, the problem can be broken up into two pieces: (a) Can we find a data distribution that, given an assignment of (n^k, s_j^k) that is source and destination conflict-free, can produce a source and destination conflict-free schedule in slot $j+F$ (service required in the next frame)? and (b) Can we find a data distribution that, given an assignment of (n^k, s_j^k) that is source, destination, and network conflict-free, produce a source, destination, and network conflict-free schedule in slot $j+F$? The second part of the problem, (b), depends on the network of the multicomputer and that is the only reason for addressing the problem in two stages. We will propose a general solution that addresses (a). We then tailor this solution to suit the multicomputer network to address the problem (b).

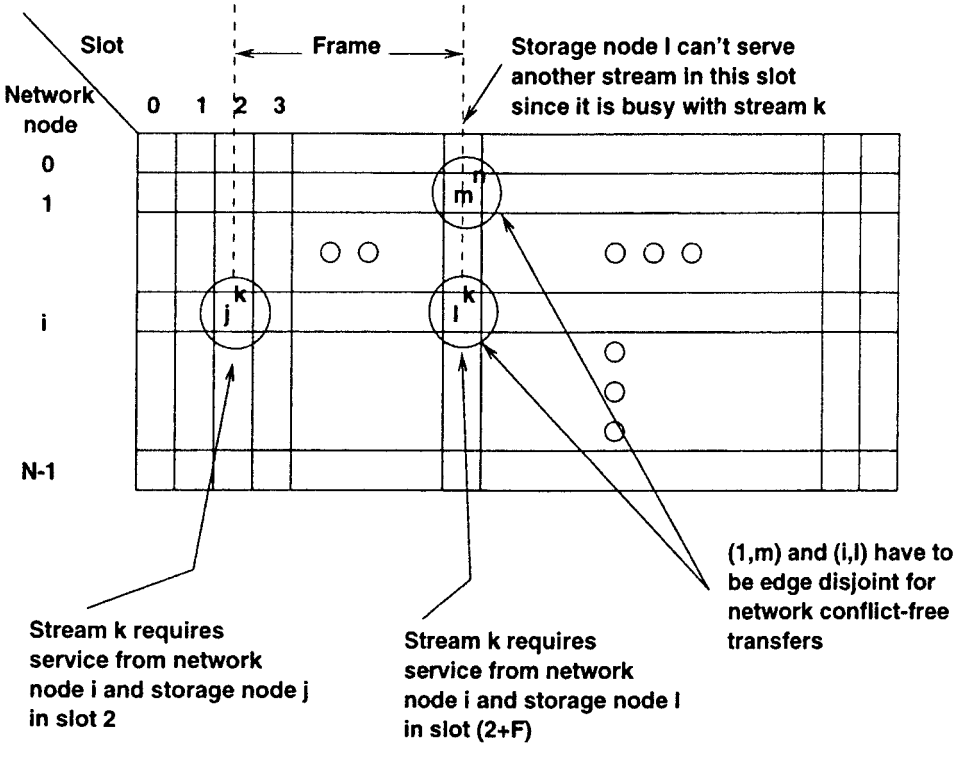


FIG. 3. Characteristics of a schedule table.

3.2. Movie Scheduling and Data Distribution

Assume all the movies are striped among the storage nodes starting at node 0 in the same pattern; i.e., block i of each movie is stored on a storage node given by $i \bmod N$, N being the number of nodes in the system. Then, a movie stream accesses storage nodes in a sequence once it is started at node 0. If we can start the movie stream, it implies that the source and the destination do not collide in that time slot. Since all the streams follow the same sequence of source nodes, when it is time to schedule the next block of a stream, all the streams scheduled in the current slot would request a block from the next storage node in the sequence and, hence, would not have any conflicts. In our notation, a set $\{n^k, s_j^k\}$ in slot j is followed by a set $\{n^k, (s_j^k + 1) \bmod N\}$ in slot $j + F$ in the next frame. It is clear that if $\{n^k, s_j^k\}$ is source and destination conflict-free, $\{n^k, (s_j^k + 1) \bmod N\}$ is also source and destination conflict-free. Variants of such data distributions have been proposed and analyzed [11, 8].

This simple approach makes movie distribution and scheduling straightforward. However, it does not address the communication scheduling problem. Also, it has the following drawbacks: (i) not more than one movie can be started in any given slot. Since every movie stream has to start at storage node 0, node 0 becomes a serial bottleneck for starting movies; (ii) when short movie clips are played along with long movies, short clips increase the load on the first few nodes in the storage node sequence, resulting in nonuniform loads on the storage nodes; (iii) as a result of (i), the latency for starting a movie may be high if the request arrives at node 0 just before a long sequence of scheduled busy slots.

The proposed solution uses one sequence of storage nodes for storing all movies. But, it does not stipulate that every movie start at node 0. We allow movies to be distributed across the storage nodes in the same sequence, but with different starting points. For example movie 0 can be distributed in the sequence of 0, 1, 2, ..., $N-1$, movie 1 can be distributed in the sequence of 1, 2, 3, ..., $N-1$, 0 and movie $k \pmod N$ can be distributed in the sequence of k, $k+1$, ..., $N-1$, 0,..., $k-1$. We can choose any such sequence of storage nodes, with different movies having different starting points in this sequence.

When movies are distributed this way, we achieve the following benefits: (i) multiple movies can be started in a given slot. Since different movies have different starting nodes, two movie streams can be scheduled to start at their starting nodes in the same slot. (ii) Since different movies have different starting nodes, even when the system has short movie clips, all the nodes are likely to see similar workload and, hence, the system is likely to be better load-balanced. Different short movie clips place the load on different nodes and this is likely to even out. (iii) Since different movies have different starting nodes, the latency for starting a movie is likely to be lower since the requests are likely to spread out more evenly.

The benefits of the above approach can be realized on any network. Again, if the set $\{n^k, s_j^k\}$ is source and destination conflict-free in slot j of a frame, then the set $\{n^k, (s_j^k + 1) \bmod N\}$ is given to be source and destination conflict-free in slot $j + F$, whether or not all the movies start at node 0. As mentioned earlier, it is possible to find many such distributions. Similar approaches to load balancing have been

adopted in [24, 11]. In the next section, it will be shown that we can pick a data distribution that can also guarantee conflict-free transfers in the network.

3.3. Communication Scheduling

The issues addressed in this section are specific to the network of the system. As mentioned earlier, we will use the Omega network as an example multicomputer interconnection network. This problem arose out of a need to build a VOD server on an IBM SP2 machine that employs an Omega network as an interconnection fabric. The solution described is directly applicable to hypercube networks as well. The same technique can be employed to find a suitable solution for other networks. We will show that the movie distribution sequence can be carefully chosen to avoid communication conflicts in the multicomputer network. The approach is to choose an appropriate sequence of storage nodes such that if movie streams can be scheduled in slot j of a frame without communication conflicts, then the consecutive blocks of those streams can be scheduled in slot $j+F$ without communication conflicts.

With our notation, the problem is to determine a sequence of storage nodes $s_0, s_1, ..., s_{N-1}$ such that given a set of nodes $\{n^k, s_j^k\}$ that are source, destination, and network conflict-free, it is automatically guaranteed that the set of nodes $\{n^k, s_{i+1}^k\}$ are also automatically source, destination, and network conflict-free.

First, let us review the Omega network. Figure 2 shows a multicomputer system with 16 nodes which are interconnected by a 16×16 switch with a single path between any pair of nodes. Figure 2 is an Omega network constructed out of 4×4 switches. To route a message from a source node whose address is given by $s_0 s_1 s_2 s_3$ to a destination node whose address is given by $d_0 d_1 d_2 d_3$, the following procedure is employed: (a) shift the source address left circular by two bits to produce $s_2 s_3 s_0 s_1$, (b) use the switch in that stage to replace $s_0 s_1$ with $d_0 d_1$, and (c) repeat the above two steps for the next two bits of the address. In general, steps (a) and (b) are repeated as many times as the number of stages in the network. Network conflicts arise in step (b) of the above procedure when messages from two sources need to be switched to the same output of a switch.

Now, let us address our problem of guaranteeing freedom from network conflicts for a set $\{n^k, s_{i+1}^k\}$, given that the set $\{n^k, s_i^k\}$ is conflict-free. Our result is based on the following theorem of Omega networks.

THEOREM 3.1. *If a set of nodes $\{n_i, s_i\}$ is network conflict-free, then the set of nodes $\{n_i, (s_i + a) \bmod N\}$ is network conflict-free for any a.*

Proof. Refer to [5]. ∎

The above theorem states that given a network conflict-free schedule of communication, then a uniform shift of the source nodes yields a network conflict-free schedule.

There are several possibilities for choosing a storage sequence that guarantees the above property. A sequence of 0, 1, 2, ..., $N-1$ is one of the valid sequences—a simple solution indeed! Let us look at an example. The set $S_1 = \{(0, 0), (1, 1), (2, 2), ..., (14, 14), (15, 15)\}$ of network-storage nodes is conflict free over the network (identity mapping). From the above theorem, the set $S_2 = \{(0, 1), (1, 2),$

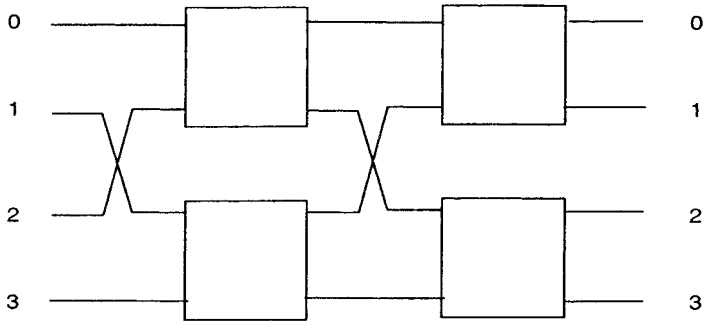
$(2, 3), ..., (14, 15), (15, 0)\}$ is also conflict-free and can be so verified. If S_1 is the conflict-free schedule in a slot j, S_2 will be the schedule in slot $j+F$, which is also conflict-free.

We have shown in this section that a simple round-robin distribution of movie blocks in the sequence of 0, 1, 2, ..., $N-1$ yields an effective solution for our problem. This data distribution with different starting points for different movies solves (a) the movie scheduling problem, (b) the load balancing problem, (c) the problem of long latencies for starting a movie, and (d) the communication scheduling problem.

Now, the only question that remains to be addressed is how does one schedule the movie stream in the first place, i.e., in which slot should a movie be started. When the request arrives at a node n^k, we first determine its starting node s_0^k based on the movie distribution. We look at each available slot j (where n^k is free and s_0^k is free) to see if the set of already scheduled movies do not conflict for communication with this pair. We search until we find such a slot and schedule the movie in that slot. Then, the complete length of that movie is scheduled without any conflicts.

An example schedule is shown in Figs. 4a, 4b, and 4c. The system has four nodes interconnected by an Omega network as shown in Fig. 4a. The example system has four nodes, 0, 1, 2, and 3, and it contains five movies, A, B, C, D, and E. The distribution of movies A, B, C, D, E across the nodes 0, 1, 2, and 3 is shown in Fig. 4b. For example, movie C is distributed cyclically across nodes in the order of 2, 3, 1, and 0. For this example, we will assume that the frame length $F=3$. Consider scheduling the following requests: (i) Movie C requested by network node 0 in slot 0, (ii) movie A requested by node 1 in slot 1, (iii) movie E requested by node 3 in slot 1, (iv) movie B requested by node 3 in slot 2, (v) movie A requested by node 1 in slot 3.

Since (i) is the first the request, it can be scheduled without any conflicts. We schedule the first block of the movie C from storage node 2 to network node 0 in slot 0. Subsequent blocks of this movie stream are scheduled in slots 3, 6, and 9 as shown in Fig. 4c. Each row in the schedule shows the blocks received by a node in different time slots. The entries in the table indicate the movie and the id of the sending node. The request (ii) for movie A can be scheduled in slot 1 since the nodes involved in the transfer 0 and 1 are free in slot 1. Request (iii) for movie E cannot be scheduled in slot 1 since it contends for the same storage node 0 as (ii). We can schedule (iii) in time slot 2. Request (iv) for movie B requires service between nodes 1 and 3 in slot 2. This causes a conflict at the network node with request (iii) in slot 2. We can schedule request (iv) in time slot 3 since there is no contention between already scheduled movie stream (i) in that slot. Request (v) for movie A in time slot 3 contends with the already scheduled request (iv) in the network in time slot 3. Looking forward, we find that request (v) can be scheduled in time slot 8 (network node 1 busy in time slot 4, network conflict with (3, 1) in time slot 5, storage node conflict in time slot 6, network node 1 busy in time slot 7). This example sequence illustrates how the scheduling takes place, avoiding conflicts at the nodes and in the network. It is to be noted that to schedule the video streams, we only considered the problem of scheduling the first block without any



a.

Movie/Blocks	0	1	2	3
A	0	1	2	3
B	1	2	3	0
C	2	3	1	0
D	3	0	1	2
E	0	1	2	3

b.

Req/Movie.Sender	Slot 0	1	2	3	4	5	6	7	8	9	10	11
0	2^{i}			3^{i}			0^{i}			1^{i}		
1		0^{ii}			1^{ii}			2^{ii}	0^{v}		3^{ii}	1^{v}
2												
3			0^{iii}	1^{iv}		1^{iii}	2^{iv}		2^{iii}	3^{iv}		3^{iii}

c.

FIG. 4. a. Example of a 4-node VOD system. b. Movie distribution. c. A complete example schedule.

conflicts. Once a conflict-free slot is found for the first block, the rest of the movie did not have any conflicts with the already scheduled streams.

4. ANALYSIS OF THE SCHEDULING PROCESS

We have proposed using the packet-switched network of IBM's SP2 in a time-division multiplexing mode to guarantee delay bounds on individual transmissions. Does this approach utilize the network resources optimally? Is it possible that the proposed approach fails in cases where a packet switched operation may effectively guarantee the required delay bounds? In this section, we will show that the proposed approach is optimal in utilizing network resources.

Assume that there are N storage nodes, and N network nodes (or N nodes that are both storage and data nodes) connected by an Omega network. Nodes are numbered 0, .., $N-1$. Movies are partitioned into blocks of data. The data is distributed between the data nodes. The first block of a movie may reside in an arbitrary

node (i.e. different movies start at different data nodes). However, the order in which the blocks are distributed between the data nodes is the same, in particular we assume, without loss of generality, that if a given block of a movie is at node i, the next block is at data node $i+1 \pmod N$.

A frame has k slots and each movie stream needs to be served once in each frame. A stream is said to be *schedulable* if the stream can be scheduled with no conflicts at the storage node, network node, and in the network.

A schedule is *legal* if every movie stream is served at least once in each frame, and if the set of pairs (data-node, network-node) that have to communicate in each slot can be connected by edge-disjoint paths on the network. A legal schedule guarantees that each scheduled movie stream receives sufficient service from the system and that the scheduled transmissions can take place without any conflict in the network.

Consider the proposed scheduling process: a new request is scheduled in the first possible slot (i.e. the first slot in which the requesting network node can be connected by an edge-disjoint path to the data node storing the first block of the movie, without changing any other scheduled communication). Once communication for the first block has been scheduled, subsequent communication steps are scheduled at intervals of exactly k slots.

It is easy to verify that this schedule is legal (applying Theorem 1). By the nature of the scheduling process, the transmissions do not conflict in the first slot when this movie stream is scheduled. During the next frame, when this movie stream needs service, all the communication pairs (s_i, n_i) would have shifted to (s_i+1, n_i). From Theorem 1, if the set $\{(s_i, n_i)\}$ is conflict-free, then the set $\{(s_i+1, n_i)\}$ is also conflict-free and, hence, the schedule is legal.

We prove next that this simple, on-line scheduling procedure gives optimal utilization of the network bandwidth.

Since each movie needs to be serviced at least once in each frame and a data node can serve only one request per step, it is clear that no communication process (packet routing or circuit switching) can satisfy more than k requests per network node in each frame. The following theorem proves that the above scheduling process can satisfy any set of requests up to k requests per network node, thus proving that the proposed policy provides as much bandwidth as any other scheduling procedure for this problem.

CLAIM. *Let G be a directed acyclic network. Assume that the in-degree and out-degree of each internal node in G are equal. Assume that a set of inputs in G are connected by edge disjoint path to a set of outputs, and assume that an input node v does not participate in any path. There is a path, edge disjoint from all the existing paths, connecting v to some output node.*

Proof. Node v is connected to some internal node. Since the in degree of that node equal its out-degree it has a free outgoing edge. Either this out-going edge is connected to an output node, or it is connected to another switch. Repeating this argument we eventually obtain a path from v to an output node, that is disjoint of the other existing paths. ▮

An Omega network can be seen as an acyclic graph with output ports of the nodes being connected by unidirectional links to the input ports of the nodes as shown in Fig. 2. Hence, given any set of existing edge-disjoint transfers through the network, we can connect an idle storage node (output port) to an idle network node (input port) without any conflicts within the network.

THEOREM 4.1. *Assume that a set of movie streams has already been scheduled and that a network node v is processing less than k requests. As long as node v has no more than k requests, any new request can be satisfied within Nk slots from the time it arrives (i.e., the maximum delay in transmitting the first block of any movie is* $Nk-1$).

Proof. Assume that the new request arrived at slot t. Since node v is serving less than k requests, there is a slot t', in the interval $t, t+1, ..., t+k-1$ in which node v is not active. Since the Omega network satisfies the conditions of the above fact, v can be connected at slot t' to some output node w. At time $t'+k$, v can be connected to output node $w+1 \pmod N$ and so on. Thus, there is a slot in the interval $t, ..., t+Nk-1$ in which network node v can be connected to the data node that contains the first block of the requested movie, and the rest of the movie is scheduled in intervals of k slots. ▌

Based on the above theorem, we can schedule a request at a network node, as long as one of the k slots in a frame is not busy at that network node. Since every transmission requires the data to be sent from a data node, the network resources can be fully utilized. We will present results from simulations to give an idea of the delays that may be experienced in finding an empty slot, i.e. latency for scheduling the first block of the movie stream.

5. SIMULATION RESULTS

Our analysis of the scheduling algorithm has shown that the proposed policy is optimal in utilizing network resources. This section presents simulation results to show that satisfactory latencies can be achieved for starting a movie stream using such an approach. We simulated a 16-node Omega network built out of 4×4 switches. The system is assumed to contain 320 movies. The movies are distributed uniformly across all the nodes in the system with the starting blocks of the movies distributed in a round-robin (or circular) order among the the nodes. Requests are assumed to arrive uniformly at all the (network) nodes. Several distributions of requests to different movies are considered as described below.

We considered a block size of 256 Kbytes. At a stream rate of 200 KB/s, frame length is then equal to 256 KB/200 KB/s = 1.28 s. A slot size of 6.4 ms (with 40 MB/s sustained network throughput of IBM SP2, 256 KB/40 MB/s = 6.4 ms) was chosen. Since the size of the frame is an integer multiple of slot size (1.28 s/6.4 ms = 200), no further adjustments of slot size were necessary. Since there are 16 nodes in the system, the system's capacity is $16 * 200 = 3200$ movie streams. The system was simulated such that 80% of the slots were utilized for delivering movie streams. We will show later that at higher operating points, movie schedulability

decreases rapidly. At 80% capacity, 2560 movie streams are scheduled. We simulated scheduling 2560 random movie streams to reach the 80% operating capacity. Once this operating point is reached, a previously scheduled movie stream is deleted randomly (with uniform probability) from the schedule and a new movie stream is started randomly at one of the network nodes. Based on the distribution of the movies, this would generate a request for the first block of that movie. This request is scheduled as soon as possible without altering the schedule of the streams scheduled earlier. The movie scheduling followed the procedure described earlier; it looks at all the slots where the storage node (of the first block of the movie stream) and the network node are free to see if this block can be scheduled without conflicting with the already scheduled streams in the network (both at the switches and the links). By deleting one request and adding a new request, we maintain the operating capacity at 80%. The latency required to schedule the first block of the newly generated request is noted. We generated 100,000 such new requests (along with deletions of old requests).

Startup latency, the latency to schedule the first block of a request stream, is the primary performance measure used in the simulations. If a stream request arrives in time slot t and the first block of the stream is scheduled in slot t', then the startup latency $= t' - t$ slots. The latency distribution of such requests is shown in Fig. 5. It is to be noted that the frequency axis is in log scale. Most of the requests observe very short latencies and the percentage of requests experiencing long latencies is quite small. We also calculated the average latency and the 99th percentile

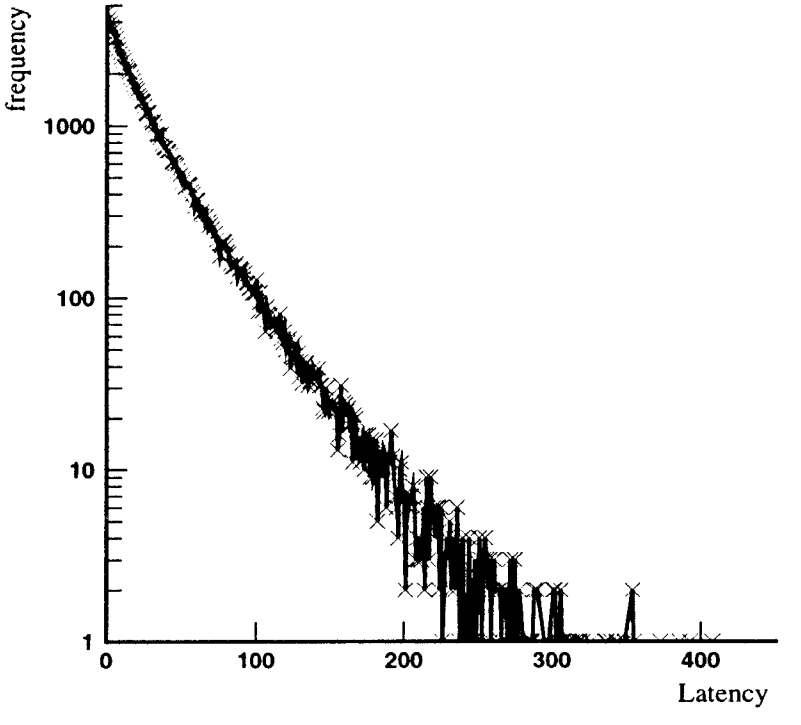


FIG. 5. Startup latency distribution at 80% operating capacity.

TABLE 1

Startup Latency Distribution with Uniform Movie Distribution

Performance measure	Slots	Seconds
Average latency	26.757	0.171
90th-percentile latency	65	0.416
95th-percentile latency	89	0.570
99th-percentile latency	150	0.960
Maximum latency	408	2.611

latency of this distribution. These are shown in Table 1. It is observed that the average latency observed by the new requests is quite low, less than 171 ms. Very few requests, less than 1%, observe latencies above 0.960 s. The latencies are quite tolerable for starting the first block of a movie in a video server. Since our goal in this paper is a video-on-demand service, even the maximum latency of 2.61 s is quite tolerable for starting a movie. It is possible to reduce this latency further by shifting already scheduled streams around (without violating their deadline requirements) to make room for an arriving request. It has been shown that such techniques are quite effective in reducing the startup latencies to a few slots [25].

To study the impact of different distributions of requests to movie streams, we considered the following cases: (1) 80% of the requests go to 20% of the movies, (2) 90% of the requests go to 10% of the movies, and (3) 95% of the requests go to 5% of the movies. The results for these three cases, from a statistical point of view, were identical, as shown in Table 2. In Tables 1 and 2, x% of the requests experience a startup latency delay below the xth-percentile latency shown. Even though the requests for movies are nonuniformly distributed, the block requests were uniformly distributed among the nodes since the movies were randomly distributed among the nodes. The movies were ranked based on the frequency of use, and the starting block of the movies were distributed in a round-robin fashion among the nodes in order of their rank. This results in distributing the load evenly across all the nodes by distributing the frequently accessed movies across all the nodes. For example, with the 95–5 distribution, with 320 movies in the system, the most frequently watched movies = 5% of the movies = 16 movies. If we distribute the starting blocks of these 16 movies across 16 nodes and the starting blocks of the

TABLE 2

Startup Latency Distribution with 95–5 Movie Distribution

Performance measure	Slots	Seconds
Average latency	26.963	0.173
90th-percentile latency	65	0.416
95th-percentile latency	90	0.576
99th-percentile latency	151	0.966
Maximum latency	436	2.791

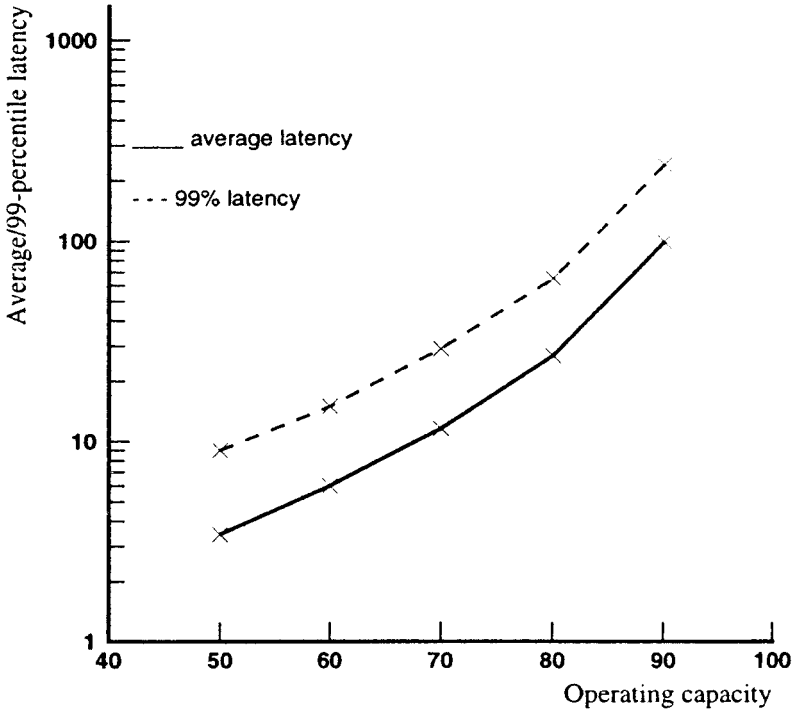


FIG. 6. Impact of lead on startup latency.

other 95% movies uniformly across 16 nodes, it is still possible to maintain a uniform load across all the nodes. From the results, it is observed that the averages were slightly higher, compared to results in Table 1, but not significantly different.

So far, we have considered a system that is operating at 80% capacity. We varied the operating capacity from 50% to 90% to study the impact of load on the results. Figure 6 shows the impact on average latencies (solid lines) and 99th-percentile latencies (dashed lines) as the operating capacity is varied from 50% to 90% with different load imbalances. At low operating capacities of up to 70%, the latencies are quite low and do not vary significantly. When the load is increased up to 80% and beyond, the latencies grow considerably. These results indicate that it is possible to obtain lower latencies by simply utilizing the reources to a lesser extent. This is called overdesigning or overconfiguring the system.

To see the impact on schedulability, we artificially created load imbalance across the nodes in the system. Load is evenly balanced across all nodes when movies are striped across all nodes in the system, since a movie requires service from every storage node during playback. To create an unbalanced load, we considered a system where data is striped across two sets of $N/2$ nodes (please see Section 6.3 for more details on how this can be done). In these experiments, half the nodes received $x\%$ of the requests and the other half of the nodes received $(100-x)\%$ of the requests. When $x=50\%$, the system is load balanced. When $x=60\%$, half the nodes received 60% of the requests, and the other half of the nodes received 40% of the requests, resulting in an unbalanced load. We considered values of 50, 55, 60,

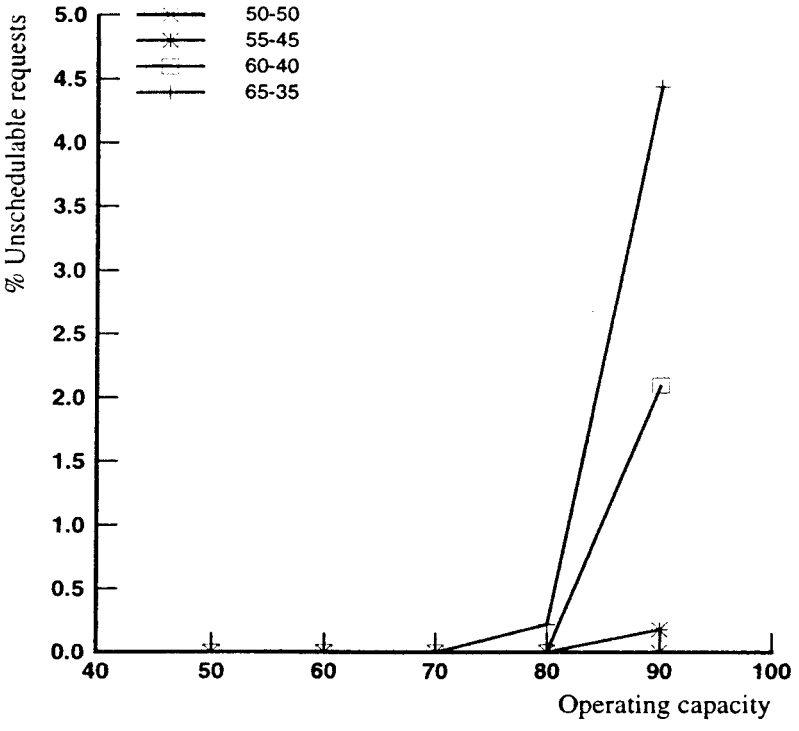


FIG. 7. Impact of load on stream schedulability.

and 65 for x. When $x = 65\%$, half the nodes service nearly twice as many requests as the other half of the nodes.

Figure 7 shows the number of requests that could not be scheduled (out of 100,000 attempted) as a function of load. At higher loads, not all the requests could be scheduled, especially when the system is not load balanced. Even at 90% of operating capacity, the number of unschedulable requests is below 5%. Up to an operating capacity of 70%, all the requests could be scheduled, even with the most unbalanced system considered. Figures 6 and 7 point to the importance of keeping the system load balanced. If the load can be balanced across the system, the system can support a larger number of streams (operate at higher capacity) and offer better latencies than an unbalanced system.

6. OTHER ISSUES

6.1. Clock Synchronization

Throughout the paper, it is assumed that the clocks of all the nodes in the system are somehow synchronized and that the block transfers can be started at the slot boundaries. If the link speeds are 40 MB/s, a block transfer of 256 Kbytes requires 6.4 ms, quite a large period of time compared to the precision of the node clocks which tick every few nanoseconds. If the clocks are synchronized to drift at most, say 600 μs, the nodes observe the slot boundaries within $\pm 10\%$. During this time, it is possible that the block transfers observe collisions in the network. But during

the rest of the 90% transfer time in the slot, the block transfers take place without any contention over the network. This shows that the clock synchronization requirements are not very strict. It is possible to synchronize clocks to such a coarse level by broadcasting a small packet of data at regular intervals to all the nodes through the switch network. The nodes of IBM's SP2 machine can be synchronized to within a few microseconds of each other [26].

6.2. *Different Stream Rates*

We have assumed that all the request steams require the same data rate from the system. However, due to different compression standards or due to different qualities of compression, different streams can have different data rate requirements. Given a stream rate and the design parameter of frame size, we can calculate the amount of data that has to be retrieved by the system to keep the client busy for a frame. Then slots can be alloted to accommodate the required amount of data. For example, for realizing a 3 Mbits/s stream rate, 2 slots are assigned to the same stream within a frame if a slot can fetch enough data for a 1.5 Mbit/s stream (as in our earlier example). These two slots are scheduled as if they are two independent streams, making sure that the assigned second slot is not more than half a frame behind the first slot (otherwise, stream starvation results). The only difference is that the network node assigns a larger number of block buffers and handles them differently than with a stream at the basic rate. When the required stream rate is not a multiple of the basic stream rate, a similar method can be utilized with the last slot of that stream, not necessarily transferring a complete block. The complexity of buffer management increases at the network node.

6.3. *Different Network Nodes and Storage Nodes*

We have assumed so far that all the nodes in the network are combination nodes. Even when this is not the case, it is possible to find mappings of network nodes and storage nodes to the multicomputer nodes that guarantee freedom from network conflicts. For example, assigning the network nodes to the even addresses and the storage nodes to the odd addresses in the network and distributing the movies in round-robin fashion among the storage nodes yields similar guarantees. For example, assigning storage nodes to addresses of (0, 2, 4, 6) and the network nodes to (1, 3, 5, 7) in an 8-node network provides similar guarantees. The communication pattern for a movie stream originating at node 1 will then follow a circular order of ...(1,0), (1,2), (1,4), (1,6), (1,0), (1,2).... The set of communicating pairs $\{n_i, s_i\}$ in one slot are followed by the set $\{(n_i), (s_i + 2) \bmod N\}$ in the next frame and, hence, by Theorem 1 (with shift $a = 2$) are guaranteed to be conflict-free if the original set was conflict-free. Similarly, it can be shown that it is possible to provide conflict-free delivery when data is striped across half of the storage nodes.

6.4. *Choosing a Slot Size*

Ideally, we would like all block transfers to complete within a slot. However, due to variations in delivery time all the block transfers may not finish within a slot.

Transfer delay variations are minimized by the proposed approach through careful scheduling and minimization of network conflicts. The network adapters at the source and the destination can still add variations to the delivery time due to memory bandwidth conflicts. When the transfer time exceeds a slot, different transfers may contend for the same link in the network. To avoid such problems, it will be necessary to overestimate the slot size. It is also pointed out earlier that to make the frame an integral multiple of the slot size, we may have to overestimate the slot size (since exact values may not be multiples). This larger slot size would also mitigate the possible effects of the drift among clocks at the nodes in the network.

6.5. *Communication Overhead*

As discussed earlier, wide striping of data across all the nodes in the system results in all-to-all communication patterns. For example, a network node serving a movie stream communicates with all storage nodes in the system during the life of that stream. It is possible to localize the communication to a small set of nodes. For example, if the movie is striped across 4 nodes of the system, the network node would only have to communicate with those 4 storage nodes. However, this localization is achieved at a cost of reduced bandwidth to movies stored on these 4 storage nodes. The number of streams to movies on these 4 storage nodes will be limited. This localization also results in load imbalance at the storage node disks along with imbalance in communication across the network, storage nodes with popular movies receiving more demand than others. Wide striping, as explained above, makes the load uniform and predictable at the disks and across the network, making it easier to schedule streams for guaranteed service.

It is also to be pointed out that in multistage interconnection networks such as an Omega network, communication between any pair of nodes takes the same number of hops (links and switches) across the network. Localization has little benefit in reducing the load across such networks. However, indirect networks, such as hypercubes, can benefit from localization of communication traffic.

6.6. *Other Interconnection Networks*

The proposed solution may be employed even when the multicomputer system is interconnected by a network other than an omega network. To guarantee conflict-free transfers over the network, appropriate data distributions for those networks have to be designed. For the hypercube type of networks that can emulate an omega network, the same data distribution provides similar guarantees as in the Omega network. It can be shown that if movie blocks are distributed uniformly over all nodes in a hypercube in the same order 0, 1, 2, ..., $n-1$ (with different starting nodes), a conflict-free schedule in one slot guarantees that the set of transfers required a frame later would also be conflict-free. The Oracle's media server uses NCube's hypercube machine [14].

7. SUMMARY

Deadline scheduling is normally used in most of the real-time applications. For our application here, each network node needs to send a request with the earliest deadline to the storage node holding that block. Each storage node serves the request with the earliest deadline. The network also needs to implement deadline scheduling for switching packets through the network. Also, even if all the components of the system could implement deadline scheduling, the following issues will remain to be addressed: (a) it is possible for two storage nodes to initiate two transfers that collide in the network. Then, one of the transfers gets delayed if deadline scheduling is employed (and thus, idling that storage node) or both the transfers take twice as long if round-robin scheduling is employed. In this scenario, the scheduling decisions are being independently made at the storage nodes and the switches in the network. How does one avoid such scenarios or how does one do centralized deadline scheduling (without the knowledge of the current packets in the network)? (b) when should a movie be started?

Our approach/solution addressed a large number of these issues which remain to be addressed in other approaches. In this paper, we presented a unified solution for data distribution, movie scheduling, and communication scheduling in a multicomputer video server. The solution is proposed in two stages. The first step argued that a simple regular pattern for storing movies across storage nodes with different starting nodes can reduce the movie scheduling problem to a simpler problem of scheduling the first block of the movie without any adverse affects of the earlier proposals. The second step consists of deriving such a sequence such that communication conflicts are minimized in the network. We exploited the network topology of the multicomputer to derive such a sequence that guarantees freedom from communication conflicts if the first block of the movie is scheduled without any communication conflicts. We presented extensive simulation results to show the effectiveness of the proposed approach. We also addressed some real world issues such as load balancing, and clock synchronization. We have shown that most of the issues can be satisfactorily addressed. In situations where other issues may be present, the proposed solution can be used as a starting point for reducing (rather than guaranteeing freedom from collisions) the communication scheduling problems in a video server.

ACKNOWLEDGMENTS

Discussions with Roger Haskin and Jim Wyllie have contributed to our understanding of the problem. Reviewers comments have helped to improve the presentation of the paper.

REFERENCES

1. A. L. Narasimha Reddy and J. Wyllie, Disk scheduling in a multimedia I/O system, *in* "Proc. of ACM Multimedia Conf.," Aug. 1993.
2. D. P. Anderson, Y. Osawa, and R. Govindan, Real-time disk storage and retrieval of digital audio/video data, Tech. Rep. UCB/CSD 91/646, Univ. of California, Berkeley, Aug. 1991.

3. P. S. Yu, M. S. Chen, and D. D. Kandlur, Grouped sweeping scheduling for dasd-based multimedia storage management, *Multimedia Systems* **1** (1993), 99–109.
4. J. Yee and P. Varaiya, Disk scheduling policies for real-time multimedia applications, Tech. report, Univ. of California, Berkeley, Aug. 1992.
5. D. H. Lawrie, Access and alignment of data in an array processor, *IEEE Trans. Comput.* **C-24**, 12 (Dec. 1975), 1145–1155.
6. C. L. Liu and J. W. Layland, Scheduling algorithms for multiprogramming in a hard real-time environment, *J. Assoc. Comput. Mach.* (1973), 46–61.
7. K. Jeffay, D. F. Stanat, and C. U. Martel, On non-preemptive scheduling of periodic and sporadic tasks, "Proc. of Real-time Systems Symp.," pp. 129–139, Dec. 1991.
8. S. Berson, S. Ghandeharizadeh, and R. Muntz, Staggered striping in multimedia information systems, "Proc. of SIGMOD," 1994.
9. H. M. Vin and P. V. Rangan, Designing file systems for digital video and audio, "Proc. of 13th ACM Symp. on Oper. Sys. Principles," 1991.
10. D. Anderson, Y. Osawa, and R. Govindan, A file system for continuous media, *ACM Trans. Comput. Systems* (Nov. 1992), 311–337.
11. R. Haskin, The shark continuous-media file server, "Proc. of IEEE COMPCON," Feb. 1993.
12. F. A. Tobagi, J. Pang, R. Biard, and M. Gang, Streaming raid: A disk storage system for video and audio files, "Proc. of ACM Multimedia Conf.," pp. 393–400, Aug. 1993.
13. C. Martin, P. Narayanan, B. Ozden, R. Rastogi, and A. Silberschatz, The Fellini multimedia storage server, *in* "Multimedia Information Storage and Management" (S. Chung, Ed.), Kluwer, New York, 1996.
14. A. Laursen, J. Olkin, and M. Porter, Oracle media server: Providing consumer based interactive access to multimedia data, *in* "Proc. of SIGMOD," pp. 470–477, 1994.
15. Microsoft, The tiger video server, *Microsoft Press Release*, Apr. 1994.
16. D. Jadav, A. Choudhary, and B. Berra, An evaluation of design trade-offs in a high-performance, media-on-demand server, *ACM Multimedia Systems J.*, Jan. 1997.
17. C. P. Kruskal, M. Snir, and A. Weiss, On the distribution of delays in buffered multistage interconnection networks for uniform and nonuniform traffic (extended abstract), *Int. Conf. on Parallel Processing* (1984), pp. 215–219.
18. J. H. Patel, Performance of processor-memory interconnections for multiprocessors, *IEEE Trans. Comput.* **C-30** (Oct. 1981), 771–780.
19. D. M. Dias and R. Jump, Analysis and simulation of buffered delta networks, *IEEE Trans. Comput.* **C-30**, No. 4 (April 1981), 273–282.
20. G. F. Pfister and V. A. Norton, Hot spot contention and combining in multistage interconnection networks, *IEEE Trans. Comput.* **C-34**, No. 10 (Oct. 1985), 943–948.
21. J. Jang, M. Nigam, V. Prasanna, and S. Sahni, Constant time algorithms for computational geometry on the reconfigurable mesh, *IEEE Trans. Parallel Dist. Systems* (Jan. 1997).
22. B. Smith, Cyclic-UDP: A priority-driven best-effort protocol, Tech. Report, Cornell University, 1996.
23. K. Jeffay, D. Stone, T. Talley, and F. Smith, Adaptive, best-effort delivery of digital audio and video across packet-switched networks, *in* "Proc. Network and Operating System Support for Digital Audio and Video," 1993.
24. S. Berson, L. Golubchik, and R. Muntz, Fault tolerant design of multimedia servers, UCLA Technical Report, 1994.
25. A. L. Narasimha Reddy, Improving the interactive responsiveness in a video server, *in* "Multimedia Information Storage and Management," Kluwer Academic, New York, 1996.
26. IBM, IBM SP2 product literature, 1994.