RC 20020 (4/6/95)
Computer Science/ Mathematics

# IBM Research Report

## Real-Time Issues for Clustered Multimedia Servers

Renu Tewari, Rajat Mukherjee and Daniel M. Dias

IBM Research Division
T.J. Watson Research Center
Yorktown Heights, New York

Harrick M. Vin

Department of Computer Science
University of Texas at Austin
Austin, TX 78712



Research Division
Almaden · T.J. Watson · Tokyo · Zurich

# Real-Time Issues for Clustered Multimedia Servers

Renu Tewari* Rajat Mukherjee Daniel M. Dias Harrick M. Vin

IBM Research Division
T. J. Watson Research Center
Hawthorne, NY 10532

Department of Computer Science
University of Texas at Austin
Austin, TX 78712

**Abstract**

In this paper, we investigate the suitability of clustered architectures (consisting of a set of nodes, each with a local disk array, connected by a high bandwidth switch or network) for designing scalable multimedia servers, that can support thousands of concurrent streams. Specifically, we evaluate the effects of: (i) architectural design of the cluster, (ii) the stripe unit size, and (iii) read-ahead buffering and scheduling, on the real-time performance guarantees provided by the server. We develop an analytical model of clustered multimedia servers, to analyze the effects of these parameters, and then validate it through extensive simulations. We present and discuss our results.

# 1 Introduction

## 1.1 Motivation

Recent advances in computing and communications technologies have given rise to a wide range of applications that involve access to digital audio and video. The realization of such services will require the development of multimedia storage servers that support efficient mechanisms for storing and accessing digital audio and video streams from disk arrays. Since the data transfer rates supported by most of the state-of-the-art disks is substantially larger than the playback rate requirement of an isolated video stream, design of a single-user video server is relatively straightforward. However, the design of scalable multimedia servers that can service a large number of clients simultaneously imposes several research challenges. The design of a scalable multimedia server based on a processor-cluster architecture is the subject matter of this paper.

## 1.2 Relation to Previous Work

The fundamental problem in developing such multimedia servers is that images, audio, video, and other similar forms of data differ from numeric data and text in their characteristics, and hence require totally different techniques for their organization and management. The most critical of these characteristics is that digital audio and video streams consist of a sequence of media quanta, such as video frames or audio samples, which convey meaning only when presented continuously in time (unlike text in which spatial continuity is sufficient). Hence, a multimedia server must ensure that recording and retrieval of media streams to and from disks proceed at their real-time rates.

In the recent past, several research projects have investigated techniques for designing large-scale multimedia servers [2, 4, 5, 11, 17, 24]. Almost all of these efforts, however, have focussed on relatively small scale servers (in which all the disks are assumed to be connected to a single node). The research issues involved in utilizing *clustered architectures* for video servers have not received much attention.

In a concurrent work, Freedman and DeWitt [8] have studied a model of a clustered video server. In their work, all clients have their clocks nearly synchronized and are assumed to possess information about the server architecture, data placement, and scheduling. Whereas the model is appropriate

*also with Department of Computer Science, University of Texas at Austin

for a set of terminals connected to a group of processors, it is unsuited for geographically separated clients that do not possess information about server internals. Also the model for pre-fetch requires an order of magnitude more memory than that assumed in our paper. Their study does not examine the push model, architecture comparisons, random block placement, determination of optimal block sizes and analytical models.

### 1.3 Research Contributions of This Paper

In this paper, we investigate the suitability of clustered architectures (consisting of a set of nodes, each with a local disk array, connected by a high bandwidth switch or network) for designing scalable multimedia servers. In order to ensure isochronous playout, the degree of variability in a multimedia server must be small. Many factors related to the system configuration, stream and disk characteristics, affect this variability. Specifically, we evaluate the effects of: (i) architectural design of the cluster, (ii) the size of the unit of data interleaving (also referred to as *striping unit*), and (iii) read-ahead buffering and scheduling, on the real-time performance guarantees provided by the server. To analyze the effects of these parameters, we develop an analytical model of clustered multimedia servers, and then validate it through extensive simulations. We present and analyze our results.

The remainder of the paper is organized as follows: In Section 2, we describe two possible cluster architectures for video servers, discuss the placement strategies and other assumptions made about the server design. Section 3 describes an analytical model of the system, which is validated and evaluated by extensive simulations in Section 4. Finally, Section 5 summarizes our results and outlines directions for future work.

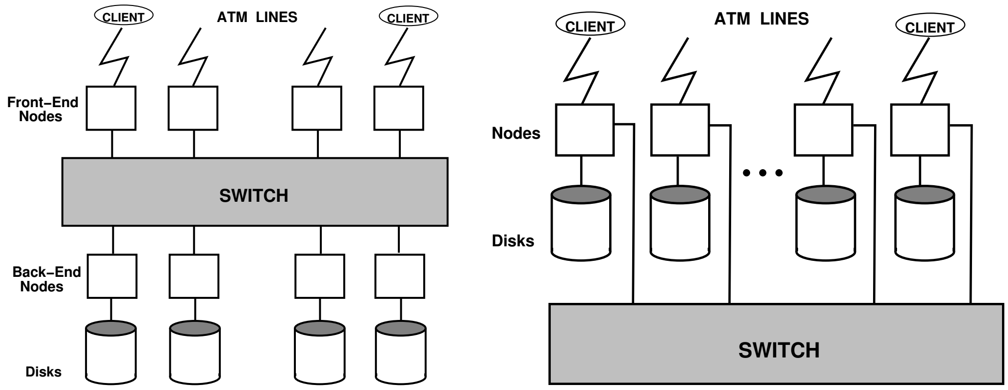
## 2 Clustered Multimedia Servers

Video servers are designed to support a large number of video streams that have continuous playout requirements. Since a single-node system with a disk array cannot sustain the bandwidth required for a scalable system with large capacity, clustered architectures have been proposed [10]. A clustered architecture consists of a group of nodes connected by a switch (interconnection network). Each node has a local disk array attached to it. The nodes of a cluster can be divided into two logical categories: (i) front-end (delivery) nodes and (ii) back-end (storage) nodes. Front-end nodes admit stream requests from the external (e.g., ATM) network based on some pre-defined or dynamic admission control strategy. Requests made by a stream at a front-end node traverse the switch to the back-end node which has the required data block, based on the data placement strategies. We assume an underlying software layer (e.g., virtual shared disk) makes the interconnection architecture transparent to the nodes [1]. The back-end node issues the request for a block to the disk which contains the given block. The block is then sent back from the back-end node to the front-end node through the switch and stored in the stream read-ahead buffer at the front-end. The blocks from the buffer are transmitted over the external network at regular intervals, based on the playout rate of the stream.

Two possible configurations for a clustered architecture are: (i) two-tier and (ii) flat (see Figures 1(a) and 1(b), respectively). In a two-tier architecture, the logical front-end and back-end nodes are mapped to different physical nodes of the cluster and are distinct. The set of front-end (delivery) nodes are connected to back-end (storage) nodes via a switch. The front-end nodes do not communicate with each other. Data is stored on disks attached to the back-end nodes. The front-end nodes are connected to the clients requesting video data via a high-speed external network [23].

In a flat architecture, each physical node can serve as a logical front-end or back-end. All nodes are identical, performing both storage and delivery functions. The nodes are connected to each other via the switch and also connect to the external network. The node that initiates the request for a stream is called the front-end node while the one that services the request is the back-end node.

Regardless of the architecture, each video stream is divided into blocks, which are then distributed among the disks in the system. Blocks of a video stream may either span all the disks in the system (referred to as *wide striping*) or may be confined to a smaller subset of disks (referred to as *short striping*). Whereas wide striping implicitly achieves higher disk-arm bandwidth and load balancing [2, 10], achieving the same objective using short striping will require replication and distribution of




(a) Two-Tier (b) Flat

Figure 1: Server Architectures

video streams. Unlike short striping, however, in a server that employs wide striping, a single disk or node failure affects all the streams being accessed [22].

In either scheme, successive blocks of a video stream may be allocated to disks either using a *round-robin* or *random* placement algorithm. As per the round-robin placement, successive blocks of a video stream are placed on consecutive disks (see Figure 2(a)). With random placement, successive blocks are placed on disks using a random permutation, with no two consecutive blocks on the same disk. In both schemes, the first block of a stream could be allocated on any disk with equal probability. A key advantage of the random placement policy is that it adapts to incremental growth. If disks are added frequently to the system, then it is much easier to redistribute the data–with the addition of a single disk, only $1/D^{th}$ of the data per disk needs copying, where D is number of disks. However, the fundamental problem with random placement is that: (i) even with all the streams playing out at a constant rate, it is statistically possible that a large number of streams may simultaneously access the same disk or node, and (ii) it requires more meta-data. In round-robin placement, on the other hand, the streams access the disks and nodes in a deterministic fashion. However, this could cause a *convoy effect* if startup is not controlled, i.e., if there is a congestion on a disk, it travels to the next disk for the next request and so on. Although this limitation can be addressed by controlling the startup of each stream, such policies may incur higher initiation latencies. A detailed comparison of these two policies can be found in [21].

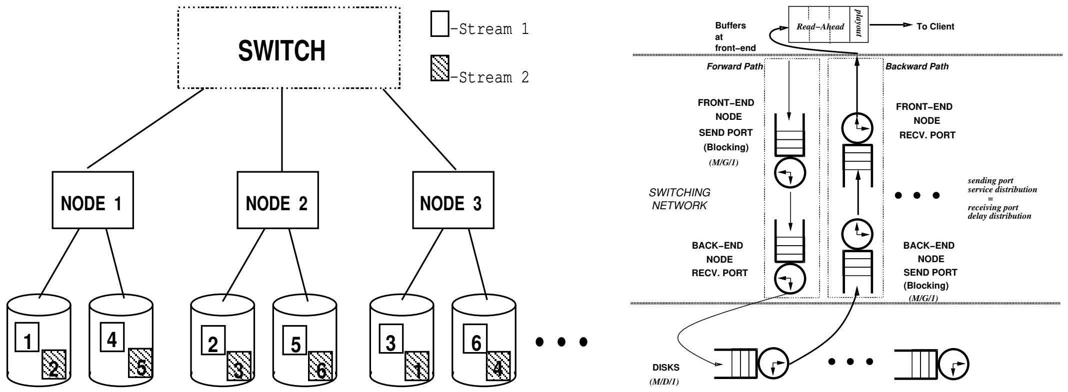



Figure 2: (a) Sequential Block Placement (b) The Analytical Model Queues

For the purpose of the analysis presented in this paper, we will assume that the server employs wide striping and random placement policies. Moreover, we will assume that the server operates in *push* mode; i.e., it provides data at a fixed rate (assuming constant bit rate encoding) after the connection is established for a stream, with limited client control. The admission control strategy is limited to ensuring that all resources required by a stream do not exceed their operational utilization. A block request that is not serviced by its deadline is treated as *lost*, and results in a *glitch* at the client end.

The quality of service (QoS) criteria assumed, is to minimize the loss rate per stream.

# 3 Analytical Model

We describe an analytical model of the system that can be used to compare the two architecture options, predict an optimal block size and show the effect of read-ahead buffering, on the real-time performance. Since long simulations are needed to compare the performance of various factors, the analytical model can be used to quickly prune the space explored by simulation and lend credence to moderately long simulation results.

## 3.1 Modeling the Server

The server can be modeled as three logical queues, i) the forward path queues of the switching network, ii) the disk queues and iii) the backward path queues of the switch, as shown in Figure 2(b). The external network is not considered a part of the server and is not modeled. The model assumes a random placement of blocks across disks in the system, but will hold for a round-robin placement with random start locations and continuous stream arrivals. The processing delay at each node is assumed to be small and a mean value is used in the model. The number of nodes in the system is $N$, and the number of disks is, $D$. The total number of streams in the system is, $S$. Each stream is periodic, with a uniform request arrival rate of, $\lambda$ blocks/sec. The time required to transfer a large block across the switch is, $1/\mu_{block}$, and for a small request it is, $1/\mu_{request}$. This includes a connection set up time and a transfer time based on block size. For any data transfer across the switch, the message is queued at the source sending port and the destination receive port. After the sending port is acquired, it is *blocked* until the receiving port is also acquired, and the data transfer is completed. (All queuing model results required are given in the Appendix, for reference purposes.)

### 3.1.1 The Disk Queue

We assume that the disk subsystem consists of $D$ identical disks attached to $N$ identical nodes (each node is connected to $D/N$ disks directly[1].) Each disk in the disk subsystem can be modeled as a single server queue. Under the assumption that streams are independent, have random startup times and positions, and the number of streams is large, we can approximate the arrival process for read requests seen by the disk subsystem to be Poisson, with mean $\lambda S$. Since a disk is randomly selected by a stream the arrival process at a particular disk is Poisson with a mean of $\lambda_d = \lambda S/D$. The service time at the disk is assumed to be deterministic and equals the mean service time $1/\mu_d$. This assumption can be made because the variation in disk service time due to seek delays is relatively small compared to the total service time, when the block size is large[2]. Each disk thus behaves like an M/D/1 queue. From the results for an M/D/1 queue [9] we know the queue length distribution, $p_n$, which is the probability that there are $n$ requests at a disk, from which the delay distribution, $W_d(t)$ (probability that a request has a disk delay equal to t) can be derived.

### 3.1.2 The Forward Path

The switching network consists of $N$ nodes connected by a cross-point switch. Each node has two ports connected to the switch; a sending port and a receiving port. The forward path consists of a small read request sent from a logical front-end node's sending port to a logical back-end node's receiving port.

The forward path is modeled as two queues, a blocking sending queue and a non-blocking receiving queue. Since the service time at the sending queue depends on the queuing delays at the receiving end, we use the *delay distribution* of the receiving queue to determine the *service time distribution* of the sending queue. Each back-end receiving port is modeled as an M/D/1 queue with one server. The service time at the receiving port is, $1/\mu_{request}$. The arrival rate at the back-end receiving queue is, $\lambda_r^{be} = \lambda S/(N_{be})$, where $N_{be}$ is the number of back-end nodes. The queue length distribution, $p_i$,

[1] The model can be easily extended to a non-uniform configuration with additional book-keeping.

[2] Further we are only interested in the tail of the delay distribution to determine the probability of requests experiencing delay more than a given bound. Since the delay experienced by a request has a smaller variance for a longer queue, an M/D/1 model is reasonable when the focus is on the probability of a long queue.

and the mean delay for an M/D/1 queue is known [9]. From the queue length distribution, $p_i$, we get the delay distribution, $W_r^{be}(t)$, at the back-end receiving queue for discrete values. The front-end sending port is modeled as an M/G/1 queue. The service time distribution at the front-end sending queue is obtained from, $W_r^{be}(t)$. The arrival rate, $\lambda_s^{fe}$, at the front-end sending queue is, $\lambda S/(N_{fe})$, where $N_{fe}$ is the number of front-end nodes. The mean service time at the front-end sending queue, $1/\mu_s^{fe} = 1/\mu_{request} + \frac{\rho_r^{be}}{2(1-\rho_r^{be})\mu_{request}}$, and equals the mean delay at the back-end receiving queue. The delay distribution at the front-end sending port, $W_s^{fe}(t)$, is derived from the known result for an M/G/1 queue [9].

### 3.1.3 The Backward Path

The backward path consists of the transfer of a large data block from the logical back-end node's sending port to the requesting front-end node's receiving port. The backward path can be modeled in a similar fashion as the forward path, with the roles of front-end and back-end nodes reversed. Also, the transfer time for a large block is much larger than that for the small request in the forward path. Each front-end receiving port is modeled as an M/D/1 queue and the back-end sending port is modeled as an M/G/1 queue. The delay distribution at the back-end sending port, $W_s^{be}(t)$, is derived similar to the forward path.

### 3.1.4 Mapping Different Architectures

**Two-tier Architecture**– The server model can be directly applied to the two-tier architecture. The logical front-end and back-end nodes can be mapped to physical front-end and back-end nodes. For simplicity, we assume in the analysis that the number of front-end and back-end nodes are equal (i.e., there are $N/2$ front-ends and back-ends). The forward and backward paths can be mapped to two separate links on the switch, and are distinct. For the two-tier architecture, the utilization at the back-end receiving port, $\rho_r^{be} = \lambda_r^{be}/\mu_{request}$, where $\lambda_r^{be}$ is the arrival rate at the port. Assuming that the arrival rates at all ports are identical, and using the queuing assumptions of the ports, we can estimate the utilization at the front-end sending port, $\rho_s^{fe} = \rho_r^{be} + (\rho_r^{be})^2/2(1-\rho_r^{be})$. The utilization at the front-end receiving port, $\rho_r^{fe} = \lambda_r^{fe}/\mu_{block}$. The corresponding utilization at the back-end sending port, $\rho_s^{be} = \rho_r^{fe} + (\rho_r^{fe})^2/2(1-\rho_r^{fe})$.

**Flat Architecture**– In the flat architecture each node functions as a logical front-end or back-end. The links in the network are common for both forward and backward paths. Since there are two types of messages at every port, the arrival rate at any port (sending or receiving) is, $\frac{2\lambda S}{N}(1-1/N)$, given that $1/N^{th}$ of the requests per stream are serviced locally and do not traverse the switch ports. The receiving port model is modified to be an M/G/1 queue, with two service times, both equally likely. The mean service time, $1/\mu_r$, equals the average time to send a message (short request and large block) across the switch. The sending port is modeled as an M/G/1 queue with the service time distribution equal to the delay distribution of the receiving port, as described in the server model. In a flat architecture the utilization at the receiving port of all nodes, $\rho_r = \lambda_r/\mu_r$. Using the switch model for a flat architecture and assuming that the arrival rates at all ports are identical, we can estimate the utilization at the sending ports, $\rho_s = \rho_r + (\rho_r^2 + \lambda_r\sigma_r^2)/2(1-\rho_r)$, where $\sigma_r^2$, is the variance in the service time at the receiving port.

## 3.2 Computing Loss Probability

The QoS criterion assumed to evaluate the server design, is the loss probability of the streams. The *loss probability* is defined as the probability that a block requested by a stream is not available for playout by its deadline. A block could be delayed due to congestion at the switch (forward and backward paths) or at the disk queues. A very accurate analytical model to predict the overall loss probability would require the convolution of the delay distribution at the switch with that at the disk, to compute the overall delay distribution. The tail of this distribution gives the probability that the delay exceeds a given deadline bound, which is the loss probability for a stream. A simpler method can be used if we know that the disk delay, $1/\mu_d$, is the dominant part of the overall delay. We reduce the total delay bound (deadline), by the mean delay at the switch, $\delta_{net}$, and the mean processing delay, $\delta_p$, to get the delay bound at the disk. The mean processing delay of a block, $\delta_p$, can be computed from the default parameters of Table 1. $\delta_{net}$, is the sum of the mean delay in the forward and backward paths of the switch and is given by,

$$\begin{aligned}\delta_{net} &= 1/\mu_{request} + \frac{\rho_r^{be}}{2(1-\rho_r^{be})\mu_{request}} + \frac{\lambda_s^{fe}}{2(1-\rho_s^{fe})}[1/\mu_{request}^2 + \frac{\rho_r^{be}}{3(1-\rho_r^{be})\mu_{request}^2} + \frac{(\rho_r^{be})^2}{4(1-\rho_r^{be})^2\mu_{request}^2}] + \\ & 1/\mu_{block} + \frac{\rho_r^{fe}}{2(1-\rho_r^{fe})\mu_{block}} + \frac{\lambda_s^{be}}{2(1-\rho_s^{be})}[1/\mu_{block}^2 + \frac{\rho_r^{fe}}{3(1-\rho_r^{fe})\mu_{block}^2} + \frac{(\rho_r^{fe})^2}{4(1-\rho_r^{fe})^2\mu_{block}^2}].\end{aligned}$$

The $\rho$ values are the utilization of the sending and receiving ports of the front-end and back-end nodes. Let us assume that the deadline of a request[3] is the time period between requests of a stream, $1/\lambda$. The maximum delay, $D_{max}$, that a block can encounter at the disk and still meets its deadline is, $(1/\lambda) - \delta_{net} - \delta_p$. The loss probability is then computed from the queue length distribution, $p_i$, of the disk queue [9], using Equation 1.

$$P[loss] = P[delay > D_{max}] = 1 - \sum_{i=0}^{i/\mu_d \leq D_{max}} p_i \quad (1)$$

# 4 Analysis and Evaluation

The analytical model cannot capture all the effects of buffering, scheduling, load imbalances, disk characteristics etc., and hence, a simulation model is required for completeness and validation.

## 4.1 Simulation Model

In the simulation model, all resources (nodes, ports, disks) are modeled as single server queues. The disk service time includes the seek time (square root function of the arm distance), the rotational latency and the block transfer time. The disks are assumed to be re-calibrated at regular intervals, blocking all requests during that period. We assume that the disks are zoned with different transfer rates per zone [16]. The service time for sending a request across the switch includes a fixed connection setup time and a transfer time proportional to the size of the block. The switching network assumed for concreteness is a fiber optic cross-point switch (FCS) with upto 64 send/receive ports, with a default port to port bandwidth of 20 MB/sec. The parameters selected for the switch are similar to a commercially available FCS product [12]. In the model, *the number of nodes* (front-end and back-end) is determined based on: i) the desired system capacity (maximum number of streams in the system), ii) the desired mean utilization of the nodes, iii) the software overheads [4] for the nodes[3], iv) the mean playout rate of the streams and v) the processor speed. The *number of disks* depends on the total amount of storage required and the necessary disk-arm bandwidth. Assuming that storage is not a constraint, the number of disks in the system is calculated from, i) the effective disk bandwidth, ii) the desired disk-arm utilization, iii) the playout rate of the streams, iv) the data block size and v) the system capacity. The disk bandwidth is in turn based on the mean seek delay, the rotational latency and the transfer time of a data block. The number of disks computed may not be a multiple of the number of the back-end nodes, hence some of the back-ends may have more disks, leading to load imbalance. More details of the configuration can be found in [21].

A triple buffer (default) is associated with each stream at the corresponding front-end node. One buffer (playout buffer) is reserved for stream playout, and the remaining two are used for read-ahead. The size of a single buffer equals the block size on disk. We assume the deadline of a read request depends on the playout rate of the stream and the buffer size.

The arrival pattern of the streams is modeled as a Poisson process. The mean interarrival time $1/\lambda_s$ between streams is computed using Little's Law [13] as $\lambda_s = \frac{TOTAL\_STREAMS}{(STREAM\_DURATION)}$. If streams are very short (less than 90 seconds), they may not span all disks in the system, causing load imbalances. In order to include the effects of short clips, we generate stream durations from an exponential

[3] The deadline will also depend on the read-ahead buffer size. If blocks are pre-fetched before playout they will have a later deadline. The model can be easily generalized to set the deadline to any desired value based on the system requirements.

[4] Software overheads at the back-end node include overhead for communications over the switch and disk access. The front-end node overhead includes the switch communication, file access and external network communications overhead.

| Description | Default |
|---|---|
| Front-End Processing Delay | 8.192 msec. |
| Back-End Processing Delay | 8.96 msec. |
| Block Size | 256 KB |
| Number of Streams | 500 |
| Front-End, Back-End Utilization | 70% |
| Number of Processors (70% util.) | 35 |
| Stream Data Rate | 0.5 MB/sec |
| Processor Speed | 100 MHz |
| Switch Bandwidth | 20MB/sec |
| Switch setup overhead | 2 msec. |
| Front-end s/w overhd. | 6.2 cyc./byte |
| Back-end s/w overhd. | 3.5 cyc./byte |
| Mean Seek Time | 10 msec. |
| Number of Zones | 10 |
| Disk RPM | 7200 |
| Zoned Disk data transfer rate | 6–12 MB/sec |
| Number of disks (80% util.) | 52 |
| Effective Disk Bandwidth | 6.04 MB/sec |
| Calibration delay/ 2mins | 200 msec |
| Scheduling Policy | FCFS |
| Number of read-ahead buffers | 2 |

Table 1: **Default Parameters Used**

distribution, with a mean of 500 seconds. Also, streams pause and resume at random intervals (exponential with mean 10 min.) to cause the read-ahead buffers to get flushed periodically. The streams start at randomly selected disks, with empty read-ahead buffers. The model is implemented using CSIM [19], and Table 1 gives the default parameter values[5] used in the simulations.

## 4.2 Validation of the Models

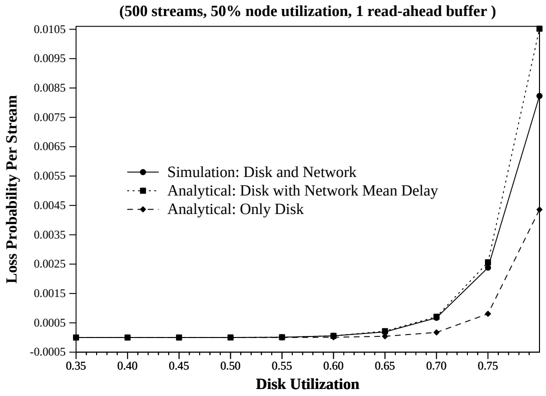
We compare the results of the analytical model with that of the simulation model, to predict the overall loss probability of streams. Figure 3(a) shows the loss probability computed by the analytical method described in Section 3.2, with the one generated by the simulation, for a two-tier architecture. In the comparison, we assume, 50% node utilization, FCFS scheduling, one effective read-ahead buffer, random placement of blocks and a 256KB block size. The analytical model using only the disk queue predicts a lower loss probability than the actual simulation. The modified disk queue model with the mean switching network and processing delays predicts a slightly higher loss probability than the simulation and is fairly close.

## 4.3 Predictions of the Models

Having validated the analytical model by simulation, we shall use the analytical model to understand the effects of different factors, on the system behavior, and use the simulation model to quantify the results based on loss rate per stream[6].

[5] The thresholds for node (front-end and back-end) utilization was based on experiments to determine the operational utilization that is used for admission control. Any stream that selects a front-end that is utilized 5% above the threshold, is rejected. The reneging (rejection) probability could be decreased if streams could be diverted to other front-end nodes for better load balancing.

[6] The loss rate per hour per stream can be used to calculate the loss probability, given by, $lossrate/(3600 * \lambda)$.



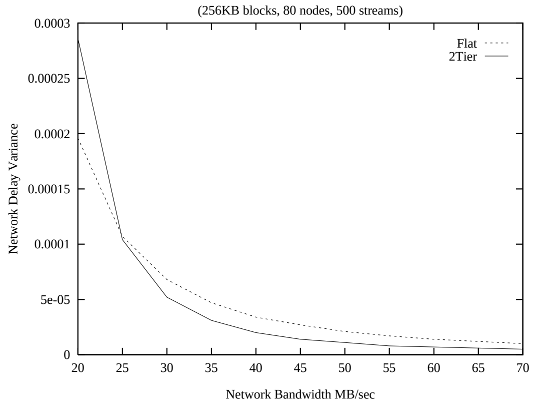




Figure 3: (a) Loss Probability- Analytical Vs. Experimental (b) Analytical Model–Switch Delay Variance

#### 4.3.1 Comparison of Architectures

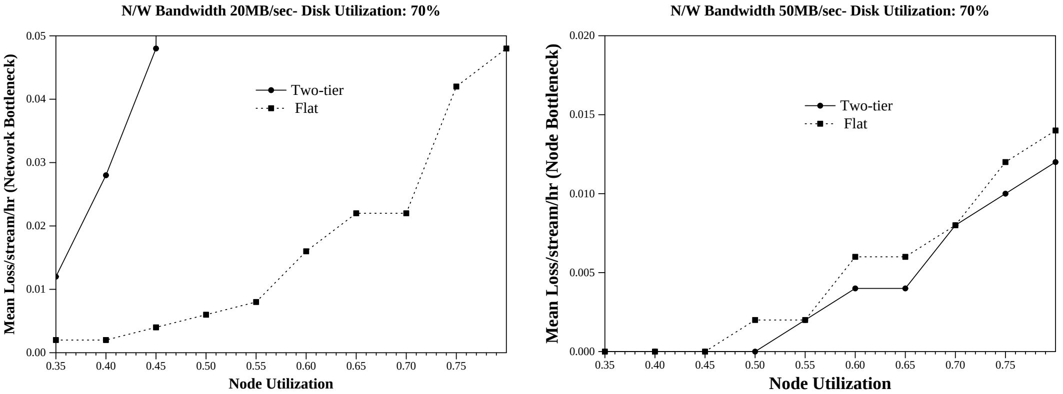
The analytical and simulation models of the network can be used to provide an insight into the performance of the flat and two-tier architectures. It seems intuitive that the port utilization in a two-tier architecture in the backward path (large block) will be much greater than the forward path (small request). On the other hand, in a flat architecture, ports are uniformly utilized, as both types of messages are handled at each port. However, the flat architecture will have more variance in the switching delay, due to two types of messages. We can compute the port utilization for the two architectures using the results in Section 3.1.4. Table 2 shows the analytically determined port utilization for the different architectures, assuming a 256KB size block and a switch bandwidth of 20MB/sec. It can be seen that the port utilization of the two-tier architecture are highly skewed, with the back-end sending port saturating for a 40 node system, while the corresponding flat architecture ports are less than 50% utilized. Thus, the two-tier architecture cannot operate with small number of nodes (high node utilization), leading to poor performance, when the switch bandwidth is small.

| Number of Nodes | Two-Tier | | | | Flat | |
|---|---|---|---|---|---|---|
| | Front-End | | Back-End | | Sending | Receiving |
| | Sending | Receiving | Sending | Receiving | | |
| 56 | 7.2% | 42% | 57.2% | 7% | 29.1% | 24.5% |
| 50 | 8.3 | 48 | 70 | 8 | 34.3 | 28 |
| 44 | 9.4 | 54 | 85 | 9 | 40 | 31.5 |
| 40 | 10.5 | 60 | 100.5 | 10 | 46 | 35 |
| 36 | 11.6 | 66 | 130 | 11 | 52.6 | 38.5 |
| 32 | 12.8 | 72 | 164 | 12 | 59.8 | 42 |

Table 2: **Port Utilization- Analytical Results** (500 streams, 0.5 MB/sec.)

The analytical model can also be used to compute the variance in the round-trip switching delay, for the two-tier and flat architectures. The variance and other higher moments required for the computation can be derived from the Laplace transform of the delay distribution of the sending port queue [9] [20] and is given in [21]. The flat architecture has a much larger delay variance in the forward path when compared with that of the two-tier architecture as there are only small messages in the two-tier forward path. In the backward path, however, the flat architecture has a slightly smaller variance than the two-tier because of the much lower port utilization. The overall variance is still higher for the flat architecture. Figure 3(b) shows the variance for the two architectures with varying switch bandwidth. For lower switch bandwidth, the port utilization dominates, leading to higher variance for the two-tier architecture. The overall performance of the two architectures depends on the network and node parameters.

We use the simulation model to quantify the analytical predictions by observing the loss rate per




(a) Switching Network Bottleneck (b) Node Bottleneck

Figure 4: Flat Vs. Two-Tier-Simulation Results

stream. The flat architecture[7] has larger delay variance but lower port utilization, compared to the two-tier. The choice of architecture depends on which among the nodes and network, is the bottleneck. If the switching network is the bottleneck, then the flat architecture has better performance, because the sending ports in the two-tier saturate, even when the number of nodes in the system is large (low node utilization). Figure 4(a) shows the loss rates increasing rapidly due to saturating ports, when the node utilization is just 50%. When the switching network is not a bottleneck (high bandwidth), the port utilization is small and the delay variance dominates. The two-tier has smaller variance, leading to lower loss rate as shown in Figure 4(b). The flat architecture is preferred due to the more optimal port utilization. When the network is not a bottleneck the two-tier architecture may be desirable for practical considerations as it is simple to configure and manage. If the interconnect is a LAN or a token-ring the two-tier and flat architecture will have similar networking performance.

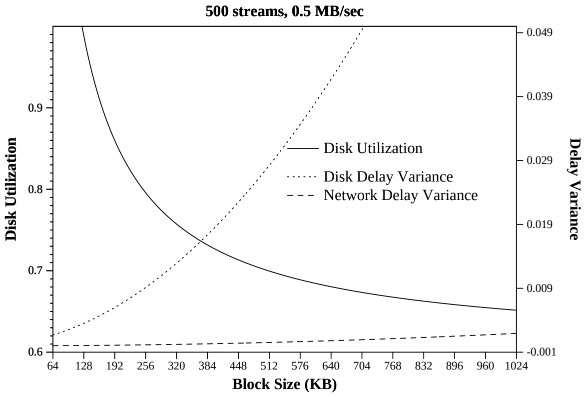
### 4.3.2 Optimal Block Size Determination

The analytical model can be used to determine the optimal block size on disk, that will minimize the probability of a request missing its deadline. As the block size increases, the thruput of the disk (or switch) increases due to smaller overhead of disk read latency (or switch startup delays). Also, the number of blocks requested per second decreases, for the same playout rate of the streams. All this favors a larger block size. However, when the block size is large, the variance in the response times at the disk and network increases. Even a small queue length results in long delays. This results in a smaller block size being favored. Figure 5(a) shows the two phenomena for different block sizes. The tradeoff between disk thruput and delay variance results in an optimal block size. We assume that the blocks on disks are of fixed size[8].

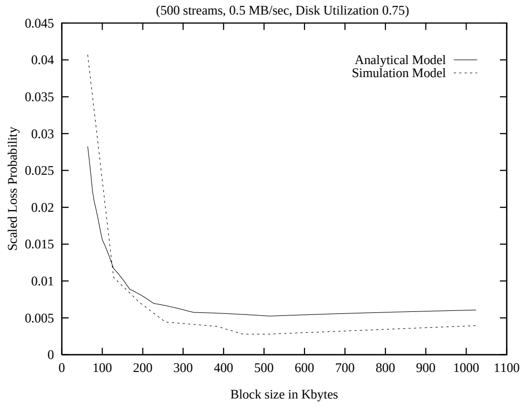
We can compute the optimal block size from the delay distribution at the disk. For simplicity, we assume that the deadline for a read request is the time period between requests of a stream. For a request rate of, $\lambda$ bytes/sec, and a block size of, $b$ bytes, the deadline for a request is, $b/\lambda$. Since the delay and variance in the network is not significant compared to that at the disk, we assume that each request for a block, encounters the mean delay in the network, $\delta_{net}(b)$, and the mean processing delay at the nodes, $\delta_p(b)$, as described in Section 3.2. To compute $\delta_{net}(b)$, we use the mean time to transfer a block across the network, $1/\mu_{block} = \mathit{startup\ delay} + b/(\mathit{network\ bandwidth})$. The maximum delay the request can encounter at the disk, $D_{max}$, is given by, $b/\lambda - \delta_{net}(b) - \delta_p(b)$. The service time at the disk, $1/\mu_d = \mathit{readlatency} + b/(\mathit{disk\ bandwidth})$. The loss probability can be then computed as in Section 3.2, using Equation 1.
For different values of block sizes, the loss probability is calculated for a system of 500 streams, 75% disk utilization, 60 nodes and one read-ahead buffer. The loss probability is scaled based on the

[7] In a flat architecture the data residing on the local disks of a node can be accessed without traversing the switch. Thus 1/Nth of the data per stream can be accessed locally.

[8] The blocks on disk could be of fixed time length and variable sizes [4], especially with variable bit rate encoding. This is discussed in Section 5.




(a) Block size Vs. Disk Performance and Delay Variance




(b) Optimal Block Size Determination

Figure 5: Effect of Block Size

block size as the loss of a larger block involves losing more data. The block size that generates the lowest scaled loss probability is the optimal block size. Figure 5(b) shows the loss probability using the analytical and simulation models using an identical system configuration. For a small block size (64KB) the analytical model has a large deviation from the simulation results, as it does not include seek variations, which are a significant part of the disk delay. The optimal block size is in the range of 250-400KB. A default block size of 256KB is used in the simulations for implementation specific reasons[9].

#### 4.3.3 Read-ahead Buffering and Scheduling Effects

Read ahead buffering is a simple approach to increase the deadline bound of a request, leading to a decrease in the loss probability, while adding some extra cost[10]. The blocks required later for playout are pre-fetched and stored in the read-ahead buffers at the front-end node. The deadline of a block request is the time duration available, before the block is played out. In the analytical model, for a stream request rate of $1/\lambda$ blocks/sec, and $k$ read-ahead buffers, the deadline of a request increases to, $k/\lambda$. The maximum delay bound, $D^k_{max}$, that a request can encounter at the disk (assuming mean delays at the switch and processors) and still meet its deadline is, $(k/\lambda) - \delta_{net} - \delta_p$. The loss probability with $k$ read-ahead buffers is identical to Equation 1, with the new delay bound. This loss probability does not capture congestion at startup to fill multiple buffers. It is valid only when the playout of a stream begins after all read-ahead buffers are full.

If a fixed size buffer space[11] is available per stream, there is a tradeoff between the disk performance due to block size and the lower loss probability due to multiple read-ahead buffers. Table 3, derived from the analytical model shows that a single read-ahead buffer with larger block sizes, performs worse than a smaller block with multiple buffers. However, when the buffer space available is larger, the gain with increasing read-ahead buffering dwindles and the disk performance favors larger blocks sizes.

The effects of buffering and startup with different scheduling policies are analyzed using only the simulation model, since the analytical model assumes a FCFS (First Come First Served) queue. With EDF (Earliest Deadline First [14]), each request for a block is tagged with a deadline, which is a function of the read-ahead buffer size and the playout rate. All streams are assumed to use a global clock for deadlines within the cluster. Figure 6(a) depicts the block loss rate with the FCFS and EDF scheduling strategy at the disk for three different buffer sizes. With an EDF policy, the observed losses are negligible[12]with two read-ahead buffers and drop to 0 with 3 read-ahead buffers. An interesting observation is that, with the FCFS disk scheduling policy, the losses actually increase when the read-ahead buffer size is increased from 2 to 3. This happens because in the simulation, at

[9] 256KB is a multiple of 64KB which is the maximum packet size allowed by IP. Special purpose communication protocols can be used to eliminate this constraint

[10] A memory cost of \$30/MB per stream

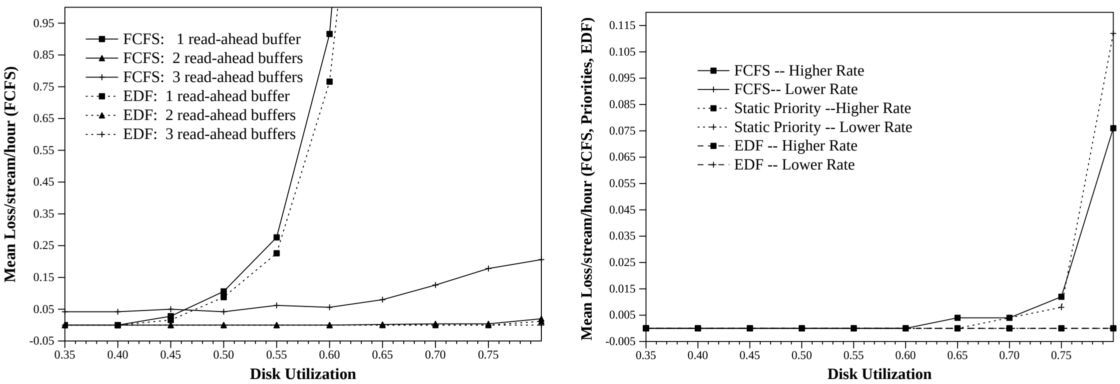
[11] playout + read-ahead buffers

[12] (0.002 blocks per stream, per hour at 80% disk utilization)

| Total Size of Memory per stream (KB) | 4 Buffers | | 3 Buffers | | 2 Buffers | |
|---|---|---|---|---|---|---|
| | Block size (KB) | Probability of Loss (Scaled) | Block size (KB) | Probability of Loss(Scaled) | Block size (KB) | Probability of Loss (Scaled) |
| 256 | 64 | 0.000014 | 85 | 0.000237 | 128 | 0.012678 |
| 512 | 128 | 0.0 | 170 | 0.000027 | 256 | 0.007161 |
| 768 | 192 | 0.0 | 256 | 0.000011 | 384 | 0.006105 |
| 1024 | 256 | 0.0 | 341 | 0.0 | 512 | 0.005899 |
| 2048 | 512 | 0.0 | 682 | 0.0 | 1024 | 0.006873 |

Table 3: **Block size and Read-ahead Buffering-** (500 streams, 0.5 MB/sec.)

stream startup, multiple requests are made to fill the buffers. FCFS does not differentiate between read-ahead requests leading to startup congestion. This can be avoided by delaying playout of a stream till the buffers are full or by spacing the read-ahead requests. Since the EDF scheme forces larger deadlines on the read-ahead requests, this phenomenon does not occur. It is obvious that EDF will perform better than FCFS for real-time requests. However, implementing EDF requires significant modifications to existing I/O subsystem software [1] that uses standard UNIX interfaces[13]. Thus we need to quantify the performance benefits of EDF, to justify the effort. If the arrival time of a request is proportional to its deadline, FCFS and EDF are nearly equivalent; however due to a clustered architecture, requests from different nodes may arrive out of order at the disk. This effect is even more pronounced when the play-out rates of streams are different.




(a) Different Buffer Sizes (b) Different Playout Rates

Figure 6: Scheduling Effects–Simulation Results

We examine 3 different scheduling policies at the disk: FCFS, Static Priority Scheduling and EDF, with streams having 2 playout rates (typical data rates of MPEG-2, 0.5 MB/sec and MPEG-1, 0.2 MB/sec). The static priority schedule assigns a higher priority to requests of the higher rate. Figure 6(b) shows that with FCFS, streams with the higher request rate (MPEG-2) have higher losses, while with static priority, streams with the lower rate have losses. The FCFS policy cannot distinguish between requests of different rates, increasing the losses of the faster rate, while the static priority causes starvation of the lower rate. The EDF strategy, using real-time deadlines, is a simple solution that reduces the loss rate for both playout rates.

The initial request of a stream at startup (i.e., the first block read) has no deadline, but requires its delay to be minimized for a quick startup. Subsequent requests have deadlines governed by the playout rate. The EDF algorithm is modified to handle non real-time requests. Scan-EDF [16] could be used to improve the performance of EDF, by reducing seeks. However, we found [21] that only a marginal improvement is achieved for small block sizes of 64KB. With larger block sizes there was negligible improvement. This is because the requests are not grouped together in rounds and hence the probability of multiple requests having the same deadline is rare. Also, with larger block sizes, the gain in optimizing seeks diminishes.

[13] Existing virtual shared disk software has to be extended to tag deadlines with each request

## 5 Conclusions and Future Work

We have examined issues related to the real-time constraints of concurrent video streams in a clustered video server. Specifically, we examined different architectures, optimal block size determination, buffering and scheduling policies with multiple data rates.

We described an analytical model for the disk and switch subsystem. The model was validated using simulations. We examined the performance of different cluster configurations and concluded that flat architectures are practical for low bandwidth switches. We computed the optimal block size by the analytical model and showed the tradeoff between disk performance and loss rates. To improve the performance of random placement, we examined the effect of read-ahead buffering and disk scheduling policies, on the real-time behavior of video streams. From our simulation studies, we concluded that it is essential to use a deadline based disk scheduling policy to meet the real-time constraints. Thus it is necessary, from an implementation point of view, to modify existing I/O interfaces. FCFS and static priority schemes are not fair to streams of different classes. Elevator and related scheduling algorithms like Scan-EDF were not discussed in detail, as the simulations showed only a marginal gain for large block sizes. Read-ahead buffers need to be used to smooth out variations in response time, due to contention and queuing for shared resources. We found that double buffering is not adequate, even with EDF and large block sizes, to mask the jitter in response times. Increased buffering with FCFS disk scheduling could actually increase the loss rate, due to congestion at stream startup. Large amount of buffering may not be cost effective as shown in [6]. The real-time performance of the streams could be affected by many other factors. In modern disk with uniform recording density, the cylinders are often divided into multiple zones with varying data transfer rates [18]. We are currently investigating optimal placement of variable sized blocks on zoned disks for video server applications. We study the effect of thermal re-calibration of disks using simulations and analysis on the real-time performance of streams in [21]. The server can also be enhanced to efficiently offer full VCR functionality. Stream performance can also be improved by using replication schemes which are discussed in [7, 15, 6]. To guarantee high-availability for clustered architectures with wide striping we are currently exploring mechanisms [22] that can be used to mask single node and disk failures without adversely affecting the real-time requirements.

**Acknowledgments**
We would like to thank Martin Kienzle, Roger Haskin and Christos Polyzois for their comments and ideas during several discussions about the architecture model and model parameters. We would also like to thank Manoj Kumar and for his help in disk parameters.

## References

[1] C.R. Attanasio, M. Butrico, C.A. Polyzois, S.E. Smith, and J.L. Peterson. Design and implementation of a recoverable virtual shared disk. *IBM Research Report RC-19843*, November 1994.

[2] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju. Staggered striping in multimedia information systems. *Proceedings of the fifth intl. conf. on management of data*, may 1994.

[3] Chran-Ham Chang and et.al. High performance TCP/IP and UDP/IP networking in DEC/OSF/1 for alpha AXP. *Proceedings of the 3rd IEEE Symposium on High Performance Distributed Computing*, pages 36–42, August 1994.

[4] E. Chang and A. Zakhor. Scalable video data placement on parallel disk arrays. *Proceedings of storage and retrieval for image and video databases II*, February 1994.

[5] H.-J. Chen and T.D.C. Little. Physical storage organizations for time-dependent multimedia data. *Proceedings of the fourth international conf. on the foundations of data organizations and algorithms*, october 1993.

[6] Asit Dan, Dan Dias, Rajat Mukherjee, Dinkar Sitaram, and Renu Tewari. Buffering and caching in large scale video servers. *COMPCON*, 1995.

[7] Asit Dan and Dinkar Sitaram. Buffer Management Policy for an On-Demand Video Server. IBM Research Report RC 19347, T.J Watson Research Center, Yorktown Heights, New York, 1993.

[8] Craig Freedman and David DeWitt. The SPIFFI scalable video-on-demand server. *SIGMOD*, June 1995.

[9] Donald Gross and Carl M Harris. *Fundamentals of Queueing Theory.* John Wiley and Sons, 1985.

[10] Roger Haskin and Frank L. Stein. A system for delivery of interactive television programming. *COMPCON*, Spring 1995.

[11] J. Hsieh and et.al. Performance of a mass storage system for video-on-demand. *Journal of parallel and distributed computing*, submitted.

[12] IBM. RISC System/6000 7319 model 100 fiber channel switch 16/266. *Product description*, 1994.

[13] J. D. C. Little. A Proof of the Queueing Formula L = $\lambda$ W. *Operations Research*, 9:383 – 387, 1961.

[14] C.L. Liu and J.W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20(1):46 – 61, January 1973.

[15] Rajat Mukherjee, Daniel M. Dias, Christos A. Polyzois, Jerry Leitherer, and Jeffrey S. Lucash. Cost-Performance Issues for Clustered Video Servers. IBM Research Report In Preparation, T.J Watson Research Center, Yorktown Heights, New York, 1994.

[16] Narasimha A.L. Reddy and Jim Wyllie. I/O Issues in a Multimedia System. *IEEE Computer*, 27(3):69 – 74, March 1994.

[17] L.A. Rowe and B.C. Smith. A continous media player. *Proceedings of the 3rd intl wkshp. on network and operating system support for digital audio and video*, November 1992.

[18] Chris Ruemmler and John Wilkes. An Introduction to Disk Drive Modeling. *IEEE Computer*, 27(3):17 – 29, March 1994.

[19] Herb Schwetman. CSIM Reference Manual (Revision 16). MCC Technical Report ACT-ST-252-87, Microelectronics and Computer Technology Corporation, Austin, Texas 78759, May 1992.

[20] H. Takagi. *Queueing Analysis*, volume 1. North Holland, 1991.

[21] Renu Tewari, Dan Dias, and Rajat Mukherjee. Design of a clustered video server. *IBM-Research Report -in preparation*, 1995.

[22] Renu Tewari, Dan Dias, Christos Polyzois, and Rajat Mukherjee. High availability in clustered video servers. *in preparation*, 1995.

[23] Ronald Vetter. ATM concepts, architecture and protocols. *CACM*, February 1995.

[24] H.M. Vin and P.V. Rangan. Designing a multiuser hdtv storage server. *IEEE journal on selected areas of communications*, January 1993.

# Appendix

An appendix is attached that consists of the required queuing results that can be derived or are given in the queuing theory literature [9]. The symbols used in the equations are described here for completeness.

| | |
|---|---|
| $D$ : | Number of disks |
| $N$: | Number of Nodes |
| $S$ : | Number of streams |
| $N_{fe}$ : | Number of front-end nodes |
| $N_{be}$ : | Number of back-end nodes |
| $\lambda$ : | Stream arrival rate blocks/sec. |
| $1/\mu_d$ : | Mean delay at the disk |
| $\rho_s^{fe}$ : | Utilization of front-end sending port |
| $\rho_s^{be}$ : | Utilization of back-end sending port |
| $\rho_r^{fe}$ : | Utilization of front-end receiving port |
| $\rho_r^{be}$ : | Utilization of back-end receiving port |
| $\rho_s$ : | Utilization of sending port - Flat |
| $\rho_r$ : | Utilization of receiving port - Flat |
| $W_d(t)$ : | Delay distribution at the disk for discrete values |
| $\lambda_s^{fe}$ : | Arrival rate at the front-end sending port |
| $\lambda_r^{fe}$ : | Arrival rate at the front-end receiving port |
| $\lambda_r^{be}$ : | Arrival rate at the back-end receiving port |
| $\lambda_s^{be}$ : | Arrival rate at the back-end sending port |
| $\lambda_s$ : | Arrival rate at the sending port- Flat |
| $\lambda_r$ : | Arrival rate at the receiving port- Flat |
| $1/\mu_{block}$ : | Time to send a large block across switch |
| $1/\mu_{request}$ : | Time to send a small request across switch |
| $1/\mu_s^{fe}$ : | Mean service time at front-end sending port |
| $1/\mu_s^{be}$ : | Mean service time at back-end sending port |
| $1/\mu_s$ : | Mean service time at sending port -Flat |
| $1/\mu_r$ : | Mean service time at receiving port -Flat |
| $W_r^{be}(t)$ : | Delay distribution at the back-end receiving port |
| $W_r^{fe}(t)$ : | Delay distribution at the front-end receiving port |
| $W_s^{fe}(t)$ : | Delay distribution at the front-end sending port |
| $W_s^{be}(t)$ : | Delay distribution at the back-end sending port |
| $\delta_{net}$ : | Mean delay in the switching network |
| $var_s^{be}[delay]$ : | Delay variance at the back-end sending port |
| $var_s^{be}[service]$ : | Service time variance at the back-end sending port |
| $E_s^{be}[service^k]$ : | kth moment of the service time distribution at back-end sending port |
| $E_r^{fe}[delay^k]$ : | kth moment of the delay distribution at front-end receiving port |
| $E_r^{fe}[service^k]$ : | kth moment of the service time distribution at front-end receiving port |
| $\sigma_r^2$ : | Variance in service time at receiving port - Flat |
| $var_s[delay]$ : | Variance in delay at sending port - Flat |
| $var_s[service]$ : | Variance in the service time at sending port - Flat |
| $E_s[service^k]$ : | kth moment of the service time at sending port -Flat |

## Disk

The queue length distribution at the disk, $p_n$, (M/D/1 queue) is given by,
$p_0 = (1-\rho)\ where\ \rho = \lambda_d/\mu_d$
$p_1 = (1-\rho)(e^{\rho}-1)$
$p_n = (1-\rho)\Sigma_{j=1}^{n}(-1)^{n-j}e^{j\rho}[\frac{(j\rho)^{n-j}}{(n-j)!} + \frac{(j\rho)^{n-j-1}}{(n-j-1)!}]\ \ n \geq 2$
The delay distribution at the disk for discrete values is,
$W_d(t = i/\mu_d) = p_i;\ i = 1, 2, ...$

## Switch

The arrival rates at the switch ports are:
$\lambda_s^{fe} = \lambda S/N_{fe}$
$\lambda_r^{be} = \lambda S/N_{be}$
$\lambda_s^{be} = \lambda S/N_{be}$
$\lambda_r^{fe} = \lambda S/N_{fe}$

The port utilizations (assuming all arrival rates are equal) are:
$\rho_r^{be} = \lambda_r^{be}/\mu_{request}$
$\rho_s^{fe} = \rho_r^{be} + (\rho_r^{be})^2/2(1-\rho_r^{be})$.
$\rho_r^{fe} = \lambda_r^{fe}/\mu_{block}$.
$\rho_s^{be} = \rho_r^{fe} + (\rho_r^{fe})^2/2(1-\rho_r^{fe})$.

The mean service times (from the mean delay at the corresponding receiving port's M/D/1 queue) at the sending port (M/G/1 queue) are:
$1/\mu_s^{fe} = 1/\mu_{request} + \frac{\rho_r^{be}}{2(1-\rho_r^{be})\mu_{request}}$
$1/\mu_s^{be} = 1/\mu_{block} + \frac{\rho_r^{fe}}{2(1-\rho_r^{fe})\mu_{block}}$

The delay distributions at the receiving ports (M/D/1 queues), $W_r^{be}(t)$ and $W_r^{fe}(t)$ can be obtained similar to that of $W_d(t)$.

The delay distributions at the sending ports (M/G/1 queue) are:
$W_s^{fe}(t) = (1-\rho_s^{fe})\sum_{n=0}^{\infty}(\rho_s^{fe})^n[\mu_s^{fe}\int_0^t(1-W_r^{be}(x))dx]^{(n)}$, where (n) is the nth convolution.
Similarly,
$W_s^{be}(t) = (1-\rho_s^{be})\sum_{n=0}^{\infty}(\rho_s^{be})^n[\mu_s^{be}\int_0^t(1-W_r^{fe}(x))dx]^{(n)}$

The mean network delay is the sum of the mean delay at the front-end sending port and the mean delay at the back-end sending port's M/G/1 queues and is given by,

$$\delta_{net} = 1/\mu_{request} + \frac{\rho_r^{be}}{2(1-\rho_r^{be})\mu_{request}} + \frac{\lambda_s^{fe}}{2(1-\rho_s^{fe})}[1/\mu_{request}^2 + \frac{\rho_r^{be}}{3(1-\rho_r^{be})\mu_{request}^2} + \frac{(\rho_r^{be})^2}{4(1-\rho_r^{be})^2\mu_{request}^2}] + 1/\mu_{block} + \frac{\rho_r^{fe}}{2(1-\rho_r^{fe})\mu_{block}} + \frac{\lambda_s^{be}}{2(1-\rho_s^{be})}[1/\mu_{block}^2 + \frac{\rho_r^{fe}}{3(1-\rho_r^{fe})\mu_{block}^2} + \frac{(\rho_r^{fe})^2}{4(1-\rho_r^{fe})^2\mu_{block}^2}].$$

The variance of the delay distribution of the sending port's M/G/1 queue is,
$var_s^{be}[delay] = var_s^{be}[service] + \lambda_s^{be}E_s^{be}[service^3]/3(1-\rho_s^{be}) + (\lambda_s^{be})^2(E_s^{be}[service^2])^2/4(1-\rho_s^{be})^2$.
$var_s^{be}[service] = E_s^{be}[service^2] - (1/\mu_s^{be})^2$, from the definition of variance.
$E_s^{be}[service^k] = E_r^{fe}[delay^k]$
The kth moment of the delay distribution of an M/G/1 queue can be derived from the Laplace transform of the delay distribution and is,
$E_r^{fe}[delay^k] = E_r^{fe}[service^k] + \frac{\lambda_r^{fe}}{(1-\rho_r^{fe})}\sum_{j=1}^{k}\binom{k}{j}E_r^{fe}[delay^{k-j}]E_r^{fe}[service^{j+1}]/(j+1)$,
$E_r^{fe}[service^k] = (1/\mu_{block})^k$

For the flat architecture the corresponding values are:
$\lambda_s = \lambda S(N-1)/N^2$
$\lambda_r = \lambda S(N-1)/N^2$
$1/\mu_r = (1/\mu_{block} + 1/\mu_{request})/2$
$1/\mu_s = 1/\mu_r + \frac{\lambda_r}{2(1-\rho_r)}(\sigma_r^2 + 1/\mu_r^2)$, from the mean delay of an M/G/1 queue.
$\rho_r = \lambda_r(\mu_{block} + \mu_{request})/(\mu_{block}\mu_{request})$.
$\rho_s = \rho_r + (\rho_r^2 + \lambda_r\sigma_r^2)/2(1-\rho_r)$
$\sigma_r^2 = (1/\mu_{block} - 1/\mu_{request})^2/4$
$var_s[delay] = var_s[service] + \lambda_s E_s[service^3]/3(1-\rho_s) + (\lambda_s)^2(E_s[service^2])^2/4(1-\rho_s)^2$.