Multimedia Tools and Applications 8, 249–283 (1999)
© 1999 Kluwer Academic Publishers. Manufactured in The Netherlands.

Strategic Replication of Video Files in a Distributed Environment

YOUJIP WON yjwon@email.hanyang.ac.kr
Division of Electrical and Computer Engineering, Hanyang University, Seoul, Korea

JAIDEEP SRIVASTAVA srivasta@cs.umn.edu
Department of Computer Science & Engineering, University of Minnesota, MN 55455

Abstract. The *per service* cost has been a serious impediment to wide spread usage of on-line digital continuous media service, especially in the entertainment arena. Although handling continuous media may be achievable due to technology advances in the past few years, its competitiveness in the market with existing service type such as video rental is still in question. In this paper, we propose a model for continuous media service in a distributed infrastructure which has a video warehouse and intermediate storages connected via a high speed communication network, in an effort to reduce the resource requirement to support a set of service requests. The storage resource and network resource to support a set of requests should be properly quantified to a uniform metric to measure the efficiency of the service schedule. We developed a *cost model* which maps the given service schedule to a *quantity*. The proposed cost model is used to capture the amortized resource requirement of the schedule and thus to measure the efficiency of the schedule. The distributed environment consists of a massive scale continuous media server called a *video warehouse,* and *intermediate storages* connected via a high speed communication network. An intermediate storage is located in each neighborhood, and its main purpose is to avoid the repeated delivery of the same file to a neighborhood. We consider a situation where a request for a video file is made sometime in advance. We develop a scheduling algorithm which strategically replicates the requested continuous media files at the various intermediate storages.

Keywords: video caching, storage overflow, video scheduling, continous media delivery, distributed service, cost model

1. Introduction

1.1. Motivation

Entertainment, education, teleconferencing, telemarketing, telemedicine, etc. are a number of promising applications of the *Video-On-Demand* (*VOD*) technology. In some applications such as business and medicine, the cost of the technology may be justifiable, but in other applications such as home entertainment, this technology will have to compete with $2–3 video rental cost. There is a significant opinion in the community that a true *on-demand* video service, i.e., one where the video begins as soon as the customer makes a request for it, cannot be provided at a comparable cost to video rental rate in the near future. Given the fact that there is sufficient consumer dissatisfaction with fast rising cable rates, any expensive technology is not likely to be very successful in the mass entertainment arena. On the other hand, if we examine existing video rental patterns, it is not unreasonable to assume that customers would be satisfied with a *Video-On-Reservation* (*VOR*) service, where the customer makes the service request some time in advance of the actual presentation time, i.e., an hour, a day, etc. We believe it is possible to provide *Video-On-Reservation* (*VOR*) service in a cost-effective manner. Since the set of user requests is available to the service provider in *VOR* service, the server can perform global optimizations based on the user request information as well as the availibility of various resources. For the *Video-On-Demand* service, such pre-planning or off-line computing is not possible, due to its inherent *on-demand* nature.

Given the limits on I/O subsystems of the server, there are many difficulties in supporting multiple service requests directly from a single server, while simultaneously achieving low cost. Furthermore, as the length of the communication route between server and client increases, the cost of maintaining smooth media flow across the network increases dramatically. In this paper we present an approach for providing continuous media service in a distributed manner. The distributed environment consists of a number of *intermediate storages* and *video warehouses*, all connected by a high speed network. When there are multiple storages distributed over the area, a user can get service from various sources other than the central server, and the users can share a file at an intermediate storage. Figure 1 depicts the environment, which consists of video warehouses, VW_i's, intermediate storages IS_i's, and the neighborhoods N_i's of customers. A video warehouse is an archive that may have several thousand video files in its storage system.

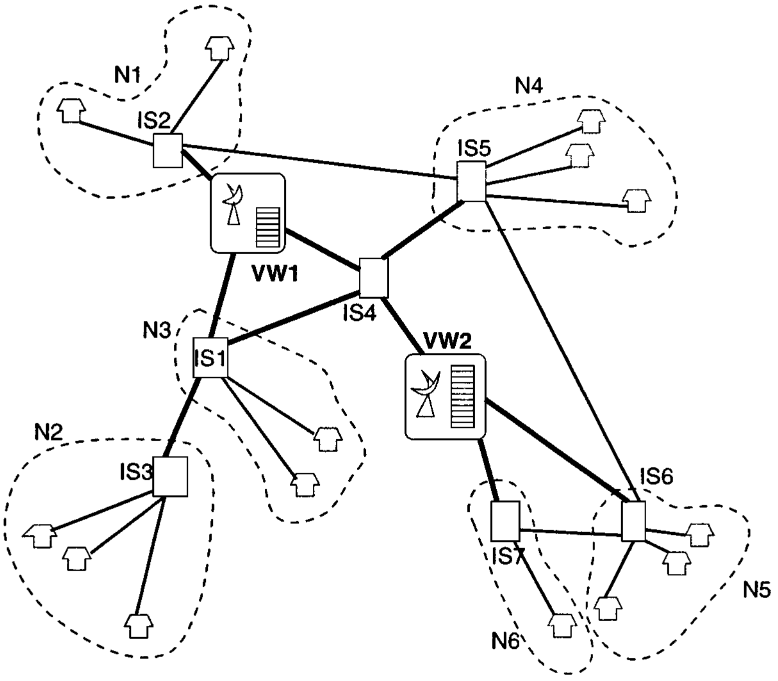


Figure 1. Topological layout.

1.2. Related work

In this section, we briefly introduce the work done in each component of our distributed service environment. Papadimitriou et al. [16] proposed the service model for on-line home entertainment service and also devised an algorithm to arrange the service schedule.

We categorize user requests into two different types based on whether service request is made *in advance* or not: *Video-On-Demand* and *Video-On-Reservation*. Little and Venkatesh [14] classify interactive services into five categories based on the amount of interactivity allowed.

Distributed storage architecture [2, 3, 9, 19, 24] is one of the emerging technology to maintain excessively large amounts of data. In their architectural models, there is a centralized video server which consists of a two-stage hierarchy and a number of relatively small storage systems which are geographically placed in various locations within the metropolitan area. Bianchi et al. [2] investigate various cache management policies at the local switch level, which is dedicated to store a heavily requested file. De Giovanni et al. [9] investigate the communication link allocation between the central video server and local storages. Sienknecht et al. [19] explore the implication of distributed data on the design of data-storage management systems. Brubeck and Rowe [3] developed storage management algorithms for data stored in the hierarchical storage system.

The purpose of the intermediate storage is to temporarily store the files. To exploit the bandwidth capacity and space capacity of the intermediate storage, the files need to be efficiently placed over a set of disks. Little and Venkatesh [15] proposed an algorithm to place video files on the distributed storage system based on the popolarity of the video file. Dan and Sitaram [6] proposed to consider the ratio between the space requirement and bandwidth requirement of a file in allocating the disk space. Wan and Du [23] used the idea of bandwidth to space ratio in determining striping width of a file. Due to huge volume and continuity requirements, data block placement is an important factor in determining the performance of the disk storage. By placing the blocks efficiently, it is possible to support a larger number of concurrent video sessions [12, 22].

Video warehouse uses a high speed network to deliver the service to customers all over the metropolitan area [2, 3, 9]. The existing network infrastructure is not originally designed to support time critical applications. There have been a number of proposals to guarantee the continuity of video stream over high speed communication networks [22]. Another approach in both academia and industry is focusing on utilizing the existing communication medium, such as twisted pair telephone line or coaxial cable line, which is already available to most residential units. The idea of a cable modem is to provide the internet service to PC users over the TV cable system. Such modems can deliver upto 30 Mbps to each residential unit [1, 10]. *Asymmetric Digital Subscriber Line* [1, 20] is one of the most promising technologies to increase the capacity of the copper loops. ADSL is now capable of pumping data at upto 6 Mbps.

Development of an optimal pricing model, i.e., *how much a user has to pay for the service*?, suddenly draws wide attention from various communities. This is because the network infra structure is making the transition from research testbed to commercial enterprise. Pricing models and QoS policies are tightly coupled to each other. Shenker et al. [18] and Cocchi et al. [4] investigate the pricing models of multiple service class networks.

Video Warehouse stores several thousands of continuous media files in its archive. There have been a number of suggestions about how to design a massive scale data server. Hierarchical storage structure in video warehouse is a promising solution to achieve cost-effectiveness of data storage. Ghandeharizadeh and Cyrus [8] and Keinzle et al. [21] worked on hierarchical storage structures to efficiently utilize the bandwidth of several storage types. Doganata and Tantawi [7] provide a cost model for hierarchical-storage-based continous media server. Won and Srivastava [25, 26] provide a stochastic model to measure the throughput of the hierarchical storage server and proposed policies to maintain the files on the staging disks of a hierarchical storage.

1.3. Contributions

A substantial amount of effort is being spent by the research communities to support the continuity requirements of on-line multimedia information delivery. However, economic aspect of servicing a set of users, considering storage and network resources, is paid little attention. In this paper, we propose the usage of intermediate storage to relieve the burden of the central server, i.e., *information provider*, and to reduce the network traffic. We develop a mechanism, *cost model*, which maps the overall resource requirement for supporting the set of requests to the domain of comparable values, i.e., *cost*. The proposed cost model enables the information provider to examine the cost-effectiveness of a certain service delivery schedule. We focus on the problem of minimizing the *cost* for servicing a set of *Video-On-Reservation* requests. The cost of using intermediate storage and the cost of using network determines how to deliver the service to the end-user. We develop an algorithm to strategically replicate the files to intermediate storages to obtain efficient service schedule. An important side benefit of this optimization is that for a given load, the environment will be optimized to utilize the least required network capacity, thus freeing up valuable spare capacity for the truly *Video-On-Demand* applications.

The rest of the paper is organized as follows: Section 2 introduces the model of the environment and the services provided. In Section 3, a mathematical formulation of the problem and the cost model for service delivery is provided. In Section 4, we present our approach to solving the video delivery problem, and in Section 5, we discuss how to handle storage overflow. Performance evaluation is presented in Section 6 and conclusions in Section 7.

2. Video delivery environment and services

2.1. Video-On-Demand and Video-On-Reservation

In the proposed environment, we classify the type of services into two categories, *Video-On-Demand* and *Video-On-Reservation*. In *VOD* service, a server takes an immediate action to service the request, while in *VOR* service, user request is to make the reservation for service in advance. To handle *VOR* requests, time is divided into *cycles*. For example, supposing we consider 24 hours to be the length of a cycle, all requests for videos to be shown during cycle i must arrive at the server by a *predefined time* before the start of cycle i. If 24 hours is too long a period for an advance reservation, one could consider smaller cycles, say 12 or 6 hours. Alternatively, one can have a number of cycles of various lengths, and requests for shorter cycles would have a higher cost. For example, the user gets 75% discount on the regular price if he makes a video request 48 hour in advance of his requested showing time, 24 hour advance with 60% discount, 6 hour advance with 25% discount, etc. Considering the computation time to obtain the schedule, the scheduler has to start the scheduling algorithm early enough so that the schedule information is available by the start of the respective cycle. The advance reservation property of the *VOR* request enables the scheduler to collect all the requests before it starts scheduling.

In scheduling *VOD* requests, there is neither sufficient information nor time to perform the global resource optimization which is possible for *VOR* requests. Furthermore, since overall network and storage capacities are fixed, developing efficient, i.e., *least resource utilizing*, schedules for the *VOR* requests has the added benefit of making more resources available for dynamically arriving *VOD* requests.

2.2. Economics of storage and network

Due to the huge volume and bandwidth constraints of playback, continous media applications impose another dimension of complexity for computer systems. Multimedia file compressed with MPEG-2 encoding [11] scheme requires 3–6 Mbps bandwidth for its playback. Assuming the typical length of a video file to be 110 min, it requires about 3.3 GB of storage space for an average sized video file. With the current market price for hard disk at approximately $100/(Giga Byte), it costs $165,000 to maintain 500 video files on the disk storage, excluding other hardware overhead.

Aside from the cost of disk storage, the problem of interconnecting and maintaining huge number of disks with the existing SCSI interface arises as an obstacle to store 500 video files on the disk subsystem. For example, considering the capacity of a single disk which is commercially available ranges from 4 to 10 GB, at least 50 disk drives need to be interconnected to form a disk storage subsystem. We argue that a hierarchical storage structure, where secondary storage being disk subsystem and tertiary storage being tape library or optical drive is the promising storage architecture for cost effective data management.

The purpose of an intermediate storage is to temporarily store the video files to avoid repeated transmissions to the same neighborhood. Each intermediate storage is primarily responsible for supporting its neighborhood consisting of a number of customers such as residential units or office units. It is not expected that an intermediate storage IS_i will have sufficient capacity to store all the files requested by neighborhood N_i at peak demand. Limited space in the intermediate storage causes some requests to handle a continuous playback across the network. We provide examples for the usage of intermediate storage in figure 2. For a popular video, it is likely that the video file will have to be shown several times in the same neighborhood. Consider figure 2, which shows three requests for *Star Wars* from *VW* in a 90 minute interval. Rather than delivering the video directly from *VW* three times, it is possible to store the video file at *IS* while serving the earliest request. The second and third video requests can now be serviced directly from *IS*. Considering the

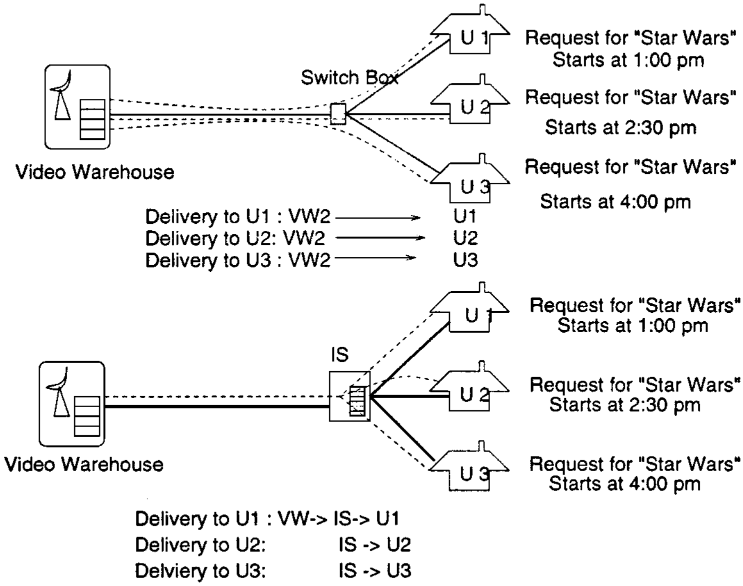


Figure 2. Usage of the intermediate storage system.

high load on the warehouse server and on the metropolitan area network during *prime time* hours, it may be much cheaper to service the three viewers with *video caching*. Figure 2 illustrates the usage of the intermediate storage system for this purpose.

3. Problem formulation

3.1. Service schedule

A user request arriving at the server consists of three attributes, *user_id*, *video_id* and *starting_time*. The *service schedule* $\mathcal{S}$ is a set of information about how to arrange the delivery of the continuous media streams to end users. There are two types of information in a service schedule—*network transfer information* $\mathcal{D} = \{d_1, d_2, \ldots, d_{n_d}\}$ and *usage of the intermediate storage* $\mathcal{C} = \{c_1, \ldots, c_{n_c}\}$. Servicing a video file to end user requires network transfer of the stream and possible caching of the file at the intermediate storage. The network transfer information d_i is $(route_i, t_i^d, id_i)$. $Route_i$ is a sequence of network nodes, i.e., $n_i^{src}, \ldots, n_i^{dst}$, where n_i^{src} can be an intermediate storage or Video Warehouse and n_i^{dst} is an intermediate storage. Instead of representing *file transfer route* as a *source* and *destination* pair, sequence of nodes are used to represent the delivery route. This enables the scheduler to dynamically select the route between each node, depending on the network situation. Delivery information d_i informs the server and intermediate storages that flow of file id_i from n_i^{src} to n_i^{dst} is to begin at t_i^d. Routing information between n_i^{dst} and end user is

Table 1. Notation.

Variables	Description
IS	Set of intermediate storages
IS_i	Intermediate storage i
$srate$	Set of storage costs
$srate(IS_i)$	Charging rate for intermediate storage IS_i (in \$/(byte $\cdot$ s))
$nrate$	Set of network costs
$nrate(i, j)$	Cost rate of the network connection between IS_i and IS_j in \$/byte
m	Number of available video files
$\mathcal{R}$	$\bigcup_{i=1}^{m} \mathcal{R}_i$
$\mathcal{R}_i$	A set of requests for file i
$\mathcal{S}_i$	Service schedule of file i obtained from individual video scheduling
$\mathcal{S}_i^{new}(\Delta t, IS_j)$	Service schedule obtained with the constraints $(\Delta t, IS_j)$
$\mathcal{S}$	Service schedule for all files i.e., $\bigcup_{i=1}^{m} \mathcal{S}_i$
$\Psi(\mathcal{S}_i)$	Total cost of service schedule $\mathcal{S}_i$
$\Psi(\mathcal{S})$	Total cost of service schedule $\mathcal{S}$
$Overflow\_Set(\Delta t,\ IS_j)$	Set of *stays* involved in cache overflow during interval Δt at IS_j
$OF_{\Delta t, IS_j}$	Overflow situation $\Delta t, IS_j$.
$\mathcal{H}_{\Delta t, IS_j, c_i}$	Heat of rescheduling c_i with respect to $OF_{\Delta t, IS_j}$
ρ_i	Length of service for file i
$size_i$	Size of the file i (byte)
d_i	File transfer information, $(route_i, t_i^d, id_i)$
c_i	File residency information, $([t_i^s, t_i^f], loc_i, id_i, n_i^{src}, service\,list)$.

not specified since we assume the path between the user and its local intermediate storage is uniquely defined. The intermediate storage is said to be *local* to a user if it is located in the same neighborhood as the user (for the list of variables see Table 1).

In servicing a set of requests, a video file has to be stored temporarily in various intermediate storages by copying data blocks from the on-going continuous media stream. The purpose of storing a file in an intermediate storage is to temporarily cache the video file for subsequent delivery to other users. The information c_i about temporary storage of a video file is a vector of five elements: $([t_i^s,\ t_i^f],\ loc_i,\ id_i,\ n_{src}, service\,list)$. $[t_i^s,\ t_i^f]$ denotes the interval of caching. t_i^s is the time when the file starts being loaded at the intermediate storage. t_i^f is the start time of the last service. A file resides at intermediate storage to service a set of the requests. The data blocks which are consumed by the last request in chronological order are no longer required. Caching interval $[t_i^s,\ t_i^f]$ is followed by the playback duration of the last service. loc_i and id_i represent the intermediate storage and the respective file. n_i^{src} is the source of the on-going stream from which the data block is copied. n_i^{src} can be either another intermediate storage or *VW*.

3.2. *Cost model*

To measure the overall cost-effectiveness of the service schedule $\mathcal{S}$, there needs to be a generic mechanism which maps the given schedule $\mathcal{S}$ into a *cost*. Schedule $\mathcal{S}$ is a collection of network transfer information d_i's and file residency information c_i's. The mapping mechanism should capture $\mathcal{S}$'s resource consumption in storage devices and the communication network. We define Ψ as a mapping function that transforms service schedule $\mathcal{S}$ into a *quantity*. The *cost of* $\mathcal{S}$ thus corresponds to $\Psi(\mathcal{S})$. We use a monetary metric to represent the cost of a schedule $\mathcal{S}$. The *cost* of a schedule $\mathcal{S}$ is the sum of the cost of using storage devices and the cost of using the communication medium. The network transfer information d_i's and file residency information c_i's have different structure and thus different functions are used to obtain their respective costs. It is worth noting that amortized resource requirement needs to be discriminated from *cost*. Amortized resource requirement is a metric to capture the amount of resources needed. *Cost* is amount of the resource consumed multiplied by *per unit cost*[1] of each resource. In a heterogeneous environment, it is not possible to impose a uniform *per unit cost* on all resources, which have different performance characteristics. For example, *per unit cost* of high performance disk subsystem can be higher than that of a general purpose disk subsystem. *Per unit cost* is inherent to an individual resource entity, e.g., each intermediate storage or each network hop. We develop the functions $\Psi_{\mathcal{C}}(c_i)$ and $\Psi_{\mathcal{D}}(d_i)$ for file residency information c_i, and network transfer information d_i, respectively, which map given information into the domain of *cost*. The cost of schedule $\mathcal{S}$ is hence represented as

$$\Psi(\mathcal{S}) = \sum_{i=1}^{n_d} \Psi_{\mathcal{D}}(d_i) + \sum_{i=1}^{n_c} \Psi_{\mathcal{C}}(c_i). \tag{1}$$

3.2.1. Cost model for storage. For continuous media files, playback length of the data file cannot be determined solely from the file size. This is due to the fact that different files may require different playback rates and thus two files of different sizes may have the same playback length. The amortized resource requirement at an intermediate storage is a function of *file size*, *duration of residency*, and *playback length*. $\Psi_{\mathcal{C}}(c_i)$ is the amortized resource requirement multiplied by the charging rate at the respective intermediate storage. To capture the amortized storage space requirement of the service properly, $\Psi_{\mathcal{C}}(c_i)$ has to reflect all the three attributes. In the charging mechanism for storage, we propose a cost model that considers both the duration and the size of the service. Unit of the storage cost model is \$/(byte · s).

A file is cached on an intermediate storage by copying data blocks from on-going playback, and thus the disk space is filled incrementally until the entire file is cached. We assume that storage space of $size_{id_i}$ needs to be reserved from the start of the caching. We categorize the file residency information c_i into two types: namely, *short residency* and *long residency*, depending on the length of interval $[t_i^s, t_i^f]$. Let ρ_{id_i} be the playback length of a file id_i. c_i is categorized as *short residency* type if $(t_i^f - t_i^s) < \rho_{id_i}$, and as

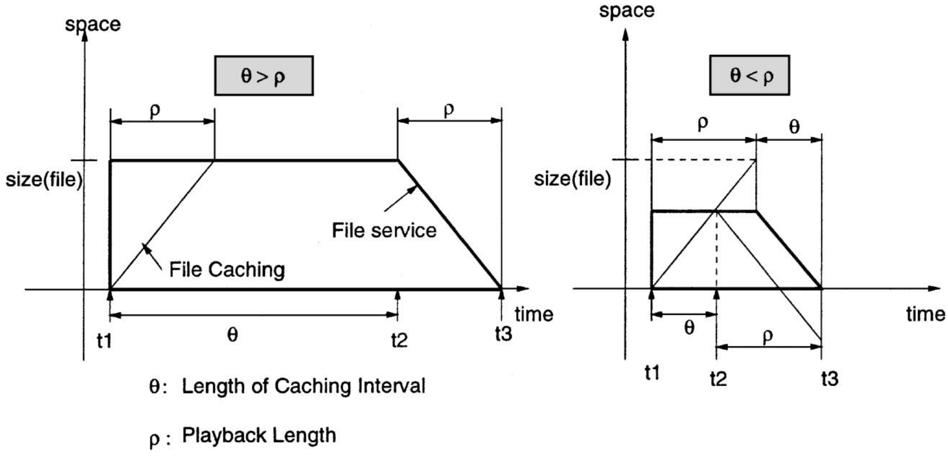


Figure 3. Service length vs. resident duration.

long residency type otherwise. In the rest of the section, c_i implies $([t_i^s, t_i^f], loc, id_i, n_{src},$ *service list*$)$.

Long residency. Let us assume that the service list of c_i is $u_k, u_{k+1}, \ldots, u_l$, with service starting times $t_k, t_{k+1}, \ldots, t_l$, respectively. These users are serviced via a continuous media stream from IS_{loc_i}. Without loss of generality, we assume $t_k \le t_{k+1} \le \cdots \le t_l$. t_i^f corresponds to t_l, respectively. It is not necessary to keep the entire file until the end of the service for u_l, i.e., until time $t_l + \rho_{id_i}$. This is because u_l is the last user in chronological order, and hence the file blocks sent to u_l can be discarded. Thus, the space used by file id_i will decrease linearly from t_l to $t_l + \rho_{id_i}$, and eventually becomes 0 as the service for u_l proceeds to the end, as in figure 3.

Given this model of the storage space, the amortized storage cost for c_i is formulated as

$$\Psi_{\mathcal{C}}(c_i) = srate\big(IS_{loc_i}\big) \cdot size_{id_i}\left(\left(t_i^f - t_i^s\right) + \frac{\rho_{id_i}}{2}\right). \tag{2}$$

$\Psi_{\mathcal{C}}(c_i)$ is equivalent to the area enclosed by the solid line in the left graph in figure 3.

Short residency. It takes ρ_{id_i} time to load the entire file to an intermediate storage. This is because the file is cached to an intermediate storage by copying the data from the on-going playback of a file. In case the time interval between the first and last service time is no more then ρ_{id_i}, i.e., $(t_i^f - t_i^s) \le \rho_{id_i}$, playback for the last request starts before the entire file is loaded. Right graph of figure 3 depicts the storage space requirement for *short residency*. Let t_1 and t_2 be the starting time of the first and last request in the service list in c_i, respectively.

The file starts being loaded at t_1 and is completely loaded by $t_1 + \rho_{id_i}$. The second service starts at t_2, which means that file blocks are gradually discarded from t_2 until $t_2 + \rho_{id_i}$.

From t_2, file loading and file discarding proceed simultaneously until the first service is completed at $t_1 + \rho_{id_i}$. From $t_1 + \rho_{id_i}$, there is only file discarding until the last service, i.e., the second service in this case, is completed at $t_2 + \rho_{id_i}$. Hence, the size of the storage space which needs to be reserved at the start of the stay is $size_{id_i} \cdot (\frac{t_i^f - t_i^s}{\rho_{id_i}})$. The amortized storage cost for c_i for *short residency* can be formulated as follows:

$$\Psi_{\mathcal{C}}(c_i) = srate\big(IS_{loc_i}\big) \cdot size_{id_i}\left(\left(t_i^f - t_i^s\right) + \frac{(t_i^f - t_i^s)^2}{2\rho_{id_i}}\right) \tag{3}$$

Srate(IS_{loc_i}) is the charging rate of unit resource for IS_{loc_i}. The actual value is determined by the service provider based on the characteristics of the storage device.

3.2.2. Cost model for network. The objective of the cost model for the communication network is to properly capture the amortized bandwidth cost to service a set of requests. A certain amount of network bandwidth needs to be guaranteed to deliver the service to end users with the given *QoS*. The amount of bandwidth to be reserved for each stream is determined by the *QoS* parameter. A number of policies have been proposed to provide the *end-to-end* bandwidth requirement of the continuous media playback with given QoS [13]. Network transfer information d_i is a vector of three elements, $(route_i, t_i^d, id_i)$. Let B_{id_i} be the bandwidth requirement for file id_i's playback. The amortized bandwidth requirement for d_i corresponds to $\rho_{id_i} B_{id_i}$ bytes. As in the storage cost model, we use a monetary metric as the basic unit of cost, i.e., $/byte. The network charging rate can be defined on either end-to-end basis or per hop basis. Let $route_i$ be $(n_i^1, \ldots, n_i^{n_{nw}})$ of d_i, where n_i^1 and $n_i^{d_i}$ correspond to n_i^{src} and n_i^{dst}. $nrate(n_i^j, n_i^k)$ is the charging rate of the route between n_i^j and n_i^k. In general, *nrate* (n_i^j, n_i^k) will depend on various factors, e.g., network topology, link capacities, routing, etc. However, we assume the underlying communication infrastructure maps all of these factors into a cost (i.e., $ amount) per proposed unit, i.e., byte. We assume that $nrate(n_i, n_j)$ is fixed for the duration of video scheduling and service, and is also known a priori. Depending on the underlying network structure, charging rate can be defined on *per hop basis* or *end-to-end basis*. The function $\Psi_{\mathcal{D}}(d_i)$ which quantifies the network transfer information d_i is formulated as follows:

$$\Psi_{\mathcal{D}}(d_i) = \begin{cases} nrate\big(n_i^{src}, n_i^{dst}\big) \cdot \rho_{id} \cdot B_{id} & \text{end-to-end basis network rate} \\ \displaystyle\sum_{j=1}^{n_{nw}-1} nrate\big(n_i^j, n_i^{j+1}\big) \cdot \rho_{id} \cdot B_{id} & \text{per hop basis network rate.} \end{cases} \tag{4}$$

3.3. Problem description

We can view the overall service schedule $\mathcal{S}$ as the union of the service schedules for each file i, $\mathcal{S}_i$. The proposed mapping function Ψ and the cost models for storage and network

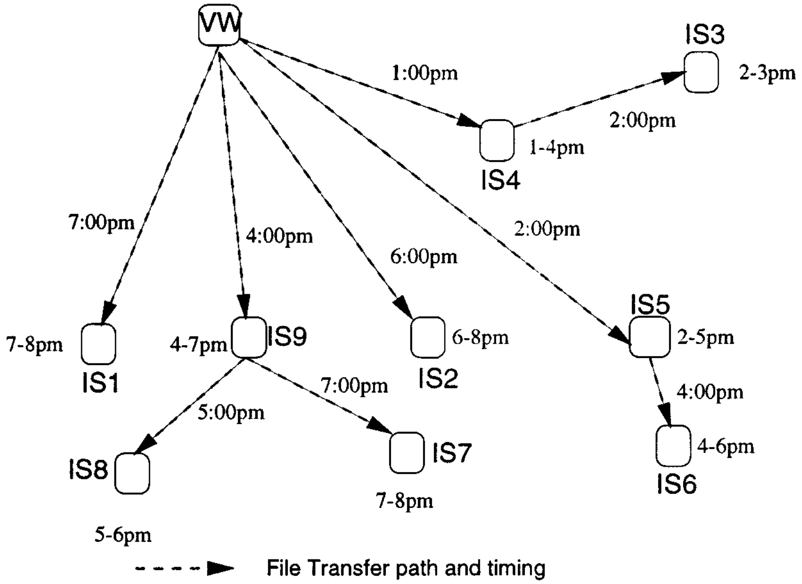


Figure 4. Tree structured layout of a service schedule.

enable the scheduler to compute the cost of servicing a set of requests. Similarly, the scheduler can compute the cost of a schedule $\mathcal{S}_i$ for video file i, i.e., $\sum_{i=1}^{m} \Psi(\mathcal{S}_i)$ for a cycle. The cost of servicing all requests in a cycle is $\sum_{i=1}^{m} \Psi(\mathcal{S}_i)$, i.e., $\Psi(\mathcal{S}) = \sum_{i=1}^{m} \mathcal{S}_i$. A file can be shipped to an intermediate storage directly from the video warehouse or from an intermediate storage, with cost being the determining factor. The service schedule of a video i, for a cycle, forms a tree structure as in figure 4. The goal of the video scheduler is to generate a service schedule with minimum cost. The service scheduling problem can now be stated as follows:

Definition 1 (*Video Scheduling Problem* ($\mathcal{VSP}$)). Given a set of video requests $\mathcal{R}$, a set of network edges connecting a video warehouse *VW*, and a set of intermediate storages, the charging rates for intermediate storage *srate* and network *nrate*, the storage capacity of each intermediate storage, find the set of file service schedules, $\mathcal{S}_{opt}$ such that

$$\forall \mathcal{S}, \quad \Psi(\mathcal{S}_{opt}) \leq \Psi(\mathcal{S})$$

Unfortunately, the problem $\mathcal{VSP}$ is NP-*complete*, and thus an efficient algorithm for the optimal solution is unlikely. The details of the proof are provided in the Appendix. Due to the intractable nature of $\mathcal{VSP}$, we focus our interest on developing an efficient heuristic algorithm.

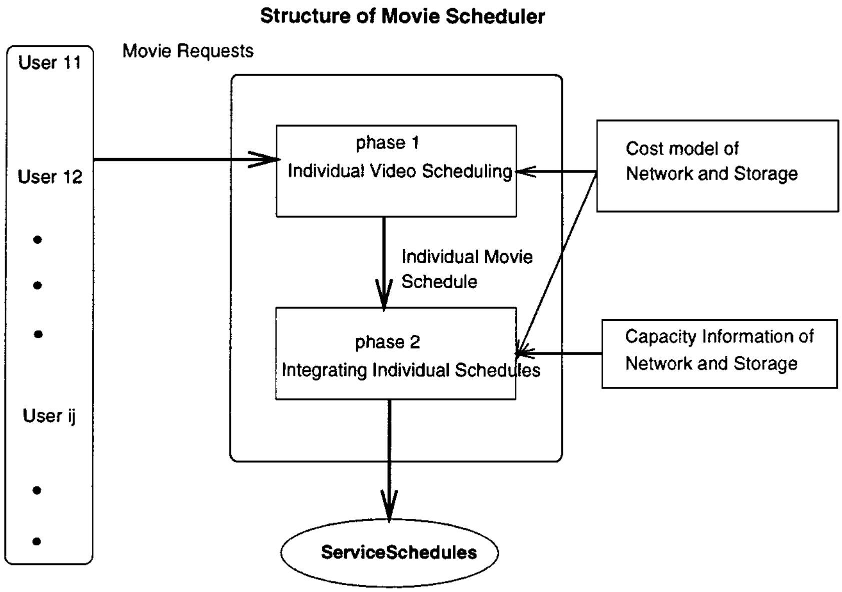


Figure 5. Structure of video scheduler.

4. Scheduling of service delivery

4.1. Design of video scheduler

4.1.1. Two phase scheduling. *Video Scheduler* is a program in charge of arranging the service delivery. The objective of the scheduler is to determine the *service schedule $\mathcal{S}$ with minimum* $\Psi(\mathcal{S})$, given the set of service requests. We approach the problem with a two phase algorithm. Figure 5 depicts the structure of the video scheduling algorithm.

1. *Individual video scheduling*: Find the optimal schedule $\mathcal{S}_i$ for each file i individually, assuming that the capacity of intermediate storage is large enough to store any one video file. $\mathcal{S}_i$ determines (1) the network transfer route of the continous media stream and (2) the residency period and location of the file. In figure 2, user u_3 has requested a video beginning at 4 pm. Two other users have requested the same video, but at different start times prior to u_3's start time. From the aspect of resource requirement, it may be less expensive to download the video at *IS* when it is delivered to u_1, to be later delivered to the other users, than to service the three users directly from the video warehouse. The objective of the individual video scheduling is given the set of user requests for file i, to devise $\mathcal{S}_i$ with the minimum $\Psi(\mathcal{S}_i)$.
2. *Integrating individual video schedules*: Integrate the schedules $\mathcal{S}_i$, taking into account the storage capacity constraints, and resolving any resulting storage overflows. In computing the individual video schedules, the scheduler does not consider the space limitation in the

intermediate storage. An intermediate storage may not be able to accommodate all the video files scheduled for it during a certain time interval because the sum of the file sizes may exceed its capacity. This situation is called *storage overflow*. After computing the individual video schedules, the scheduler examines the individual video schedules and detects storage overflow situations. The scheduler runs the overflow resolution algorithm by rescheduling some of the requests. The integration of the individual video scheduling consists of *overflow detection* and *overflow resolution*. Resolving storage overflow may result in a less efficient delivery schedule, and thus increase the total service cost. Hence, a key objective of integrating individual video schedules is to minimize the increase in the overall service cost.

If the capacity of every intermediate storage is sufficiently large so that it can accommodate all the video files residing at it at any time, scheduling can be completed in phase 1. However, service requests are not distributed evenly over time. For example, the existence of *prime time* confirms the uneven distribution of request arrival times in TV entertainment. Thus, we assume it is not economical for the service provider to make all the intermediate storages large enough to store the peak service requests.

4.1.2. Why two phase scheduling approach?. It is possible to take into account the space availability information in the first phase and thus storage overflow can be avoided. One phase scheduling approach can be sketched as follows.

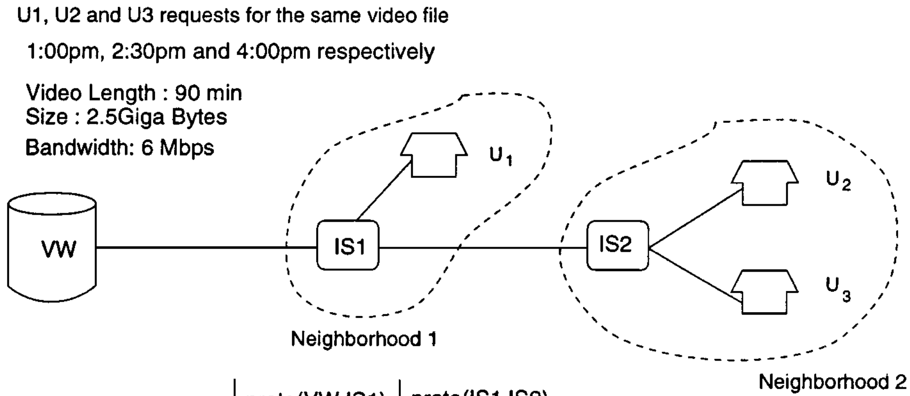
When the scheduler computes the schedule for a file id_i, it considers the intermediate storages with sufficient available space as a possible caching site. The scheduler reduces the space availability information at the respective intermediate storage according to $\mathcal{S}_{id_i}$ after $\mathcal{S}_{id_i}$ is obtained.

Under one phase scheduling paradigm, the scheduler is not given an opportunity to analyze the scheduling priorities among the files. Scheduling priority of a file means its relative importance to consume the limited resources prior to the other files. Scheduling priority can be determined by the popularity of the video file. However, without reflecting the actual set of requests, the respective time and the respective locations, it is not possible to determine which file or request has priority to consume the limited resources first. Analyzing the priority of the given set of requests introduces another stage in scheduling.

In our scheduling policy, *individual video scheduling* and *storage overflow detection* enables the scheduler to globally analyze the service cost for each file i, i.e., $\mathcal{S}_i$, $i = 1, \ldots, m$, and to determine the scheduling priorities among themselves. The file called *victim*, which is determined to have lowest priority in scheduling, is thus rescheduled to resolve *storage overflow*.

4.2. Individual video scheduling

In finding the service schedule $\mathcal{S}$, the scheduler collects the requests for the cycle and partitions them into sets $\mathcal{R}_i$, $i = 1, \ldots, m$ for each of the m distinct video files requested.



	nrate(VW,IS1)	nrate(IS1,IS2)	
Network Cost	¢ 0.2	¢ 0.1	cents/(Mbps*Sec)

	VW	IS1	IS2	
Storage Cost	$0	$1	$2	$/(GBxHour)

Figure 6. Service requests with network cost and storage cost.

$\mathcal{R}_i$ is to the set of requests for video i. The schedule $\mathcal{S}_i$ for each $\mathcal{R}_i$ is computed individually, i.e., in computing $\mathcal{S}_i$ the scheduler does not consider the resources required to service the other sets $\mathcal{R}_j$, $j \neq i$. In the individual video scheduling phase, the scheduler focuses on minimizing the cost of each $\mathcal{S}_i$, $i = 1, \ldots, m$. Figure 6 illustrates a sample topological layout, with respective *nrate*, *srate*, and user requests. We enumerate some of the possible video schedules for figure 6 and the cost of each schedule. Let *id*, ρ_{id} and B_{id} be the file id, playback length, and bandwidth requirement of the file, respectively. The cost is computed using the mapping function Ψ described in Section 3.3. Since all video files reside at *VW* permanently, the cost of storing video files at the *VW* is assumed to be 0, i.e., $sratet(VW) = 0$. IS_1 is *local* to U_1 and IS_2 is *local* to $\{U_2, U_3\}$. We assume that ρ_{id} is 90 min, and $size_{id}$ is 2.5 GB. We assume 6 Mbps bandwidth needs to be reserved for the playback of the given file.

Schedule $\mathcal{S}$1: *All the requests for videos are fulfilled directly from the video warehouse.* $\mathcal{S}1$ is as follows: $\mathcal{S}1 = \{d_1, d_2, d_3\}$ and $d_1 = ((VW, IS_1), 1{:}00, id)$, $d_2 = ((VW, IS_1, IS_2), 2{:}30, id)$, $d_3 = ((VW, IS_1, IS_2), 4{:}00, id)$. Since there is no storage cost incurred in the schedule, the cost of the schedule consists only of the network cost. The playback length of the file is 90 min and it requires 6 Mbps bandwidth. Service for U_1 uses only the edge (VW, IS_1). The cost for servicing U_1, i.e., $\Psi_{\mathcal{D}}(d_1)$ is $nrate(VW, IS_1) \cdot 90 \cdot 60 \cdot 6 \cdot 0.002$. Likewise, we can compute $\Psi_{\mathcal{D}}(d_2)$ and $\Psi_{\mathcal{D}}(d_3)$. $\Psi(\mathcal{S}1)$ corresponds to $\sum_{i=1}^{3} \Psi_{\mathcal{D}}(d_i)$, and thus $90 \cdot 60 \cdot 6 \cdot 0.008 = \$259.2$.

Schedule $\mathcal{S}$2: *While U_1 gets service directly from the VW, IS_1 caches the file. U_2, and U_3 are serviced from the video file cached at IS_1*: $\mathcal{S}2 = \{d_1, d_2, d_3, c_1\}$ where $d_1 = ((VW, IS_1),$

1:00, *id*), $d_2 = ((IS_1, IS_2)$, 2:30, *id*), $d_3 = ((IS_1, IS_2)$, 4:00, *id*), $c_1 =$ ([1:00, 4:00], IS_1, *id*, *VW*,(U_1, U_2)). The cost for servicing U_1 is the same as in Schedule $\mathcal{S}1$. With $\mathcal{S}2$, the file is cached at IS_1 to avoid repeated delivery over the network (*VW*, IS_1) in servicing U_2 and U_3. We can compute the storage cost using the formula given in Eq. (2). A storage space of 2.5 GB has to be reserved from 1:00 pm to 4:00 pm, and the length of the service is 1.5 hours. $\Psi_{\mathcal{C}}(c_1)$ corresponds to $3 \cdot 1 \cdot 2.5 + \frac{1.5 \cdot 1 \cdot 2.5}{2} =$ \$9.375.

The cost for d_2 and d_3, i.e., $\Psi_{\mathcal{D}}(d_2) + \Psi_{\mathcal{D}}(d_3)$ corresponds to $2 \cdot 90 \cdot 60 \cdot 6 \cdot 0.001 =$ \$64.8. The cost for servicing d_1 remains unchanged at \$64.8. Thus, $\Psi(\mathcal{S}2)$ is \$138.975.

Schedule $\mathcal{S}$3: Schedule $\mathcal{S}3$ is enhanced from $\mathcal{S}2$. $\mathcal{S}3 = \{d_1, d_2, c_1, c_2\}$ and $d_1 = ((VW, IS_1)$, 1:00, *id*), $d_2 = ((IS_1, IS_2)$, 2:30, *id*), $c_1 =$ ([1:00, 2:30], IS_1, *id*, *VW*, U_1), $c_2 =$ ([2:30, 4:00], IS_2, *id*, IS_1, U_2). While serving U_2, the file is cached at IS_2. U_3 gets service from IS_2. By loading the file at IS_2, it is possible to eliminate the network transmission cost in servicing U_3. The cost for servicing U_1 remains the same. $\Psi_{\mathcal{C}}(c_1)$ is $1.5 \cdot 2.5 \cdot 1 + \frac{1.5 \cdot 2.5 \cdot 1}{2} =$ \$5.625 and $\Psi_{\mathcal{C}}(c_2)$ is \$11.25. $\Psi_{\mathcal{D}}(d_2)$, which is network cost for servicing U_2, is $90 \cdot 60 \cdot 6 \cdot 0.001 =$ \$32.4. Hence, $\Psi(\mathcal{S}3)$ is $64.8 + 32.4 + 11.25 + 5.625$ $=$ \$114.075.

Based on the proposed cost model, $\mathcal{S}3$ turns out to be the most cost-effective schedule to service the three users in figure 6.

Table 2 presents a skeleton algorithm for the first phase. Function *find_video_schedule*( ) in Table 2 computes the service schedule for each file. Papadimitriou et al. [16] proposed a greedy heuristic based on rectilinear dynamic programming, which can be used as an implementation of the function *find_video_schedule*( ). Let us assume that there are l users $u_1, \ldots, u_l$ requesting video i, M available video files, and N intermediate storages. The users are numbered chronologically with respect to service start time, i.e., $t_i \leq t_{i+1}$. The steps below describe the function *find_video_schedule*( ) in Table 2.

1. The algorithm iterates l times, i.e., once for each user U_i, $i = 1, \ldots, l$.
2. For u_i, given the schedule and respective cost found for $u_1, \ldots, u_{i-1}$, the scheduler computes the incremental network cost and storage cost for servicing u_i via each $IS_1, \ldots, IS_N$.
3. To service u_i in addition to the users $u_1, \ldots, u_{i-1}$, the existing schedule has to be updated in one of the following ways: (1) The resident period of the file at a certain intermediate storage has to be extended, or (2) another intermediate storage which has not been used

Table 2. Pseudocode for function of solving the individual video scheduling problem.

Algorithm 1 IVSP_solve

```
IVSP_solve (M, R, nrate, srate){
    for each files i ∈ M
        S_i^good ← find_video_schedule(R_i, nrate, srate);
    Ψ(S^good) ← Σ_{i∈M} Ψ(S_i^good);
    return(S^good);
}
```

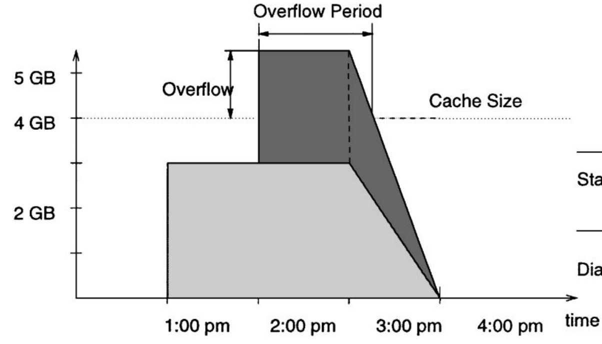
in servicing $u_1, \ldots, u_{i-1}$ is introduced to cache the file. Either situation creates extra storage cost and network cost, or (3) the request is serviced from *VW*.

4. All the intermediate storages $IS_1, \ldots, IS_N$ are considered in a new schedule for servicing $u_1, \ldots, u_i$. The scheduler computes the incremental cost for each. An updated schedule with the minimum incremental cost would be chosen as the schedule for servicing $u_1, \ldots, u_i$.
5. Considering the graph structure of the network, there can be more than one path between any pair of nodes, each of which can be *VW* or intermediate storage. If a new intermediate storage is introduced to cache the file, the scheduler has to compute the network transmission cost of transferring a file to a new cache.

4.3. Integration of individual video schedules into a global schedule

It is possible that one or more intermediate storages are over-committed during certain time intervals when the individual schedules are integrated. Figure 7 illustrates this situation, where intermediate storage is over-committed during the interval [1:30 pm, 2:40 pm] approximately. In this *Storage Overflow* situation, the scheduler has to reschedule some of the files involved in the overflow so that the file does not reside at the intermediate storage during the overflow period. Possible rescheduling strategies are to supply some videos directly from *VW* or from the nearest intermediate storage that stores them. Specifically, the following decisions should be made for each *storage overflow* situation: (1) Which video(s) are selected as victims? and (2) for each victim to be rescheduled, how to compute the new schedule for the victim?

Victim is a file that is selected to be rescheduled. In *most* cases rescheduling entails additional cost. However, there can be cases in which a less expensive schedule is found as a result of rescheduling. This is because the schedule for victim $\mathcal{S}_{victim}$ is computed via a *greedy* heuristic in the first phase, and hence there can exist a less expensive schedule for $\mathcal{S}_{victim}$. An important goal of handling *storage overflow* is to resolve the storage overflow with least possible cost increase in the scheduling which is usually incurred as a result of rescheduling.



	length	user	starting time
Star Wars	1Hr	U1	1:00 pm
	3 GB	U2	3:00 pm
Dial M	1 hr	U3	2:00 pm
	2.5 GB	U4	3:00 pm

Figure 7. Change of available space at intermediate storage.

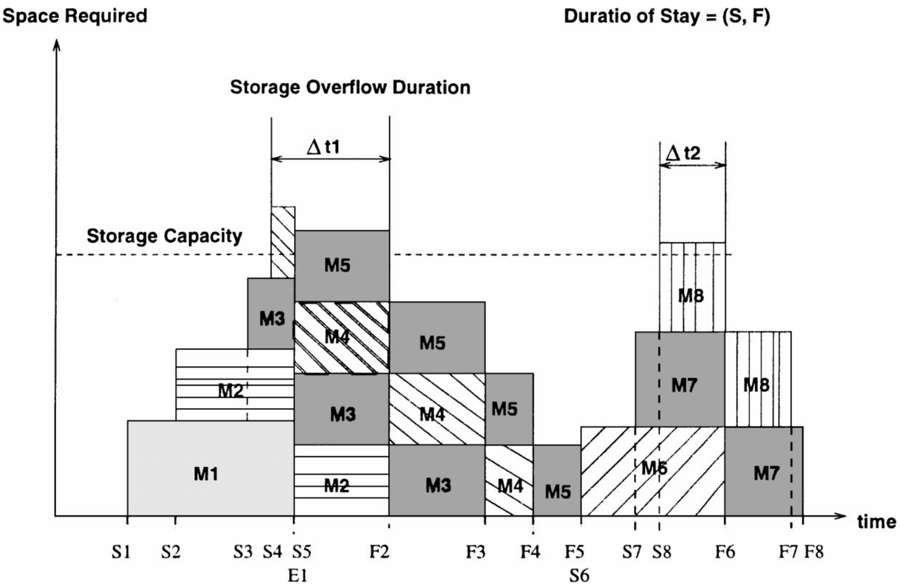


Figure 8. Integrated schedule and respective storage requirement.

5. Handling storage overflow

5.1. Detection of storage overflow

Storage overflow $OF_{\Delta t, IS_j}$ is identified by its location IS_j and the time interval Δt during which the overflow occurs. The scheduler maintains information about the available space at the intermediate storages. Analyzing this information, namely the *storage requirement* and the *storage availability*, the video scheduler detects all *storage overflow situations*. We define *Overflow_Set*$(IS_j, \Delta t)$ as *the set of file residency information c_i's* which are involved in $OF_{\Delta t, IS_j}$. Figure 8 illustrates the storage space requirement of the integrated video schedules at an intermediate storage. For the sake of simplicity, figure 8 does not show the gradual decrease in the storage requirement at the end of each file residency period. In figure 8, there are two distinct storage overflow situations, namely in intervals Δt_1 and Δt_2. The two overflow sets are: *Overflow_Set*$(\Delta t_1, IS) = \{(M_1, [S_1, F_1]), (M_2, [S_2, F_2]), (M_3, [S_3, F_3]), (M_4, [S_4, F_4]), (M_5, [S_5, F_5])\}$ where $\Delta t_1 = [S_4, F_2]$ and *Overflow_Set*$(\Delta t_2, IS) = \{(M_6, [S_6, F_6]), (M_7, [S_7, F_7]), (M_8, [S_8, F_8])\}$ where $\Delta t_2 = [S_8, F_6]$.

5.2. Victim selection

$OF_{\Delta t, IS_j}$ is resolved by recomputing the service schedule for some of the files in *Overflow_Set* $(\Delta t, IS_j)$. In recomputing the service schedule of id_i, scheduler takes into account additional constraints in obtaining the new schedule $\mathcal{S}^{new}_{id_i}$. The additional constraints are that id_i

is not allowed to be cached at IS_j *during* Δt and also the *scheduler considers the currently available space at other intermediate storages*. In *Individual Video Scheduling* phase, the scheduler does not consider the space limitation of the intermediate storage. As a result of imposing additional constraints, i.e., *space availability*, possible caching site of file i is restricted to the intermediate storage with sufficient space. Let $\mathcal{S}_i^{new}(\Delta t, IS_j)$ be the schedule obtained by rescheduling file i with respect to the constraints $(\Delta t, IS_j)$. In selecting a victim, the scheduler needs to consider the overhead cost of rescheduling and respective improvement of overflow situation.

5.2.1. Overhead cost. In resolving $OF_{\Delta t, IS_j}$, there can be more than one choice for a victim. Thus, selecting a victim with minimum cost increase is an important factor to obtain an efficient schedule. When more than one file needs to be rescheduled to resolve $OF_{\Delta t, IS_j}$, the total number of choices for a set of victims is bounded by the number of subsets of *Overflow_Set*$(\Delta t, IS_j)$, i.e., $2^{|Overflow\_Set(\Delta t, IS_j)|}$. We would like to analyze the complexity of victim selection using the example in figure 8. Consider the set *Overflow_Set*(Δt_1) in figure 8. Rescheduling of $\{M_1, M_5\}$ resolves the overflow, as does the rescheduling of $\{M_2\}$. The cost increase as a result of rescheduling file id_i with respect to Δt and IS_j is the *overhead cost*, $\Psi(\mathcal{S}_{id_i}^{new}(\Delta t, IS_j)) - \Psi(\mathcal{S}_{id_i})$. If the *overhead cost* is the same for all video files regardless of length and size, it is desirable to reschedule as few video files as possible. However, there are various factors such as file size, a set of user requests for the file, etc., which determine the overhead cost.

5.2.2. Metrics for effective improvement. *Overhead cost* alone is not sufficient information to select a victim. The purpose of rescheduling is to improve the overflow situation, i.e., reduction in the overflow interval or reduction in the worst case space requirement during the overflow interval. Thus, *improvement* in the overflow situation can be defined along the domain of time, of disk space, or of space-time product. In selecting a victim, the overhead cost of rescheduling and respective improvements needs to be considered. Let $\mathcal{I}_{\Delta t, IS_j, c_i}$ denote the improvement over $OF_{\Delta t, IS_j}$ accomplished by rescheduling file id_i with respect to Δt and IS_j. We define *heat*, $H_{\Delta t, IS_j, c_i}$ as the *improvement per overhead cost* of rescheduling file id_i with respect to $OF_{\Delta t, IS_j}$. *Heat* is defined as:

$$\mathcal{H}_{\Delta t, IS_j}(c_i) = \frac{I_{\Delta t, IS_j}(c_i)}{\Psi\left(\mathcal{S}_{id_i}^{new}(\Delta t, IS_j, c_i)\right) - \Psi\left(\mathcal{S}_{id_i}\right)}. \tag{5}$$

Heat is used as the selection criterion for *victim*. A file with a larger *heat* means that its rescheduling accomplishes *better* improvement with the same cost increase and thus is preferred for rescheduling in resolving an overflow situation. To determine *better* improvement, a metric for improvement has to be defined beforehand. There are a number of metrics for $\mathcal{I}_{\Delta t, IS_j}(c_i)$, i.e., *time*, *space*, or *time-space product*. Each of the metrics have different interpretation of improvement and hence different files can be selected as victim under different metrics. We describe four metrics for *heat* in greater detail in Section 5.3 and compare the performance through extensive simulation.

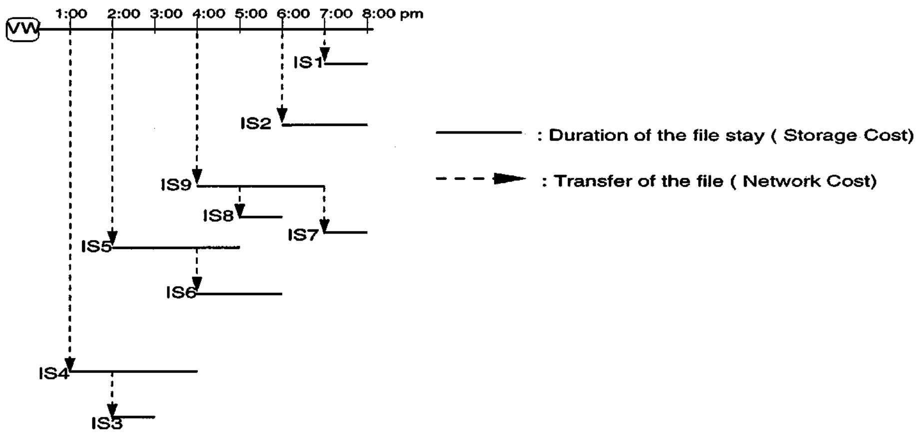


Figure 9. Service schedule of a file.

5.2.3. Rescheduling of a file. The file transfer routes to various intermediate storages form a tree structure, as shown in figure 9, which is based on the schedule in figure 4. In figure 9, solid horizontal lines represent the residency of the file in the respective intermediate storage and dashed vertical lines represent the network transfer route. During residency at an intermediate storage, the file is serviced to one or more users who requested it. It is not possible for two different dashed lines to have the same horizontal solid line segment as a sink, and hence the graph forms a tree structure. The rescheduling of a file i to resolve overflow $OF_{\Delta t, IS_j}$ is to recompute the service schedule i, given that a *solid horizontal line at* IS_j *during* Δt *is not allowed* for file i.

5.3. *Storage overflow resolution algorithm*

Table 3 has the skeleton algorithm for the storage overflow resolution problem. $\mathcal{SORP}$_solve( ) in Table 3 takes service schedule $\mathcal{S}$, information about intermediate storages, which includes the overflow sets, storage usage information, and storage capacity as input. Network rate *nrate* and storage rate *srate* are globally available. $\mathcal{S}$ is a service schedule obtained from the individual video scheduling phase. Let $\mathcal{S}_{id_i}^{new}(\Delta t, IS_j)$ be the schedule for file id_i obtained with the constraint that file id is not allowed to stay at IS_j during Δt. In most cases, changes to $\mathcal{S}_{id_i}$ by rescheduling results in an increase in *cost* for file id_i, i.e., $\Psi(\mathcal{S}_{id_i}(\Delta t, IS_j)) > \Psi(\mathcal{S}_{id_i})$. This is due to the fact that rescheduling a video with storage constraints causes the video to be delivered to the users in a less efficient way than in the previous service schedule, resulting in an unavoidable increase in service cost. However, since the schedule $\mathcal{S}_{id_i}$ is obtained via a heuristic, a less expensive schedule may be found as a result of resolving overflow under certain circumstances. According to our experiment, under various conditions of storage capacity, network cost, storage cost, and user access patterns, in 1.2% of all situations the service schedule actually becomes less expensive after storage overflow resolution. Let us define $\mathcal{S}^{SORP}$ as a service schedule

Table 3. Pseudocode for $\mathcal{SORP}$_solve(*SvcSchedule, C, Overflow_Set*).

```
1:  IS: set of intermediate storages ;
2:  Overflow_Set : set of storage overflows ;
3:  Overflow_Set(IS_j) : set of storage overflows at IS_j ;
4:  S_min, S_tmp : Service Schedule ;
5:  SORP_solve(S, IS) {
6:      Resolved = False ;
7:      While (Resolved != TRUE) {
8:          For all IS_j ∈ IS {
9:              For all Overflow_Set(Δt, IS_j) ∈ Overflow_Set(IS_j) {
10:                 For all c_i ∈ Overflow_Set(Δt, IS_j) {
11:                     S_tmp = ReflectiveGreedy(Δt, IS_j, c_i) ;
12:                     heat = ComputeHeat(S_id_i, S_tmp) ;
13:                     If (heat ≤ minheat){
14:                         heat = minheat ;
15:                         S_min = S_tmp ;
16:                         victim = id_i ;
17:                     }
18:                 }
19:             }
20:         }
21:         S_victim = S_min ;
22:         Resolved = True ;
23:         For all IS_j ∈ IS {
24:             UpdateStorageUsage(S_min) ;
25:             Resolved = Resolved and DetectStorageOverflow(IS_j)
26:         }
27:     }
28: }
```

obtained as a result of storage overflow resolution on schedule $\mathcal{S}$. The increase in the total service of cost is

$$\Delta Cost = \Psi(\mathcal{S}^{SORP}) - \Psi(\mathcal{S}). \tag{6}$$

$\mathcal{S}^{SORP}$ is a new set of video schedules which resulted from the $\mathcal{SORP}$*_solve*( ) algorithm. The ultimate objective in the Storage Overflow Resolution phase is to minimize Δ*Cost*.

For each iteration of the outermost while loop (lines 7–28) of *SORP_solve*( ) in Table 3, *SORP_solve* examines all residency information c_i which is involved in one of the overflows. *SORP_solve*( ) computes the new service schedule for the respective video file $\mathcal{S}^{new}_{id_i}(\Delta t, IS_j)$ for file id_i in c_i and computes the *effective* improvement of the rescheduling. The file with largest *effective* improvement is chosen as the victim in each iteration of the lines 7–28 while loop.

We introduced *heat* as a criterion for victim selection in Section 5.2. The actual improvement accomplished over the $OF_{\Delta t, IS_j}$ as a result of rescheduling file id_i may not be directly relevant to *overhead cost*. For example, rescheduling file id_i reduces the length of $OF_{\Delta t, IS_j}$ by 10 min with a \$40 overhead cost, while rescheduling file id_j reduces the length by 20 min with an overhead cost of \$50. It is not unreasonable to select file id_j as a *victim*. In another situation, the rescheduling of file id_i improves the overflow situation by 300 MB $\cdot$ min, and the overhead cost is \$30. Meanwhile, the improvement of rescheduling file id_j is 50 MB $\cdot$ min and the overhead cost is \$25. It may be desirable to select file id_i as victim, especially when more victims need to be rescheduled in addition to rescheduling of file id_i or id_j.

Let Δt of $OF_{\Delta t, IS_j}$ be $[t_s, t_f]$ and interval of c_i be $[t_i^s, t_i^f]$. Rescheduling of id_i with respect to $OF_{\Delta t, IS_j}$ will reduce the storage requirement over the interval $[\max(t_s, t_i^s), \min(t_f, t_i^f + \rho_{id_i})]$, where ρ_{id_i} is the playback length of file id_i. As explained in Section 3.2, every caching interval is followed by the playack duration which also requires storage space. Let ΔS be the amortized time-space product which can be improved by rescheduling a file id_i with respect to $OF_{\Delta t, IS_j}$. Under the continuous time domain, ΔS can be formulated as:

$$\Delta S_{\Delta t, IS_j, c_i} = \int_{\max(t_s, t_i^s)}^{\min(t_f, t_i^f + \rho_{id_i})} f_{c_i}(t)\, dt \tag{7}$$

$$f_{c_i}(t) = \begin{cases} \gamma \cdot size_{id_i} & \text{if } t < t_i^f \\ \gamma \cdot size_{id_i}\left(1 - \dfrac{t - t_i^f}{\rho_{id_i}}\right) & \text{otherwise} \end{cases} \tag{8}$$

$$\gamma = \begin{cases} 1 & \text{if } t_i^f - t_i^s \geq \rho_{id_i} \\ \dfrac{t_i^f - t_i^s}{\rho_{id_i}} & \text{otherwise} \end{cases} \tag{9}$$

$f_{c_i}(t)$ in Eq. (8) computes the storage space requirement at each time t based on the storage cost model in Section 3.2. The maximum amount of disk space required for c_i depends on whether c_i is a *long residency* or a *short residency*. γ in Eq. (9) is a coefficient adjusting the maximum space requirement according to Eqs. (2) and (3). We compare four different metrics for *heat* of rescheduling id_i with respect to $OF_{\Delta t, IS_j}$.

$$\mathcal{H}_{\Delta t, IS_j, c_i} = \min\bigl(t_f, t_i^f + \rho_{id_i}\bigr) - \max\bigl(t_s, t_i^s\bigr) \tag{10}$$

$$\mathcal{H}_{\Delta t, IS_j, c_i} = \frac{\min\bigl(t_f, t_i^f + \rho_{id_i}\bigr) - \max\bigl(t_s, t_i^s\bigr)}{\Psi\bigl(\mathcal{S}_{id_i}^{new}(\Delta t, IS_j)\bigr) - \Psi\bigl(\mathcal{S}_{id_i}\bigr)} \tag{11}$$

$$\mathcal{H}_{\Delta t, IS_j, c_i} = \Delta S_{\Delta t, IS_j, c_i} \tag{12}$$

$$\mathcal{H}_{\Delta t, IS_j, c_i} = \frac{\Delta S_{\Delta t, IS_j, c_i}}{\Psi\bigl(\mathcal{S}_{id_i}^{new}(\Delta t, IS_j)\bigr) - \Psi\bigl(\mathcal{S}_{id_i}\bigr)} \tag{13}$$

The meaning of *heat* is *per cost* improvement obtained as a result of rescheduling id_i with respect to $OF_{\Delta t, IS_j}$. In Eq. (10) the effective improvement is measured in terms of the length of the improved time interval. In Eq. (11), the length of improved period per overhead cost is considered as *heat*. In Eqs. (12) and (13), the improvement of amortized time-space product, and the improvement of amortized time-space product *per overhead cost* is considered as *heat* of rescheduling. In the iteration of lines 7–18 of *SORP_solve*( ) algorithm, the file with the largest *heat* is selected as the victim. According to our experiments, Eq. (13) performs best, i.e., generates the least expensive schedule, on the average. Details of the experimental results are provided in Section 6.

5.4. Reflective greedy heuristic

The rescheduling algorithm *reflective_greedy*( ) in line 11 of Table 3 receives a tuple (c_i, Δt, IS_j) as an input. It reschedules the file id_i of c_i with the constraint that there is no space available during Δt at IS_j. Rescheduling a file means rearranging the service delivery of all requests for the file. Different from the *greedy heuristic* in individual video scheduling, *Reflective Greedy* maintains the space usage information for the intermediate storages, and does not schedule a video file to the intermediate storage if sufficient storage capacity is not available. This is to avoid generating a subsequent overflow situation as a result of rescheduling. The *reflective greedy* algorithm with input (c_i, Δt, IS_j) generates a service schedule for file id_i with the constraint that it is not allowed to be cached at IS_j during Δt.

6. Experiment and results

In this section, we present performance results obtained from simulations under various conditions. Before describing our experimental results, we first discuss some general information about the experiment, including the performance baseline, performance metrics of interest, and parameter settings.

6.1. General experiment information

Figure 10 illustrates the topology and connection layout of the video warehouse and intermediate storages for our experiments. There are 20 nodes in total—one video warehouse and 19 intermediate storages. Each intermediate storage supports the users in its respective neighborhood. For notational simplicity, the users are omitted in the graph representation in figure 10. In our experiment, the number of users in each neighborhood is 10. Various factors can determine the characteristics of the video service environment. Among them, we take into account four attributes which affect the service scheduling process and its cost—the *Storage Charging Rate*, the *Intermediate Storage Size*, *Network Charging Rate*, and the *user access pattern*. Table 4 shows the actual values of each attribute used in our experiment. The values of the *storage charging rate* and the *network charging rate* represent values in the charging system. In a practical situation, we can substitute these values for the actual charging rate of the monetary metric.

Table 4. System parameters.

Attributes	Values
Number of available video files	500
Average video file size	3.3 GB
Storage charging rate	3, 4, 5, 6, 7, 8 (/Gbyte · s)
Intermediate storage size	5, 8, 11, 14 (Giga Bytes)
Network charging rate	300, 400, 500, 600, 700, 800, 900, 1000 (/Gbyte)
Access pattern: *Zipf* distribution	$\alpha = 0.1, 0.271, 0.5, 0.7$

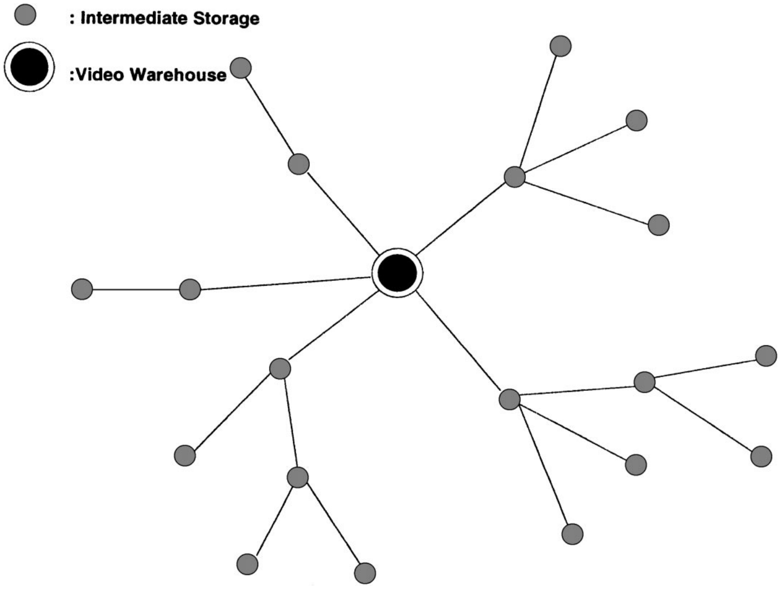


Figure 10. Graph representation of network and storage topology.

The scheduler selects the victim with the largest *heat* value. Depending on the method of computing the *heat*, which is formalized in Eqs. (10)–(13), the scheduler generates several different service schedules. We compare the efficiency of these four different policies from the aspect of total service cost.

6.2. *Experiment 1: Effect of network charging rate*

In this section, we mainly focus on analyzing the effect of network charging rate. Figure 11 shows the relationship between network charging rate and total service cost for four different storage charging rates. It also plots total service cost in the environment *without* intermediate storage. The advantage of using intermediate storage becomes more significant as the network charging rate increases. *Srate* in figure 11 means the storage charging

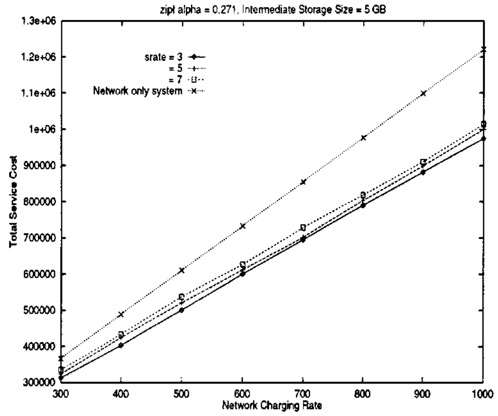


Figure 11. Under different storage charging rates.

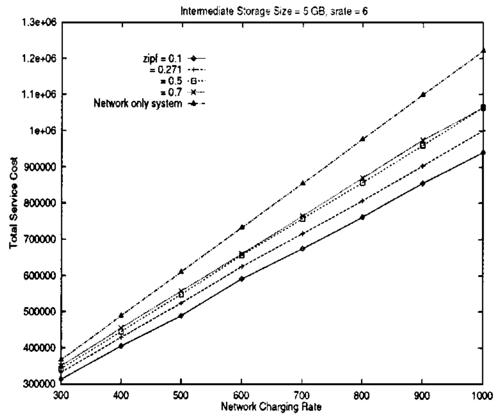


Figure 12. Under different access patterns.

rate. In figure 12, the variable is the user access pattern. In the *zipf* distribution, α determines the *skewness* of the user access pattern. Larger α implies a less biased distribution. Under the same environmental parameters, total service cost increases when the requests are more evenly distributed. The advantage of intermediate storage is to share a file between users and avoid repeated delivery of the same video file. When most of the requests are concentrated within a small set of video files, the scheduler can maximize the usage of an intermediate storage. It is observed that the total service cost increases almost linearly with the network charging rate. This is because in servicing a set of requests, there is *unavoidable* network transmission which cannot be replaced with caching the file in an intermediate storage. With a less expensive storage charging rate, the total service cost increases more slowly. In figures 13 and 14 the size of the intermediate storage is increased to 11 GB. With 11 GB intermediate storages, there is less chance of storage overflow,

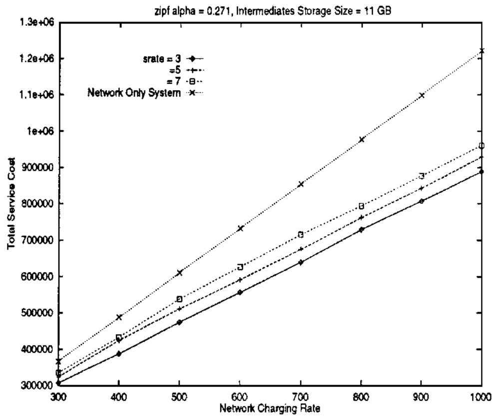


Figure 13. Under different storage charging rates.

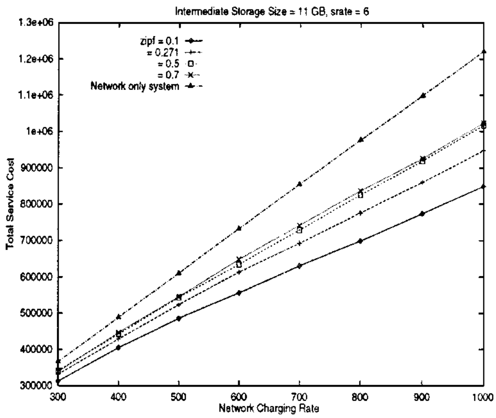


Figure 14. Under different access pattern.

and hence the total service cost can be less expensive than using 5 GB size intermediate storages.

6.3. *Experiment 2: Effect of the storage charging rate*

In this experiment, we examine the effect of storage charging rate. Figure 15 illustrates the effect of storage charging rate on the total service cost. When the storage charging rate is relatively low, the scheduler tries to avoid repeated delivery and prefers using intermediate storage to delivering the video directly from the video warehouse. The cost of using intermediate storage dominates the cost of using the network, in the situation with a low storage charging rate. Hence, the change in the storage charging rate has significant effect on total

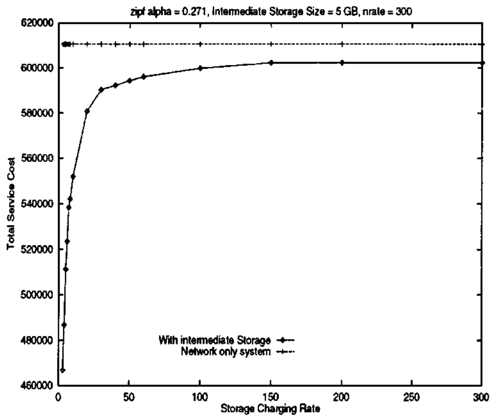


Figure 15. Storage charging rate vs. total service cost.

service cost. As the storage charging rate increases, the scheduler prefers repeated network deliveries to making the video reside in the intermediate storage for a longer period of time. Consequently, the total service cost becomes less sensitive to the increase in the storage charging rate as the storage charging rate increases. The total service cost curve in figure 15 approaches the total service cost of the *network only system* as the storage charging rate increases.

Figure 16 shows the effect of storage charging rate under different network charging rates. It is worth noting that total service cost increases linearly with the increase in network charging rate. Meanwhile, the effect of the increase in storage charging rate is substantial only when the storage charging rate is low. This phenomenon arises due to the fact that there is a substantial amount of unavoidable network delivery in the service schedule, e.g., servicing the earliest request for each neighborhood. Of course, this behavior

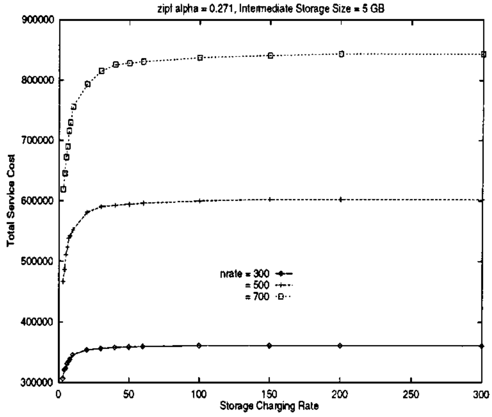


Figure 16. Storage charging rate vs. total service cost.

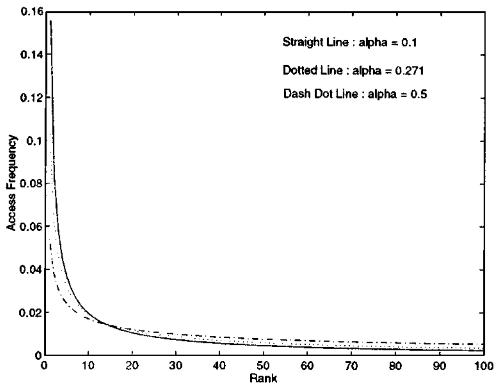


Figure 17. User access pattern with *zipf* distribution.

is not observed if we model the fact that the initial state of intermediate storage at the start of cycle $i + 1$ is the same as that at the end of cycle i. Thus, a number of videos are in various intermediate storages to begin with. We propose to do this in our future work. It is possible that none of the intermediate storage is used if the storage charging rate is expensive, and there is no mandatory requirement for using intermediate storage.

Let us examine the vertical distances between each line in figures 11 and 13. In both figures, the change in *srate* results in shifting the straight line up along the *y-axis*, with an increase in the slope of the curve. The vertical distance between each straight line, i.e., the amount of total cost increase due to the increase in the storage charging rate, is small. This implies that the cost for using intermediate storage in the service schedule is relatively small, compared to the total network cost. If the number of users in each neighborhood and/or the size of intermediate storage increases, the vertical distance between each line will become larger. This phenomenon can be observed by comparing figures 11 and 13, the size of intermediate storage is 5 and 11 GB, in figures 11 and 13, respectively.

6.4. Experiment 3: Effect of data access pattern

In this experiment, we analyze the effect of data access pattern and the total service cost. We observe that it is more advantageous to build a distributed service environment with intermediate storages as the user requests are concentrated on a small set of video files. The user request is generated using the *zipf* distribution. Figure 17 illustrates the user access pattern under various *zipf* distributions. *Zipf* distribution with $\alpha = 0.271$ approximates the commercial video rental pattern to a reasonable degree [5]. Figure 18 visualizes the effect of the data access pattern with various sizes of intermediate storage. As can be observed in figure 18, the total service cost increases as the access pattern becomes less biased. The merit of using intermediate storage is to avoid repeated file delivery to the same location. When the access pattern gets evenly distributed, the effect of using intermediate storage becomes less significant and thus total service cost increases. Let us look at the vertical distance between the three graphs in figure 18 with storage sizes 5, 8, and 11 GB,

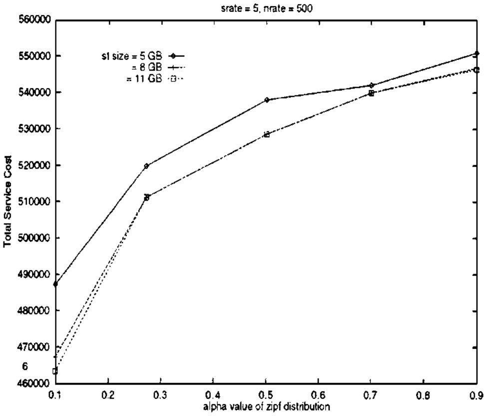


Figure 18. User access pattern vs. intermediate storage size.

respectively. The environment with the smaller size intermediate storage results in higher total service cost. As the access pattern becomes more biased, the vertical distance between the graphs becomes larger. It implies that the advantage of using larger size intermediate storage becomes more significant as user access pattern is more skewed.

6.5. Experiment 4: Effect of intermediate storage size

In this experiment, we analyze the effect of varying intermediate storage size. *Service Scheduling Algorithm* presented in this paper is based on the assumption that an intermediate storage does not have sufficient capacity to hold all the video files requested by the users in its neighborhood. This assumption is not unrealistic, considering the fact that the average size of an MPEG-2 [11] compressed video file of 110 min exceeds 3 GB. In figure 18, we show the effect of varying intermediate storage size and data access pattern. As α gets larger, i.e., the access pattern becomes more evenly distributed over the available files, the total service cost increases. This is because the schedule cannot make good use of intermediate storage if users make requests for different files. Figure 19 shows the relationship between the storage charging rate *srate* and the size of the intermediate storage. When the storage charging rate is relatively low compared to the network charging rate, there are larger numbers of storage overflows. Hence, increasing capacity at the intermediate storage is advisable to minimize service cost, but increasing the capacity beyond a certain threshold does not improve the service cost. In figure 19, it is not necessary to increase the storage capacity beyond 8 GB when $srate = 5$. As the storage charing rate increases, the benefit of using larger storage reduces. This is because the usage of the intermediate storage becomes less dominant as the storage cost increases. The analysis of this metric provides an important guideline in determining the actual size of the intermediate storage, since intermediate storage larger than a certain size does not help to reduce the service cost.

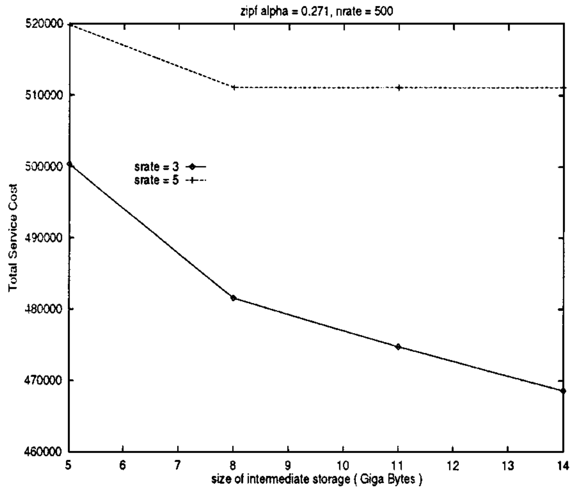


Figure 19. Intermediate storage size and *srate*.

6.6. *Experiment 5: Computing heat and cost increase*

Heat is a criteria for selecting a victim, which is equivalent to the *improvement per overhead cost* of rescheduling. *Rescheduling* should be viewed from two aspects—*cost* and *benefit*. *Cost* is the cost of rescheduling a victim, and *benefit* determines the improvement in the overflow situation. By combining these two factors, *cost* and *benefit*, we can obtain the *effective cost* of rescheduling. We provided four different ways of computing *heat* in Eqs. (10)–(13). We compare the efficiency of schedules obtained under four different *heat* metrics. In 98% of 622 different circumstances which required overflow resolution, Eq. (11) or Eq. (13) generated the best results, and thus we provide the comparison between the two. Table 5 shows the performance of each metric. The experiments were performed under 785 different combinations of *network charging rate*, *storage charging rate*, *size of the intermediate storage* and *user access pattern*. Under some situations, e.g., large intermediate storage capacity, or an expensive network charging rate, the scheduler generates an *overflow free* schedule at the individual scheduling phase. There are also situations where method 2 in Eq. (11) and method 4 in Eq. (13) generate the same output result with the same cost. Overflow resolution

Table 5. Performance of the each method.

Total number of cases	785
Service cost increase due to overflow resolution	622
Method 2 in Eq. (11)	395 out of 622 (63%)
Method 4 in Eq. (13)	437 out of 622 (70%)
Method 2 or Method 4	614 out of 622 (98%)

makes the service schedule less efficient, which leads to an increase in the service cost. In our experiments, $\frac{\Psi(\mathcal{S}^{SORP})-\Psi(\mathcal{S})}{\Psi(\mathcal{S})}$ is 12% on the average, and 34% in the worst case. Also, $\textit{find\_video\_schedule}(\mathcal{R}_i, nrate, srate)$ in Table 2 is known to generate the schedule for file i within the performance bound of 15% [20]. Empirically, the resulting schedule $\mathcal{S}^{SORP}$ is hence within a bound of 30% from the optimal solution on the average.

7. Concluding remarks and future direction

On-line digital delivery of multimedia services makes demands upon the current state of the art of computer technology. Though its application area is expanding rapidly, the high cost of service provisioning has been a serious impediment to its widespread usage. In an effort to analyze the costs of service provisioning, we have developed a model of a distributed infrastructure which has a video warehouse and intermediate storages connected via a high speed communication network. In the proposed environment, video delivery service can be provided to the user either from an intermediate storage or from a video warehouse. Unlike a non-continous media application, the *duration or bandwidth* of video service needs to be considered in measuring the resource requirements of the continuous media service requests. In this work, we developed a comprehensive cost model for storage and networks which captures the resource requirement of supporting a given set of continuous media service. We defined a *Video-On-Reservation* service where a user *reserves* a service in advance. With the combination of a *VOR* service model and a distributed environment, with a video warehouse and intermediate storages, system resources such as disk space and network bandwidth can be used in an efficient way via off-line computing and thus can accommodate more video streams and consume less storage or network resources. The algorithm presented focuses on making the best use of the *VOR* nature of requests. However, with minor modification, *Individual Video Scheduling* algorithm with storage constraints can also be applied to determine the delivery route and caching information of *VOD* requests. The *scheduler* at a video warehouse computes a schedule which determines the *file caching* and *network transfer* of the video files to service the users' requests. To provide a video delivery in a cost-effective way, the video scheduler should find a video schedule that consumes as few system resources as possible. We proposed a two stage algorithm for the video scheduler—(1) *Individual Video Scheduling*, and (2) *Storage Overflow Resolution*. With the algorithm proposed in this work, the cost of service schedule $\mathcal{S}^{SORP}$ is within 30% on the average from the optimal solution. Through extensive simulation, we analyze the effect of charging rate of resources and/or user access pattern on the total service cost. These relationships should be carefully examined in building a practical information service infrastructure where the parameters are tailored to fit particular needs. In handling storage overflow, allowing the user to specify the starting time in terms of an *interval* can help to improve the overflow situation. As future work, we plan to extend our approach to resolve the bandwidth constraints of the intermediate storages and communication network. We hope our design and algorithms can serve as useful guidelines for the design of future multimedia service provision approaches.

Appendix: Proof of NP-completeness

Theorem 1. *Cache overflow problem is NP-complete problem.*

Proof: *K-ary knapsack problem* $\propto$ *Cache overflow problem*. Hence, cache overflow is *NP* problem. □

Definition 2 (*K-ary knapsack problem*). There are a finite set of items U and K knapsacks $N_1, N_2, \ldots, N_k$ whose sizes are $B_1, B_2, \ldots, B_k$. $s(u)$ is a size of $u \in U$. $v(u)$ is a value of $u \in U$. Find the collection of set U_i's s.t. $U_i \subseteq U$, and $U_i \cap U_j = \phi\ \forall i, j$ and such that $\sum_{i=1}^{k} \sum_{u \in U_i} v(u)$ is as large as possible and $\sum_{u \in U_i} s(u) \leq B_i$.

The K-ary knapsack problem is NP-complete. Since this is a well-known problem, we will not mention the details of the K-ary knapsack problem itself.

Theorem 2. *K-ary knapsack problem* $\propto$ *Cache overflow problem.*

Proof: There exists a polynomial transformation function f that transforms the K-ary knapsack problem to the cache overflow problem.

Elements of knapsack problem

- a set of item U
- Knapsacks N_i's, $i = 1, \ldots, k$. B_i is size of N_i.
- $v(u)$: value of item $u \in U$.
- $s(u)$: size of item $u \in U$.
- **Solution**: collection of U_i's

From the elements of the knapsack problem, we can build a cache overflow problem.

Elements of the cache overflow problem

- a set of videos $= U$
- a set of video caches N_i's, $i = 1, \ldots, k$. $capacity(N_i) = B_i$.
- one video warehouse N_0
- the length of the video $u = v(u)$ minutes which determines the network transmission cost.
- the size of the video $u = s(u)$ which determines the storage cost.
- Network topology: completely connected
- Every video should be started at the same time.
- storage cost is 0 for all N_i, $i = 0, \ldots, k$.
- network cost: For each network edge in the layouts, there is a related cost for it. $nrate(i, j)$ is the cost of $edge(i, j)$ per minute. For network $edge(i, j)$,

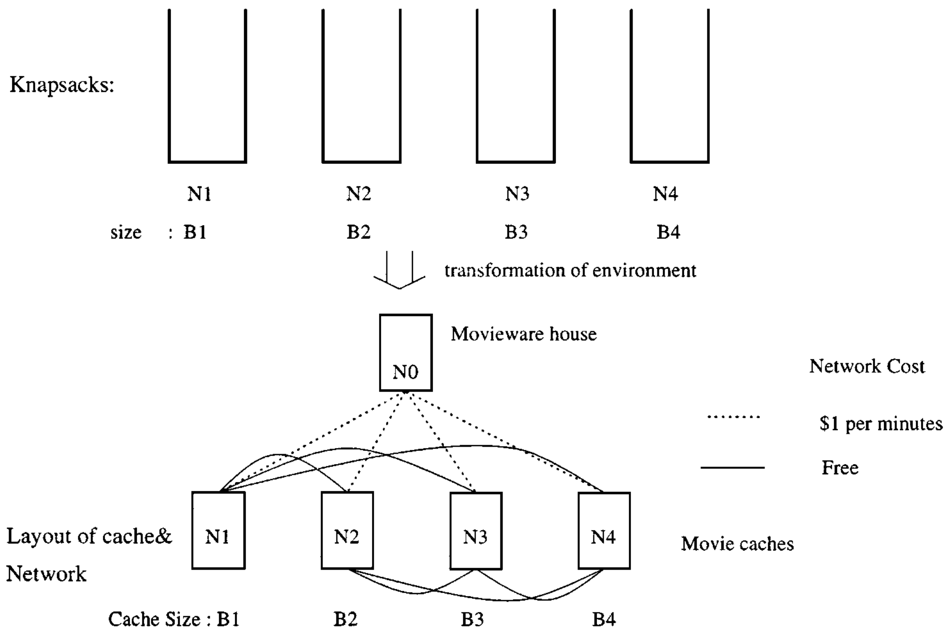


Figure 20. Transformation from K-ary knapsack problem to cache overflow problem.

— if $i = 0$ or $j = 0$, $nrate(i, j) = 1$
— otherwise, $nrate(i, j) = 0$

- *Solution*: Video schedule U_i's $i = 0, \ldots, k$ such that if $u \in U_i$, then video u should be migrated to video cache N_i for showing. If video $u \notin \bigcup_{i=1}^{k}$, then $u \in U_0$.

We can easily see that building a cache overflow problem from the K-ary knapsack problem takes polynomial time. The solution of the knapsack problem is also a solution of the cache overflow problem, and vice versa. Hence, the cache overflow problem is NP-complete. □

Lemma 1. *A solution of the K-ary knapsack problem U_i's is also a solution for the cache overflow problem U_i's $i = 0, \ldots, k$ and vice versa.*

Proof:

(1) *Knapsack problem → Cache overflow problem*: The video cache for each video is uniquely defined from $U_i, \ldots, U_k$ for all videos $u \in U$. If the resulting video schedule is not minimum, there is a video $u \in U_i$ $\exists i$ and $u' \in U_0$ such that exchange of the them will result in a smaller video delivery cost.

$$\text{video delivery cost of } u = \text{storage cost} + (\text{unit network cost}) * (\text{length of video}) \tag{A.1}$$

$$\textit{Total Delivery Cost} = \sum_{u \in U_0} v(u) \tag{A.2}$$

Hence, we can get the following relationships between the solution of the knapsack problem and the video delivery problem.

$$\textit{Total Delivery Cost} = \sum_{u \in U} v(u) - \sum_{i=1}^{k} \sum_{u \in U_i} v(u) \tag{A.3}$$

Hence, if it is possible to decrease the Total Delivery Cost by exchanging u and u', we can increase the total profit for the K-ary knapsack problem by exchanging u and u'.

(2) *Cache overflow problem* $\rightarrow$ *Knapsack problem*: It is apparent from the Eq. (16). $\square$

Acknowledgment

The work of this author was supported in part by Air Force contract number F30602-96-C-0B to Honeywell Inc., via subcontract number B0903054/AF to the University of Minnesota.

Note

1. Termed as *charging rate* of the resource in this paper.

References

1. T.E. Bell and M.J. Riezenman, "Technology 1997 analysis and forecast commucations: Communications," IEEE Spectrum, pp. 27–37, 1997.
2. G. Bianchi, R. Melen, and A. Rainoni, "Performance of video storage hierarchy in interactive video services networks," in Proc. of the 1995 IEEE Global Telecommunications Conference. Part 2 (of 3), Singapore, 1995, pp. 805–810.
3. D.W. Brubeck and L.A. Rowe, "Hierarchical storage management in a distributed VOD system," IEEE Multimedia Magazine, Vol. 3, No. 3, pp. 37–47, 1996.
4. R. Cocchi, S. Shenker, D. Estrin, and L. Zhang, "Pricing in computer networks: Motivation, formulation and example," IEEE/ACM Transactions on Networking, Vol. 1, No. 6, pp. 614–627, 1993.
5. A. Dan and D. Sitaram, "Buffer management policy for an on-demand video server," Technical Report IBM Research Report RC 19347, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY 10598, 1993.
6. A. Dan and D. Sitaram, "An online video placement policy based on bandwidth to space ratio (bsr)," in Proc. of ACM SIGMOD, 1995, pp. 376–385.
7. Y. Doganata and A. Tantawi, "Making a cost-effective video server," IEEE Multimedia, pp. 22–30, 1994.
8. S. Ghandeharizadeh and C. Shahabi, "On multimedia repositories, personal computers, and hierarchical storage," in Conf. Proc. ACM Multimedia, 1994, pp. 407–416.
9. L. De Giovanni, A.M. Langellotti, L.M. Patitucci, and L. Petrini, "Dimensioning of hierarchical storage for video on demand services," in Proc. of the 1994 IEEE International Conference on Communications, Vol. 3, New Orleans, LA, 1994.
10. T. Halfhill, "Break the bandwidth barrier," BYTE, pp. 68–80, 1996.

11. ISO/IEC JTC1/SC29 WG11/602, Coding of Moving Pictures and Associated Audio, ISO/IEC 13818-2, Committee Draft, Nov. 1993.
12. D.R. Kenchammana-Hosekote and J. Srivastava, "Scheduling continuous media on a video-on-demand server," in Proc. of International Conference on Multi-media Computing and Systems, Boston, MA, 1994. IEEE.
13. J. Kruse, "Open issues and challenges in providing quality of service guarantees in high-speed networks," Computer Communication Review, Vol. 23, No. 1, pp. 6–15, 1993.
14. T.D.C. Little and D. Venkatesh, "Prospects for interactive video-on-demand," IEEE Multimedia, Vol. 1, No. 3, pp. 14–24, 1994.
15. T.D.C. Little and D. Venkatesh, "Popularity-based assignment of movies to storage devices in a video-on-demand system," Multimedia Systems, Vol. 2, No. 6, pp. 280–287, 1995.
16. C. Papadimitriou, S. Ramanathan, and P. Rangan, "Information caching for delivery of personalized video programs on home entertainment channels," in Proc. IEEE Int'l Conf. on Multimedia Computing and Systems, Piscataway, NJ, 1994, pp. 213–223. IEEE Press.
17. P. Rangan, H. Vin, and S. Ramanathan, "Designing an on-demand multimedia service," IEEE Communication Magazine, Vol. 30, No. 7, pp. 56–65, 1992.
18. S. Shenker, D. Clark, D. Estrin, and S. Herzog, "Pricing in computer networks: Reshaping the research agenda," Computer Communication Review, Vol. 26, No. 2, pp. 19–43, 1996. ACM SIGCOMM.
19. T.F. Sienknecht, R.J. Friedrich, J.J. Martinka, and P.M. Friedenbach, "Implications of distributed data in a commercial environment on the design of hierarchical storage management," Performance Evaluation, Vol. 20, Nos. 1–3, pp. 3–25, 1994.
20. J. Sutherland and L. Litteral, "Residential video services," IEEE Communications Magazine, Vol. 30, No. 7, pp. 36–41, 1992.
21. W. Tetzlaff, M. Kienzle, and D. Sitaram, "Using a storage hierarchy in movie-on-demand servers," Technical Report, IBM T.J. Watson Research Center, Hawthorne, NY, 1994.
22. D. Towsley, "Providing quality of service in packet switched networks," Technical Report, Department of Computer Science, University of Massachesetts, Amherst, 1993.
23. Y. Wan and D. Du, "Video file allocation over disk arrays for video-on-demand," in Proc. of International Conference for Multimedia Computing and Systems, 1996.
24. Youjip Won and Jaideep Srivastava, "Distributed service paradigm of remote video retrieval request," in Proc. of 6th IEEE International Symposium of High Performance Distributed Computing, Portland, Oregon, 1997, pp. 191–200.
25. Youjip Won and Jaideep Srivastava, "Optimal configuration of storage hierarchy," in Proc. of 15th IEEE Symposium on Mass Storage System, College Park, MD, 1998.
26. Youjip Won and Jaideep Srivastava, "Stochastic analysis of storage hierarchy in massive scale data server," in Proc. of SPIE Photonics East, Multimedia Storage and Archiving Systems III, Boston, MA, 1998, pp. 308–319.

Youjip Won received the B.S. and the M.S. degree in Computer Science from the Department of Computer Science and Statistics, Seoul National University, Seoul, Korea in 1990 and 1992, respectively and a Ph.D. in Computer Science from the University of Minnesota, Minneapolis in 1997. His current research interests include multimedia systems, internet and web, database, and performance modeling and analysis. He is a member of IEEE Computer Society.

Jaideep Srivastava received the B.Tech. degree in computer science from the Indian Institute of Technology, Kanpur, India, in 1983, and the M.S. and Ph.D. degrees in Computer Science from the University of California, Berkeley, in 1985 and 1988, respectively. Since 1988 he has been on the faculty of the Computer Science Department, University of Minnesota, Minneapolis, where he is currently an Associate Professor. In 1983 he was a research engineer with Uptron Digital Systems, Lucknow, India. He has published over 110 papers in refereed journals and conferences in the areas of databases, parallel processing, artificial intelligence, and multi-media. His current research is in the ares of databases, distributed systems, and multimedia computing. He has given a number of invited talks and participated in panel discussions on these topics. He is a senior member of the IEEE Computer Society and the ACM. His professional activities have included being on various program committees, and refereeing for journals, conferences, and the NSF. Dr. Srivastava is also the Chief Technical Officer of Lancet Software, a Minneapolis based company providing products and services in data warehousing and data mining.