Scheduling and data distribution in a multiprocessor video server *

A. L. Narasimha Reddy

IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120.
reddy@almaden.ibm.com

Abstract

In this paper, we will address the problem of distributing and scheduling movies on a multiprocessor video server. We will also address the issue of scheduling communication over the multiprocessor switch for the playback of the scheduled movies. A solution is proposed in this paper that addresses these three issues at once. The proposed solution minimizes 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.

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. Multiprocessor systems may be suitable candidates for supporting 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 multiprocessor system. In this paper, we outline some of the problems and some solutions.

In this paper, we will address the problems of data organization and scheduling in a multiprocessor video server. We will assume that the multiprocessor 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 multiprocessor node, i.e., a node can be a storage node, or 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.

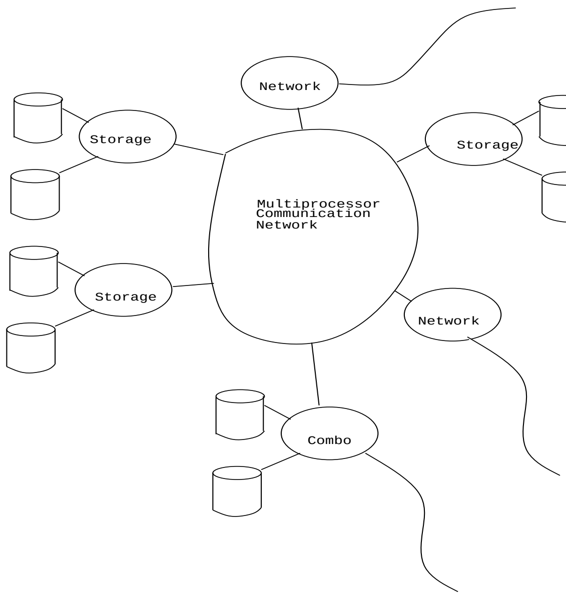

Fig. 1. System model of a multiprocessor video server.

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" 2-GB IBM disk can support upto 20 MPEG-1 streams. A popular

*Proc. of 2nd IEEE Int. Conf. on Multimedia Computing and Systems, May 1995 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. The unit of striping across the storage nodes is called a block. In our earlier studies on disk scheduling [1], we found that 256 Kbytes is a suitable disk block size for delivering high real-time bandwidth from the disk subsystem.

A network node that is responsible for delivering a movie stream to the user may have to communicate with all the storage nodes in the system during the playback of that movie. This results in a point to point communication from all the 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 is responsible for a number of movie streams. Hence the resulting communication pattern is random point-to-point communication among the nodes of the system.

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 multiprocessor system with an Omega interconnection network as an example multiprocessor system.

Movie (data) distribution 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. *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. Fig. 2. shows a 16-node Omega network [2] built out of 4x4 switches. 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 and the destination and in the network? Communication scheduling problem deals with this issue of scheduling the network resources for minimizing the communication delays. If the nodes in the multiprocessor 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. Disk scheduling problem is dealt at each node separately and we will assume that the system load is such that disk bandwidth is not a problem.

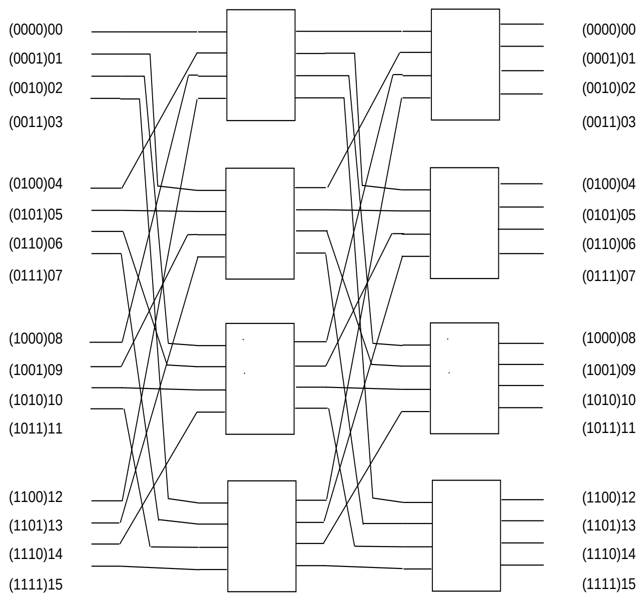

Fig. 2. A 16-node Omega network.

Recent work [3, 1, 4] has looked at disk scheduling in a video server. File systems for handling continuous media have been proposed in [5, 6, 7, 3]. Traditional deadline scheduling [8] techniques cannot be directly applied to this problem because the network transfer times are not constant and vary with the network load. Simple deadline scheduling of network and storage nodes also doesn't avoid the possible conflicts in the network between two simultaneous transfers.

2 Some notation

We will assume that time is divided into a number of *slots*. The length of a slot is roughly equal to the average time taken to transfer a block of movie over the multiprocessor network from a storage node to a network node. Average delivery time itself is not enough in choosing a slot; we will comment later on how to choose the size of a slot. For now, consider that time is divided into a number of slots. 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 movie 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/sec, the length of a frame equals 256/200 = 1.28 seconds. We will assume that a basic stream rate of MPEG-1 quality at 1.5Mbits/sec is supported by the system. When higher stream rates are required, multiple slots are assigned within a frame to achieve the required delivery rate for that stream.

For a given system, the block size is chosen first. For a given basic stream rate, the frame length is then determined. Slot width is then approximated by dividing the block size by the average achievable data rate between a pair of nodes in the system. This value is adjusted for variations in communication delay. Also, we require that frame length be an integer multiple of the slot width. From here, we will refer to the frame length in terms of number of slots per frame 'F'.

Now, the complete schedule of movies in the system can be shown by a table as shown in Fig. 3. The example system has 4 nodes, 0, 1, 2, and 3 and contains 5 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. 3 (a). For example, movie E is distributed cyclically across nodes in the order of 2, 1, 0, and 3. For this example, we will assume that the frame length F = 3. Now, if movie E needs to be scheduled at node 0, data blocks need to be communicated from nodes 2, 1, 0 and 3 to node 0 in different slots. This is shown in Fig. 3(b) where the movie is started in slot 0. Fig. 3(c) shows a complete schedule of 4 requests for movies E, C, B, and E that arrived in that order at nodes 0, 1, 2, 3 respectively. 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. Each column should not have a sending node listed more than once since that would constitute a conflict at the sender. A movie stream has its requests listed horizontally in a row. The blocks of a single stream are always separated by F slots, in this case F = 3. Node 0 schedules the movie to start in time slot 0. But node 1 cannot start its movie stream in slot 0 as it conflicts with node 0 for requesting a block from the same storage node 2. Node 2 can also schedule its movie in slot 1. Node 3 can only schedule its movie in slot 2. Each request is scheduled in the earliest available slot. The movie stream can be started in any column in the table as long as its blocks do not conflict with the already scheduled blocks. The schedule table is wrapped around i.e., Slot 0 is the slot immediately after Slot 11. For example, if another request arrives for movie E at node 2, we can start that request in time Slot 3, and schedule the requests in a wrap-around fashion in time Slots 6, 9, and 0 without any conflict at the source and the destination. The schedule table has FN slots, where N is the number of storage nodes in the system.

The schedule can be be represented by a set (n_{ij}, s_{ij}), a set of network node and storage node pairs involved in a block transfer in slot j. If we specify F such sets for the F slots in a frame (j = 1,2,...F), we would completely specify the schedule. If a movie stream is scheduled in slot j in a frame, then it is necessary to schedule the next block of that movie in slot j of the next frame (or in $(j + F)$ mod FN) as well. Once the movie distribution is given, the schedule of transfer (n_{ij}, s_{ij}) in slot j of one frame automatically determines the pair (n_{ij}, s_{ij}) in the next frame, $s_{i(j+F)mod\ FN}$ being the storage node storing the next block of this movie and $n_{i(j+F)mod\ FN} = n_{ij}$. It is observed that the F slots in a frame are not necessarily correlated to each other, but there is a strong correlation between two successive frames of the schedule. It is also observed that the length of the table (FN) is equal to the number of streams that the whole system can support.

Now, the problem can be broken up into two pieces: (a) Can we find a data distribution that, given an assignment of (n_{ij}, s_{ij}) that is source and destination conflict-free, can produce a source and destination conflict-free schedule in the same slot j of the next frame? and (b) Can we find a data distribution that, given an assignment of (n_{ij}, s_{ij}) that is source, destination and network conflict-free, produce a source, destination and network conflict-free schedule in the same slot j of the next frame? The second part of the problem, (b), depends on the network of the multiprocessor 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 multiprocessor network to address the problem (b).

3 Proposed solution

3.1 Part (a)

It has been realized earlier that if all the movies are distributed in the same pattern among the storage nodes, the movie scheduling problem becomes simple[9]. 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 mod 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 cur-

3(a). Movie distribution.

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

3 (b). Schedule for movie E.

Slot 0	1	2	3	4	5	6	7	8	9	10	11
E.2			E.1			E.0			E.3		

3(c). Complete schedule.

Req/Movie.Sender	Slot 0	1	2	3	4	5	6	7	8	9	10	11
0	E.2			E.1			E.0			E.3		
1		C.2			C.0			C.3			C.1	
2		B.1			B.3			B.0			B.2	
3			E.2			E.1			E.0			E.3

Fig. 3. An example movie schedule.

rent 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_{ij}, s_{ij}) in slot j of a frame is followed by a set $(n_{ij}, (s_{ij}+1)\ mod\ N)$ in the same slot j of the next frame. It is clear that if (n_{ij}, s_{ij}) is source and destination conflict-free, $(n_{ij}, (s_{ij}+1)\ mod\ N)$ is also source and destination conflict-free.

This simple approach makes movie distribution and scheduling stright-forward. 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 non-uniform loads on the storage nodes. (iii) as a results of (a), 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 addresses all the above issues (i), (ii) and (iii) and the communication scheduling problem. The proposed solution uses one sequence of storage nodes for storing all the 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 (mod 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_{ij}, s_{ij}) is source and destination conflict-free in slot j of a frame, then the set $(n_{ij}, (s_{ij}+1)\ mod\ N)$ is given to be source and destination conflict-free in slot j of the next frame, whether or not all the movies start at node 0. As mentioned earlier, it is possible to find many such distributions. In the next section, it will be shown that we can pick a sequence that also solves problem (b), i.e., guarantees freedom from conflicts in the network.

3.2 Part (b)

We will use Omega network as an example multiprocessor interconnection network. The solution described is directly applicable to hypercube networks as well. The same technique can be employed to find suitable solution for other networks. 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 of the next frame 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_{ij}, s_{ij}) that are source, destination and network conflict-free, it is automatically guaranteed that the set of nodes $(n_{ij}, s_{((i+1)\ mod\ N)j})$ are also automatically source, destination and network conflict-free.

First, let us review the Omega network. Fig. 2. shows a multiprocessor system with 16 nodes which are interconnected by a 16x16 switch with a single path between any pair of nodes. Fig. 2. is an Omega network constructed out of 4x4 switches. To route a message from a source node whose address is given by $s_0s_1s_2s_3$ to a destination node whose address is given by $d_0d_1d_2d_3$, the following procedure is employed: (a) shift the source address left circular by two bits to produce $s_2s_3s_0s_1$, (b) use the switch in that stage to replace s_0s_1 with d_0d_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 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's address our problem of guaranteeing freedom from network conflicts for a set $(n_{ij}, s_{(i+1)\ mod\ N\ j})$ given that the set (n_{ij}, s_{ij}) is conflict-free. Our result is based on the following theorem of Omega networks.

Theorem: If a set of nodes (n_i, s_i) is network conflict-free, then the set of nodes $(n_i, (s_i + a) mod N)$ is network conflict-free, for any a.

Proof: Refer to [2].

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's 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 of the next frame, which is also conflict-free.

We have shown in this section 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_i, we first determine its starting node s_0 based on the movie distribution. We look at each available slot j (where n_i is free and s_0 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.

4 Other issues

4.1 Choosing a slot size

Ideally, we would like all block transfers to complete within a slot. However, due to variations in delivery time due to variations in load over time, all the block transfers may not finish in the slot they are initiated. One option is to choose the slot to be large enough that it accommodates the maximum delivery time for a block. This approach, however, may not use the network as effectively since it allocates larger amount of time than the average delivery time for a block. If the slot is chosen to be the average delivery time, how do we deal with the transfers that take larger than average delivery delays?

Fig. 4. shows some results from simulation experiments on a 256-node 4-dimensional torus network with 100 MB/s link transfer speeds. These results are only being presented as an example and similar results have to be obtained for the network under consideration. In the simulations, block arrival rates are varied until the deadlines for those block transfers could be met by the network. The figure shows the average time taken for message delivery and the maximum block delivery time and the maximum delay in starting a request at different request arrival times. It is observed that the average message delivery time is nearly constant and varies from 2.8 ms to 2.89 ms over the considered range of arrival times. However, the maximum delay observed by a block transfer goes up from 5.3 ms to 6.6 ms. Similarly, the maximum delay in starting a block transfer goes up from 0 ms to 2.68 ms. Even though the average message completion time didn't vary significantly over the considered range of arrival rates, the maximum delays are observed to have a higher variation. If we were to look at only the average block transfer times, we might have concluded that it is possible to push the system throughput further since the request inter-arrival time of 4 ms is still larger than the average block transfer delay of 2.89 ms. If we were to look at only the maximum block transfer times, we would have concluded that we could not reduce the

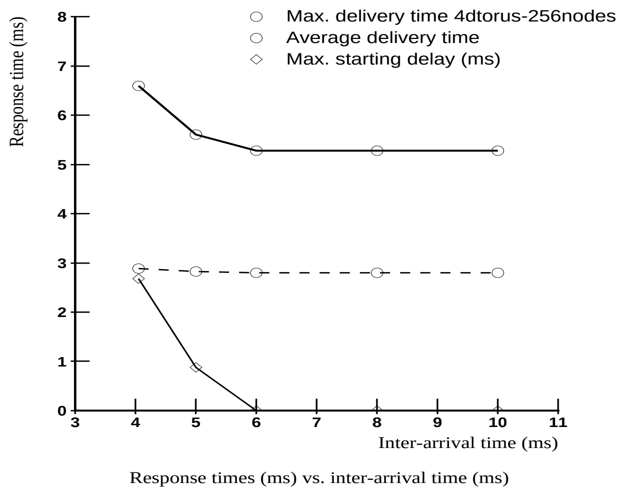

Fig. 4. Observed delays in a 4-dim. 256-node system.

inter-arrival times to below 6 ms. However, the real objective of not missing any deadlines forced us to consider both the average and maximum delays and to choose a different peak operating point of 4 ms of inter-arrival time (slot width).

It is clear from the above description that we need to carry out some experiments in choosing the optimal slot size. Both the average and the maximum delays in transferring a block over the network need to be considered. As mentioned earlier, the slot size is then adjusted such that a frame is an integer multiple of the width of the slot.

4.2 Different stream rates

When the stream rate is different from the basic stream rate, multiple slots are assigned within a frame to that stream to achieve the required stream rate. For example, for realizing a 3Mbits/sec stream rate, 2 slots are assigned to the same stream within a frame. These two slots are scheduled as if they are two independent streams. Only difference is that the network node assigns a larger number of block buffers and handles them differently than with a stream at 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.

4.3 Stream startup latency

It is possible that when a stream A is requested, the next slot where this stream could be started is far away in time resulting in a large startup latency. In such cases, an already scheduled stream B can be moved around within a frame to reduce the requested stream's latency. If stream B is originally scheduled at time T, then stream B can be moved to any free slot within $T + F - 1$ while maintaining guarantees on its deadlines.

4.4 When network nodes and storage nodes are different

It is possible to find mappings of network nodes and storage nodes to the multiprocessor nodes that guarantee freedom from network conflicts. For example, assigning the network nodes the even addresses and the storage nodes the odd addresses in the network, and distributing the movies in round-robin fashion among the storage nodes yields similar guarantees.

4.5 Node failures

In this section, we will show how to tolerate a storage node failure.

Before, we can deal with the subject of scheduling, we need to talk about how the data on the failed data is duplicated elsewhere in the system. There are several ways of handling data protection, RAID, and mirroring being two examples. RAID increases the load on the surviving disks by 100% and this will not be acceptable in a system that has to meet real-time guarantees unless the storage system can operate well below its peak operating point. Mirroring may be preferred because the required bandwidths from the data stored in the system are high enough that the entire storage capacity of a disk drive may not be utilized. The un-utilized capacity can be used for storing a second copy of the data. We will assume that the storage system does mirroring. We will also assume that the mirrored data of a storage node is evenly spread among some set of K, $K < N$, storage nodes.

Let the data on the failed node f_0 be mapped to nodes $m_0, m_1, ..., m_{K-1}$. Before the failure, a stream may request blocks from nodes $0, 1, 2, ..., f_0, ...N - 1$ in a round-robin fashion. The mirrored data of a failed node is distributed among $m_0, m_1, ..., m_{K-1}$ such that the same stream would request blocks in the following order after a failure: $0, 1, 2, ..., m_0, ..., N - 1, 0, 1, 2, ..., m_1, ..., N - 1, ..., 0, 1, 2, ..., m_{K-1}, ..., N - 1, 0, 1, 2, ..., m_0, ..., N - 1$. The blocks that would have been requested from the failed node are requested from the set of mirror nodes of that failed node in a round-robin fashion. With this model, a failure increases the load on the mirrored set of nodes by a factor of (1+1/K) since for every request to the failed node, a node in the set of mirrored nodes observes 1/K requests. This implies that K should be as large as possible to limit the load increases on the mirror nodes.

Scheduling is handled in the following way after a failure. In the schedule table, we allow l slots to be free. When the system has no failures, the system is essentially idle during these l slots. After a failure, we will use these slots to schedule the communication of movie blocks that would have been served by the failed node. A data transfer (n_i, f_0) between a failed node f_0 and a network node n_i is replaced by another transfer of (n_i, m_i) where m_i is the storage node that has the mirror copy of the block that should have been transfered in (n_i, f_0). If we can pack all the scheduled communication with the mirror nodes into the available free slots, with some appropriate buffer management, then we can serve all the streams that we could serve before the failure. Now, let's examine the conditions that will enable us to do this.

Given that the data on the failed node is now supported by K other nodes, the total number of blocks that can be communicated in l slots is given by $K * l$. The failed node could have been busy during $(FN - l)$ slots before the failure. This implies that $Kl \geq FN - l$, or $l \geq FN/(K+1)$ - (1).

It is noted that no network node n_i can require communication from the failed node f_0 in more than $(FN - l)/N$ slots. Under the assumptions of system wide striping, once a stream requests a block from a storage node, it does not request another block from the same storage node for another $N-1$ frames. Since each network node can support at most $(FN - l)/N$ streams before the failure, no network node requires communication from the failed node f_0 in more than $(FN - l)/N$ slots. Since every node is free during the l free slots, the network nodes require that $l \geq (FN - l)/N$, or $l \geq FN/(N+1)$ - (2). The above condition (1) is more stringent than (2).

We can show an upper bound on the number of free slots required. We can show that at least 4 blocks can always be transferred without network conflicts as long as the source and destinations have no conflicts, when the network is built out of 4x4 switches. If a set of four destinations are chosen such that they differ in the most significant 2 bits of the address, it can be shown that as long as the source and destinations are different, the block transfers do not collide in the network. The proof is based on the procedure for switching a block from a source to a destination and if the destinations are so chosen it can be shown that these four transfers use different links in the network. Since at most $FN - l$ blocks need to be transferred during the free slots, $l \leq (FN - l)/4$. This gives $l \leq FN/5$. This implies that if the network nodes requiring communication from the failed node are equally distributed over all the nodes in the system, we can survive a storage node failure with about 20% overhead.

The schedule of block transfers during the free slots is allocated as explained below. A maximal number of block transfers are found that do not have conflicts in the network. This set is assigned one of the free slots. With the remaining set of required block transfers, the above procedure is repeated until all the communication is scheduled. This algorithm is akin to the problem of finding a minimal set of matchings of a graph such that the union of these matchings yields the graph.

Network node failures can be handled in the following way. The movie streams at the failed node are rerouted (redistributed) evenly to the other network nodes in the system. This assumes that the delivery site can be reached through any one of the network nodes. The redistributed streams are scheduled as if the requests for these streams (with a starting point somewhere over the length of the movie, not necessarily at the beginning) are new requests.

If a combo node fails, both the above procedures for handling the failure of a storage node and a network node need to be invoked.

4.6 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 40MB/sec, 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 us, the nodes observe the slot boundaries within $\pm 10\%$. During this time, it is possible that the block transfers observed collisions in the network. But during the rest of the 90% transfer time, 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.

4.7 Other Interconnection Networks

The proposed solution may be employed even when the multiprocessor 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 hypercube type of networks that can emulate an omega network, same data distribution provides similar guarantees as in 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.

For other lower degree networks such as a mesh or a two dimensional torus, it can be shown that similar guarantees cannot be provided. For example, in a two dimensional nxn torus, the average path length of a message is 2* n/4 = n/2. Given that the system has a total of $4 * n^2$ unidirectional links, the average number of transmissions that can be in progress simultaneously is given by $4*n^2/(n/2) = 8*n$, which is less than the number of nodes n^2 in the system for $n > 8$. However, n simultaneous transfers are possible in a 2-dimensional torus when each node sends a message to a node along a ring. If this is a starting position of data transfer in one slot, data transfer in the same slots in the following frames cannot be sustained because of the above limitation on the average number of simultaneous transfers through the network. In such networks, it may be advantageous to limit the data distribution to a part of the system so as to limit the average path length of a transfer and thus increasing the number of sustainable simultaneous transfers.

4.8 Incremental growth

How does the system organization change if we need to add more disks for putting more movies in the system? In our system, all the disks are filled nearly to the same capacity since each movie gets distributed across all the nodes. If more disk capacity is required, we would require that at least one disk be added at each of the nodes. If the system has N nodes, this would require N disks. The newly added disks can be used as a set to distribute movies across all the nodes to obtain similar guarantees for the new movies distributed across these nodes. If the system size N is large, this may pose a problem. In such a case, it is possible to organize the system such that movies are distributed across a smaller set of nodes. For example, the movies can be distributed across the two sets 0, 2, 4, 6 and 1, 3, 5, 7 in an 8-node machine to provide similar guarantees as when the movies are distributed across all the 8 nodes in the system. (This result is again a direct consequence of the above Theorem 1.) In this example, we only need to add 4 new disks for expansion as opposed to adding 8 disks at once earlier. This idea can be generalized to provide a unit of expansion of K disks in an N node system, where K is a factor of N.

This shows that the width of striping has an impact on the system's incremental expansion. The wider the movies are striped across the nodes of the system, the larger the bandwidth to a single movie but also the larger the unit of incremental disk expansion.

When the whole system needs to be expanded including the processors, the switch etc., the issues are the same as in expanding a parallel system.

5 Summary

In this paper, we have shown that movie scheduling, movie distribution and communication scheduling problems in a multiprocessor based video server are closely related. We proposed a simple movie distribution that simplifies movie scheduling and guarantees conflict-free communication over the network. 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. 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 multiprocessor to derive such a sequence that guarantees freedom from communication conflicts if the first block of the movie is scheduled without any communication conflicts.

6 Acknowledgments

Discussions with Roger Haskin and Jim Wyllie have significantly contributed to our understanding of the problem and to some of the ideas in this paper.

References

[1] A. L. Narasimha Reddy and Jim Wyllie. Disk scheduling in a multimedia I/O system. *Proc. of ACM Multimedia Conf.*, Aug. 1992.

[2] Duncan H. Lawrie. Access and alignment of data in an array processor. *IEEE Trans. Comput.*, C-24(12):1145–1155, Dec. 1975.

[3] 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.*, pages 393–400, Aug. 1993.

[4] P. S. Yu, M. S. Chen, and D. D. Kandlur. Grouped sweeping scheduling for dasd-based multimedia storage management. *Multimedia Systems*, 1:99–109, 1993.

[5] 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.

[6] D. Anderson, Y. Osawa, and R. Govindan. A file system for continuous media. *ACM Trans. on Comp. Systems*, pages 311–337, Nov. 1992.

[7] R. Haskin. The shark continuous-media file server. *Proc. of IEEE COMPCON*, Feb. 1993.

[8] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of ACM*, pages 46–61, 1973.

[9] Roger Haskin. Personal communication. *IBM Almaden Res. Center*, 1994.