SCALABLE SERVICES FOR VIDEO-ON-DEMAND

Shueng-Han Gary Chan and Fouad A. Tobagi

Technical Report No. CSL-TR-98-773

December 1998

This work was supported in part by NSF under grant NCR-9016032, by Pacific Bell, and by the Center for Telecommunications at Stanford University.

Scalable Services for Video-on-Demand

Shueng-Han Gary Chan and Fouad A. Tobagi

Technical Report: CSL-TR-98-773

December 1998

Computer Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
William Gates Computer Science Building, 4-A
Stanford, CA 94305-9040

pubs@shasta.stanford.edu

Abstract

Video-on-demand (VOD) refers to video services in which users can request any video program from a server at any time. VOD has important applications in entertainment, education, information, and adverstising, such as movie-on-demand, distance learning, home shopping, interactive news, etc.

In order to provide VOD services accommodating a large number of video titles and concurrent users, a VOD system has to be scalable — scalable in storage and scalable in streaming capacity. Our goal is to design such a system with low cost, low complexity, and offering high level of service quality (in terms of, for example, user delay experienced or user loss rate).

Storage scalability is achieved by using a hierarchical storage system, in which video files are stored in tertiary libraries or jukeboxes and transferred to a secondary level (of magnetic or optical disks) for display. We address the design of such a system by specifying the required architectural parameters (the bandwidth and storage capacity in each level) and operational procedures (such as request scheduling and file replacement schemes) in order to meet certain performance goals.

Scalability in streaming capacity can be achieved by means of request batching, in which requests for a video arriving within a period of time are grouped together (i.e., "batched") and served with a single multicast stream. The goal here is to achieve the trade-off between the multicasting cost and user delay in the system. We study a number of batching schemes (in terms of user delay experienced, the number of users collected in each batch, etc.), and how system profit can be maximized given user's reneging behaviour.

Both storage and streaming scalabilities can be achieved with a distributed servers architecture, in which video files are accessed from servers distributed in a network. We examine a number of caching schemes in terms of their requirements in storage and streaming bandwidth. Given certain cost functions in storage and streaming, we address when and how much a video file should be cached in order to minimize the system cost. We show that a distributed servers architecture can achieve great cost savings while offering users low start-up delay.

Key Words and Phrases: Video-on-demand, video server, hierarchical storage systems, request batching, distributed servers architecture

Contents

Chapter 1

Introduction

Video-on-demand (VOD) refers to video services in which a user is able to request from a server any video content at any time. VOD encompasses many applications important in entertainment, education and advertising, such as movie-on-demand (MOD), news-on-demand, distance learning, home shopping, various interactive training programs, information kiosks, etc.[1, 2, 3, 4, 5, 6, 7, 8] In VOD, a user first selects a video file (such as a movie, an advertisement, or a piece of news) on a relevant video server. Once the file is displayed, the user usually can to a certain extend interact with it through fast-forward, rewind, pause, etc. However, users generally are not allowed to modify or change the contents of the files, i.e., video files are read-only.

In order to provide VOD services accommodating thousands of video titles and thousands of concurrent users, a VOD system has to be scalable – scalable in storage and scalable in streaming capacity. Our goal in this dissertation is to address these scalability issues so as to design a VOD system which offers high service quality with low cost and complexity. The service quality can be in terms of, for example, user delay experienced or user loss rate.

There are two key elements in the successful provisioning of video services: the design of the video servers and how video content is delivered to the users through networks. Previous work has mainly focused on server design, while this dissertation addresses both server and network issues in offering video services.

In this chapter, we first present in Sect. 1.1 VOD application characteristics and performance goals in server design. Then in Sect. 1.2 we briefly discuss some of the previous work in server design and point out the limitations of existing approaches. Finally in Sect. 1.3 we highlight our contributions in addressing the storage and streaming scalability in offering VOD services. The contributions are on:

- *Achieving storage scalability with the use of hierarchical storage system*, in which we have developed a model which speeds up the design process of a hierarchical storage system and solves many other storage problems of such kind;

- *Achieving streaming scalability by means of request batching and multicasting*, in which we have studied a number of batching schemes, and addressed the channel planning and profit issues in such a system; and

- *Achieving storage and streaming scalabilities with distributed servers architectures*, in which we have studied a number of caching schemes and shown the cost-advantage of such an architecture.

1.1 VOD Application Characteristics and Performance Goals in Server Design

1.1.1 Applications characteristics

In order to design a practical and useful video server, we need to understand the characteristics of VOD applications. We discuss here four applications characteristics: demand characteristics, video files characteristics, user interactions, and performance requirements.

Demand characteristics: Demand characteristics include three attributes: the requests arrival process, size of each arrival, and the request's holding time.

- Arrival process — An arrival consists of one or more simultaneous video requests, each of which asks for the display of a certain video in the database. In other words, each request demands a video stream from the video server. In a VOD environment, the arrival process may vary according to the time of day, day of week, or month of year, or be affected by external news (such as Olympic games, good or bad news around the world, or the academy awards).

  Finding a realistic arrival process with both long-term and short-term variations is generally difficult, and is often possible only by doing some field trials or experiments. The Poisson process is generally used to model the arrival process for large number of users, in which the inter-arrival time is modeled as exponentially distributed. The rate of the exponential function may vary depending on the time of day, or number of users currently seeing or waiting to see their videos. More general arrival processes can be modeled as series-parallel combination of exponential processes, such as hyperexponential distribution, Erlangian distribution, Cox-phased networks, etc [9, 10].

- Size of each arrival — The size of an arrival refers to the number of video requests in each arrival. It may be one-at-a-time or multiple requests at a time. The traffic of most applications are characterized by one-at-a-time arrival. Multiple requests per arrival is also known as "bulk arrival." Some applications have bulk arrivals, such as distance learning or multiuser training programs, in which multiple users may arrive at more or less the same time.

- Request's holding time — Request holding time is the total time a user occupies a video stream, for either seeing or interacting with the video. Depending on the particular application, the request holding time may be somewhat random (e.g., as in interactive news), or relatively deterministic (e.g., as in movie-on-demand).

Video files characteristics: The file-related characteristics of a VOD application include streaming bandwidth, size of the files, number of video titles, and video popularity.

- Streaming bandwidth — The streaming bandwidth of a video, b_0, depends on the video compression scheme used (e.g., MPEG-I, MPEG-II, motion-JPEG, etc.). It can range from less than 1 Mbps to more than 10 Mbps. Streaming bandwidth also depends on the encoding method used (e.g., Constant-bit-rate, Variable-bit-rate, etc.).

- Size of the video files — The size of a video file is the actual storage space the file consumes in a storage medium. It may range from ~10 MB (advertising clips) to more than ~1 GB (movies). All files in a VOD application may not be of the same size. In MOD (movie-on-demand), for example, the file size is likely to be similar or "homogeneous," with each file of about, say, 90 minutes playback time. On the other hand, file size in interactive news environment can be rather "heterogeneous," depending on the piece of news and whether it is a documentary or not. Somewhere between the "extremes" may be home shopping, in which file size may range from ~5 MB to ~30 MB (20 seconds to 2 minutes).

  Note that the size of a video file does not necessarily relate to the request's holding time. This is because users may finish their displays at any time. Furthermore, some parts of the video can be repeated many times, other parts may be skipped (e.g., by fast forwarding or jump command), and the user may pause at any time.

- Number of video titles — Applications targeted to general public are likely to have more titles than applications for a smaller group of users.

- Video popularity — Different videos have different access frequency. The popularity of a video is defined as the probability for the video to be accessed or chosen by any incoming request. (The popularity index of a video is proportional to the video popularity.)

  If all of the video titles are equally likely to be chosen by an incoming request, the video popularity is uniform. Non-uniform video popularity is commonly modeled using Zipf or geometric distribution. To explain these models, we first arrange the popularity of all N_v videos in the system in decreasing order. Let the popularity of the ith video be p_i, where $1 \leq i \leq N_v$.

- Zipf distribution [11, 12, 13]: In this model, the popularity of the ith video is proportional to $1/i^{\zeta}$, where ζ is a (real) parameter known as Zipf parameter. Most models in the literature take ζ to be 1.0, though according to some actual rental data, $\zeta = 0.271$ may be more likely [12]. Given ζ, we can then express p_i as:

$$p_i = C/i^{\zeta}, \tag{1.1}$$

  where

$$C = \left(\sum_{i=1}^{N_v} 1/i^{\zeta} \right)^{-1}. \tag{1.2}$$

- Geometric distribution [14, 15]: In this model, the ratio of p_i to p_{i-1} is equal to a constant, the "skew parameter" σ $(0 \leq \sigma \leq 1)$. Since $p_i/p_{i-1} = \sigma$, we can express p_i, where $1 \leq i \leq N_v$, as

$$p_i = \frac{(1-\sigma)}{(1-\sigma^{N_v})} \sigma^{i-1}. \tag{1.3}$$

User interactions: In VOD, after a video is displayed, the user may be able to interact with the video sequence. We describe here four attributes of user interactions: i) modifiability of video files, ii) the types of interactions; iii) the frequency of interactions; and iv) the locality of interaction.

i) Modifiability of video files — An application may not allow the general users to modify its video files. These read-only applications encompass most of VOD applications, such as movie-on-demand, home-shopping, interactive news, etc. As the users are not allowed to edit the files, there is no write-back traffic due to file modification. On the other hand, in a video editing environment, video files are constantly being updated or created. Video servers for such purpose may have to be designed differently, because of the issues in write-back traffic and placement of data-blocks.

ii) Types of interactions — Types of interactions refers to the user commands in interacting with the displaying video. Different VOD applications have different types of interactions. In MOD (movie-on-demand) or distance learning, for example, users would most likely interact with the videos using VCR commands such as FF (fast-forward) and RW (rewind). In home-shopping, point-and-select interactive commands will most likely be characteristic. In video-authoring environment, functions similar to "cut and paste" may have to provided.

iii) Frequency of interactions — Interaction frequency refers to how often a user interacts with the video displayed. Different applications may vary markedly in terms of the interaction frequency. For example, the frequency in MOD would be lower (possibly ≤ 0.1 interaction/min.) than that of home shopping (~ 1 interaction/min.).

iv) Locality of interaction — Locality of interaction refers to the time displacement in a video sequence between an interaction command and the video display-point prior to the interaction. Certain video, such as a movie or a lecture, is displayed in a certain sequence if the user does not interact with it. Once a user starts to interact with the video, the sequence is changed and the video is playbacked at a different point in the video length. An interaction is said to be "localized" if such new display point is temporally close to the point prior to the interaction. Some applications, such as MOD, are most likely characterized by localized interactions, while some others (e.g., interactive news and home-shopping) are expected to exhibit low locality of interaction.

Performance requirements: Different applications have different performance requirements. We list here five such requirements: i) start-up delay, ii) user interactions, iii) streaming capacity, iv) low loss/blocking probability and v) others.

i) Start-up delay — We define "start-up delay," D_{st}, as the waiting time from the moment when a user initially submits a video request until the moment when the user begins to see the video. It is therefore the total waiting time before the requested video is streamed. Obviously, D_{st} is a random variable whose value depends on where the user is in the queue, what the user's priority class is, or even the video requested. We distinguish here start-up delay from the response time of user interactions. While start-up delay is the waiting time for a user *before* the requested video is displayed, the response time of user interactions is the latency from the time of issuing a control command to the actual scene change in an *on-going* video session. Therefore, start-up delay can be longer than the response time of user interactions.

For performance evaluation, one sometimes would consider the average start-up delay over all requests in the steady state, $\overline{D}_{st}$, or for a certain request class i, $\overline{D}_{st}^{(i)}$. A bounded start-up delay or more deterministic delay (in which users are similarly delayed) may also be of interest.

Different VOD applications have different minimum start-up delay requirements. The requirement may depend on how long a user sees the video, i.e., the request's holding time. For example, in MOD in which the holding time is relatively long ($\simeq$ 90 min.), start-up delay as high as several minutes is acceptable. However, when a user is still not sure what file to be displayed and wants some kind of "preview" before making up his/her mind, the start-up delay for these "preview" video clips should be much lower, possibly in the range of seconds. For home-shopping or news-on-demand applications, on the other hand, the start-up delay should be bounded to within several seconds.

While a long start-up delay is undesirable, generally users are willing to wait longer under the following conditions:

- Delay guarantee: Users may be more willing to wait if they are sure that they can watch their videos at a particular time, even if the time is possibly minutes (or even hours) later. This is the principle behind delay guarantee systems, such as deterministic delay (in which users experience similar delay) or reservation system, in which users reserve videos to be displayed at a certain later time.

  - Variance reduction and delay estimation: Generally, users are more willing to wait if they know how long they have to wait, i.e., when the uncertainty in waiting time is reduced. Providing a good estimate of how long a user would wait before the display of his/her video is therefore valuable.

  - Delay discount service: Users in VOD are usually willing to pay for the quality of service they receive, and hence they are more willing to wait if they can enjoy some kinds of delay-dependent charges.

ii) User interactions — We list two such requirements here: response time of the interactions and control granularity of the interactions.

  - Response time of the interactions: The time elapsed between a user's interactive command (e.g., FF, RW, etc.) and the actual change of the display scene is called the response time of the interaction. Different applications have different requirement in the response time. For interactive news, the response time should be rather low (in the range of a second or so) while for MOD, the response time requirement can be more relaxed.

  - Control granularity of user's interactions: In some VOD applications, especially for movie-on-demand or distance learning, the points at which users are allowed to visit in the video sequence do not have to be exact. Such a coarse interaction "granularity" may not be acceptable for some applications (such as home-shopping). A system with very fine interaction granularity allows users to go to or pause at virtually any time, while a system with coarse granularity gates the users to interact at some fixed, specified points in time.

iii) Streaming capacity — Streaming capacity is the maximum number of concurrent users or the maximum request rate that a server can handle. Different applications have different capacity requirement. A large VOD application, such as movie-on-demand, may have hundreds or thousands of concurrent users, while a smaller local VOD applications (for company training purpose) would have fewer users, e.g., 10–100 concurrent users.

iv) Low loss/blocking probability — If an incoming video request cannot be served immediately, the request is blocked. The blocking may be due to a lack of resources (such as bandwidth or storage space). A blocked request may be rejected for service, and hence is lost on arrival. A blocked request may also be put into a queue for later service. Users may leave the system (i.e., renege) while they are waiting, and hence are lost. Blocked calls or lost calls leads to service dissatisfaction, and frustrated users may never visit the system again. A low loss/blocking rate is hence essential in providing high quality of service to the users.

v) Others — A VOD system should offer acceptable video quality. Different services may require different video quality depending on the class of the users, application, etc. Furthermore, the scheduling policies used in a server should be fair. For example, in movie-on-demand, a user who happens to request an unpopular movie should not be discriminated in favor of a user requesting a more popular movie, if both of them are charged the same.

1.1.2 Performance goals in video server design

We state here four performance goals in designing a video server:

Meeting the applications performance requirements: Needless to say, the primary goal of a video server design is to meet the specific applications requirements stated above. Such requirements include start-up delay, user interactions, streaming capacity, etc., as stated above.

Cost-effectiveness: The design of a video server involves trade-offs among many system parameters, such as storage capacity and bandwidth. A video server should satisfy specific application requirements with the lowest cost, i.e., it should be *cost-effective*. The cost of a VOD server consists of many components such as storage cost (e.g., disks or tapes used), bandwidth/communication cost (e.g., disks/tapes drives, bus bandwidth, etc.), software cost (e.g., the cost of software development), the cost of the associated peripherals (e.g, interface units or other hardware), etc. Optimal server design should minimize the cost while meeting the performance requirements.

Scalability: A server should be designed so that it can expand easily to accommodate more users or video titles.

Ease of design and implementation: A server should not be too complex to design or implement.

1.2 Prior Work in Server Design

Here we reviewed prior work pertaining to video server design. Most of the work focus on video servers based on magnetic disks for video storage and streaming. To increase the storage capacity and streaming bandwidth, more magnetic disks are added in the system. Such a system hence does not scale well to large volume of storage in excess of terabytes. In the cases that the server can support terabytes of storage, they are still too expensive and complex to be used in video-on-demand. We will show how the scalability issue can be economically addressed with the use of hierarchical storage systems in Chapter 2.

Disk array, also known as redundant array of inexpensive disks (RAID), plays important role in high performance server system. It was originally proposed by Patterson et. al. in [16, 17]. Since the disks are configured as a unit in RAID, failure in a disk would affect the whole array of disks. The paper talks about how to increase the reliability by introducing reduncdant data in the system, and discusses the bandwidth issues during R/W and rebuild processes. The strengths and weaknesses of different RAID levels in terms of their reliabilities, R/W performance, data redundancy, etc. are further examined in [18, 19, 20]. Ng in [21] addresses the effect of the stripe size (on system response time and throughput performance) and spindle synchronization (on buffer size) in RAID.

Tetzlaff and Flynn review many commercially available video servers or servers under development in terms of their switch architectures (e.g., IBM Tiger Shark server, Oracle/N-Cube media server, Microsoft Tiger, etc.) [22]. All systems are based on magnetic disks, and need more disks or processing nodes when more storage or bandwidth is needed. We briefly describe them as follows.

- Microsoft Tiger Video Fileserver [23] — The Microsoft Tiger is a distributed, fault-tolerant real-time file server. In the system, a number of computers ("cubs") with the same type and configuration of disks are connected together by a high speed network. Data is striped across all disks and all computers, and is retrieved in a round-robin fashion according to a slotted schedule (disks walk down the slotted schedule in a round-robin fashion). There is a central controller serving as the contact point for the clients, as the system clock master, and as the book-keeper in the system. Reliability in the system is achieved by data mirroring, in which a block of data having its primary copy in a disk has its mirrored data declustered among a number of disks. The measurement in terms of start-up latency with respect to the number of active streams is presented in the paper.

  Similar idea has been applied in server array, in which multiple servers are operated in parallel to increase the system throughput (storage and streaming capacity are hence increased by putting more servers in the system). The development of such a system is discussed in [24, 25, 26], in which video data is striped across a number of video servers and retrieved synchronously from the servers, which push the data concurrently to the clients.

- Oracle nCube [27, 28, 29, 30, 31] — NCube is a massively parallel processing super-computer based on hypercube architecture, scaling up to terabytes of storage with thousands of disks. The video server is technologically more advanced and sophisticated than the above, with thousands of interconnected processing nodes (each one consisting of its own memory, I/O channels, caches and I/O channels), potentially offering an aggregate I/O rate of more than hundreds GB/s. Oracle media server is a multimedia server (handling both data and continuous medium) built on top of the nCube platform, and hence able to serve a large number of concurrent users. A connectionless client-server communication protocol is used, and reliability of data transmission is implemented using positive acknowledgment with retransmission and timeout.

- IBM Tiger Shark Video Server [32, 33] — The IBM Tiger Shark server system consists of an authoring system, a video streaming unit capable of supporting hundreds of concurrent users, and set-top boxes. In the system, user's control signals are transmitted using the X.25 protocol, and video files are delivered through high-speed DS-1 lines. To increase the system throughput, the server uses large disk blocks (256 KB) which are striped across a number of disks. These servers have been used in field trials by Bell Atlantic, HK Telecom, Cox Cable, and in Japan.

- Starlight's StarWork$^{\mathrm{TM}}$ [34, 35] — StarWork$^{\mathrm{TM}}$ is a digital video networking software capable of supporting a wide range of video applications and tens to hundreds of concurrent users. It uses software striping in which no special RAID hardware is required. Central to it is the Streaming RAID$^{\mathrm{TM}}$, a disk access scheduling software which stripes video data on disks and schedules I/O requests. In [35], several I/O service disciplines (FCFS, SCAN, Grouped sweeping) and their implications in terms of memory requirements and user delay are discussed.

- HP AutoRAID [36] — HP AutoRAID is a hierarchical disk-based storage system with two RAID levels: disk mirroring and RAID level 5. Popular files are mirrored for high throughput, while not so popular ones are economically stored as RAID 5. Though "hierarchical" in nature, there is no data duplication between the two levels.

Data is transparently and dynamically migrated between the two levels as its data access pattern and disks utilization change.

Besides the above implementation/development efforts, much research has also been conducted on disk-based video servers, which would be used as one of the components in our design. We briefly review some of the work here.

The design issues of a multimedia storage server have been overviewed in [37, 38, 39, 40], including real-time request scheduling, buffer management, data placement on disks, data striping, reliability, etc. Gemmell et. al. give some fundamental conditions and bounds in the buffer size used in order to guarantee continuous retrieval of data in [41, 42], for certain retrieval scenarios. The features of a real-time operating system, such as request scheduling, resource allocation and management, and memory management, are reviewed in [43]. Ramakrishnan describes in [44] how various tasks with diverse performance requirements can be accommodated in a server (e.g., data, real-time file access with different rate or quality requirement).

Barnett and Anido in [45] compare the cost of RAID3 and RAID5 in providing video services. Two types of costs are considered: i) (fixed) system set-up cost (to meet a certain streaming capacity); and ii) maintaining cost in case disks fail in the system (hence leading to loss of revenue), for which they formulate as a Markov reward model. They show that in general RAID5 achieves lower cost than RAID3. Vin et. al. study a predictive admission policy for multimedia servers, in which a request is admitted only if the extrapolation from the past server performance indicates that there may be enough resources available for the request [46]. Mourad and Dan et. al. study in [47, 48] trade-off issues between memory and I/O bandwidth. In their schemes, video data is cached in memory for a certain amount of time and hence future requests may retrieve the data directly from the memory, thus relieving the load from the disk. Kunii et. al. and Vin et. al. in [49, 50] study how data block and inter-block gap should be laid out on disks for efficient real-time access.

1.3 Highlights of Contributions

In this thesis, we address the provisioning of VOD services with scalable storage and streaming capacities. We first study how storage scalability can be achieved by means of a hierarchical storage system, and then study how streaming scalability can be achieved by means of request batching and multicasting. Both storage and streaming scalabilities can be amplified in a distributed servers environment. We elaborate more on these contributions in the subsections.

1.3.1 Hierarchical storage systems

We see from Sect. 1.2 that previous work in server design mainly focuses on systems based on magnetic disks only. Magnetic disk technology is used due to its high throughput, low access latency, random data access, and adequate storage capacity. However, magnetic disks are still not ideal to store large volume of video files (e.g., $>$ 1000s video files for some applications) because of reliability and scalability issues. Furthermore, magnetic disks nowadays still suffer comparatively high cost — currently, storing a single 90-minutes movie ($\simeq$ 1 GB) in such a medium can cost more than a hundred dollars. As video files can also differ markedly in their popularities, some of them can be infrequently accessed. Storing all files on-line regardless of their popularities is therefore not very cost-effective.

Video servers based on hierarchical storage systems provides a cost-effective solution. They consist of both tertiary and secondary levels. Tertiary storage, commonly referred to as library or “jukebox,” is used to store all video files. The files are transferred or “staged” on demand into the secondary level to be displayed. The secondary level in the hierarchical storage system therefore acts as a “caching” or “staging” platform for video display. As the media cost of tertiary storage can be orders of magnitude lower than that of secondary storage, storing videos in the tertiary level is much cheaper. Furthermore, as a tertiary library can shelve hundreds or even thousands of removable tapes or disks, with each tape or disk offering high storage capacity ($\sim$10 – 100 GB), a tertiary system holds large storage capacity in the range of terabytes. Therefore, such a hierarchical storage system offers low cost high capacity storage as compared with a system based on magnetic disks only.

In Chapter 2, we consider using a hierarchical storage system for video-on-demand so as to achieve storage scalability. Our objective is to design such a storage system to meet certain performance goals, taking into account the applications characteristics (such as file sizes and popularity, the requirements in user interactivities like fast-forward (FF), rewind (RW), pause, etc). The design of a server based on a hierarchical storage system actually involves many architectural parameters such as secondary level bandwidth, secondary level storage capacity, tertiary level bandwidth, number of drives, etc., and many operational procedures including admission control, request scheduling and replacement policies.

Previous work on hierarchical storage systems focuses on its operating schemes, justifying its cost advantage over servers based on using magnetic disks only, and use request rejection rate as a performance measure (the previous work will be reviewed in the chapter). Our work differs from them in studying, given such cost advantages, how system parameters (bandwidth, storage and number of tertiary drives) should be dimensioned so as to meet a certain performance objective. Our interest is in user start-up delay. We address the design problem using both simulation and analysis. We have developed a simple model which leads to efficient design of such a system. The model can be extended to cover various operating schemes for different levels of user interactive capability, and a distributed storage system of the same nature. We have also addressed the influence of non-uniform video popularity on system requirements.

1.3.2 Request batching and multicasting in near video-on-demand

In pure-VOD, each user is assigned its own dedicated unicast stream. Hence users enjoy great flexibility in interacting with the server while viewing their videos. However, pure-VOD does not scale up well with the user population and becomes very expensive when a large number of concurrent requests have to be accommodated.

In Chapter 3, we address the scalability in streaming capacity. Note that when the video content is popular, the use of a single multicast stream serving many users simultaneously becomes more cost-effective. This is accomplished by grouping (i.e., batching) many requests for a given content arriving over a period of time and serving them with a single multicast stream. This is referred to as near-VOD. Today, request batching is practically used in many satellite and cable network-based movie-on-demand services. However, the bandwidth saving as compared to pure-VOD is achieved at the expense of user start-up delay. Since each user no longer has a dedicated stream, such technique is effective when user interactivity does not have to be flexible or is not essential, as is the case with such applications as movie-on-demand.

Three components constitute a near video-on-demand system:

- Video servers — The video servers store a number of movies (characterized by their duration, popularity and streaming data rate) accessible by the users. Each server has finite storage and streaming capacities. Such resources are considered to be always available and in a sense already paid for. The available streaming capacity may be partitioned or shared among the movies. In a near VOD system, the main issue is to appropriately assign the limited streaming capacity to the various requests by means of batching.

- Network — The network is considered to offer the multicast channels needed. In a near VOD system, there may be a certain number of channels that are leased by the service provider and always available (already paid for), in which case the issue is the same as for the streaming capacity of the server. On the other hand, multicast channels may be requested by the near VOD service provider on-demand at some cost. This is the case when, for example, satellite channels are used or some tolls are to be paid in order to use a multicast stream. In this case, the issue from the service provider's point of view is to amortize the cost of the channels and guarantee a certain level of profit.

- Users — The users make requests to view certain movies. These requests are characterized by the stochastic process representing the arrival in time of a request, which depends on a number of factors (the time of the day, the occurrence of some news/events, etc.), and the choice of movies, which depends on the movie's popularity (the probability of selecting the movie). What is important for the users is the waiting time, the time from when the user places a request until the movie display is started. Depending on the waiting time incurred, a user may cancel its request and leave the system (i.e., renege). The reneging behavior of the users is an important consideration in the design of a near VOD system and the underlying request batching schemes. Many user reneging models have been considered in the literature, in which the time a user is willing to wait before reneging is distributed according to some cumulative distribution functions; namely, an exponential function, a truncated Gaussian function, or a linear function (uniform distribution). However, in practice, there is no real data on the user's reneging behavior, and the use of any specific model has been either arbitrary, or driven by the need to keep the analysis tractable. In the absence of any information about user's reneging behavior, it may be appropriate to use a simple model in which users are willing to wait for a certain amount of time, beyond which they would not be satisfied and may be considered reneging with very high probability (i.e., reneging function is a step function with a delay limit). In some situations, the delay limit may be a design choice that the service provider makes and advertises for the service. A batching scheme in this case therefore should be designed so that user's delay is bounded by this time.

Previous work on near video-on-demand concentrated on the streaming capacity of the servers (which is given and already paid for), and therefore addressed the issue of how the available server channels are to be assigned to requests so as to achieve maximum throughput; more specifically, they considered a certain design goal pertaining to the loss of requests such as minimizing the loss rate, guaranteeing that loss is uniform across all movies, or some trade-off between the two. These studies would be applicable to the case where network channels are leased and therefore are available (in limited number) and paid for prior to service offering (previous work on nVOD will be reviewed in Chapter 3).

We consider a near VOD system in which the network channels are acquired on-demand. Associated with the use of each channel is a certain cost. Under this circumstance, request loss rate may no longer be the only measure of performance, and minimizing loss rate may no longer be the only design goal. From the service provider's point of view, profit is an important consideration; and maximizing profit while offering acceptable user delay (and by the same token acceptable request loss rate) becomes the important design goal. In Chapter 3, we primarily consider the case in which the number of on-demand channels that can be acquired is unlimited. Accordingly, the servicing of requests pertaining to a given movie is independent of the servicing of requests for other movies, and hence it is sufficient to consider the single movie case. We do also consider the case in which the number of available on-demand channels for a given movie is limited and study the effect of such limitation on the system design and performance.

We first examine two well-known basic batching schemes, namely, the window-size based schemes and the batch-size based scheme. We first analyze these schemes under the condition that users do not renege and compare them in terms of the number of required concurrent channels (also referred to as streams), the number of users served per stream (which translates to revenue per stream), and the delay experienced by a user. In the window-size based batching schemes, a maximum user delay is guaranteed. This maximum delay is equal to the batching window size; conversely, if the user reneging behavior is a step function, then the window-size based schemes with the window size equal to the delay limit would lead to no user loss and thus maximum throughput. Obviously this also corresponds to maximum profit. With the batch-size based scheme, per stream revenue (and thus per stream profit) can be guaranteed. The performance characteristics of these schemes without user reneging is useful to design a system with a certain delay or profit objective. For example, providers may want to provide a service in which the probability of the user delay exceeding a certain value $\hat{d}$ is very low.

We then introduce a new adaptive scheme which combines the key advantage of the window-size based schemes (namely guaranteed delay) and the advantage of the batch-size based scheme (namely guaranteed per stream revenue) by ensuring that when the arrival rate is sufficiently high (and hence profit can be easily achieved), the system guarantees fairly low delay D_{min} to the users by batching them according to a window size equal to D_{min} (for service competitiveness); but when arrival rate is not so high, the system guarantees a certain level of profit as long as users' delay does not exceed a certain bound D_{max}. The scheme therefore balances service quality (in terms of the user delay experienced) and system profit adaptively.

We then examine the design of near VOD systems with user reneging, driven by the goal to maximize the profit rate (which also corresponds to the profit achieved over a long period of time). System profit depends on the length of the batching period: if the period is too short, many network channels would be used with a few users per channel leading to high usage cost; as the period increases, the batch size increases, but due to the reneging behavior of the users, the batch size reaches a limit; thus if the batching period gets too long, the profit rate decreases due to lost opportunities. We find the optimum values for the parameters of the batching schemes to maximize the profit rate.

1.3.3 Distributed servers architecture

In Chapter 3, we discussed the use of request batching and multicast channels to limit the streaming bandwidth required (as well as the communication cost) but at the expense of user delay. Here we look at decreasing such delay by means of streaming servers which cache the requested movies locally and stream them to the users. This has the advantage of providing zero start-up delay to the users. This advantage, however, comes with the cost of additional servers and storage. It is important to note that depending on the acceptable user delay associated with request batching, it is possible that the total cost of a distributed servers architecture be yet lower.

It is considered that a number of repository servers exist which store all the video contents of interest to a large pool of geographically distributed users. To achieve large and scalable storage, repository servers may be tertiary libraries or jukeboxes which store videos of a wide range of popularity. If users were to stream their videos directly from the repository servers, then the number of users that could be served would be limited by the streaming capacity available at those servers.

Streaming capacity can be increased by using a hierarchy of servers as mentioned in Chapter 2, in which multiple streaming servers get the requested movies from the libraries, and cache locally the movies and then stream them to the users. This in fact may introduce some delay for movies of low popularity. If the streaming servers are co-located with the central repository, the cost of the long distance channels needed to stream the requested videos to the remote users may be high (e.g., through the use of some satellite links or long-haul transmission lines). Therefore, streaming videos in this way to the users would not be cost-effective.

To overcome the above limitation in channel usage cost and taking advantage of locality of usage, the streaming servers may be placed closer to the users (or clusters thereof), thus forming a distributed servers architecture. A number of repository servers (collectively referred to as a central repository) storing all the video titles deliver the video to the local servers through a communication network. The local servers cache movies locally, and hence multiple requests for a movie may be served from the local cache rather than from the central repository. In this way, the bandwidth requirement in the repository and channel usage cost can both be reduced. Such a system in fact has been discussed previously in the literature. By putting more repository servers and local servers, the system offers scalable storage and streaming capacities.

We consider that there is a cost associated with storing a movie in a local server depending on how much and how long the storage is used for. We also consider that there is a cost associated with a central server using a network channel to stream a movie, which depends on the distance and the type of network, e.g., through the use of internet, ATM, or more expensive satellite channels. Such network channels more often provide multicast capability.

Accordingly, there is a trade-off between the cost of storing a movie locally and the cost of using expensive channels. It would pay to store a movie locally if the storage cost can be offset by a big saving in channel cost: if the demand for the movie is high, we should not request it directly from the repository since it would incur too often the expensive long-haul channel cost (we hence should store the movie locally); on the other hand, if the demand is low, we should not store the movie locally since it would cause too high a storage cost (we should hence request it directly from the repository servers).

Indeed, not all movies have identical popularity, some are popular while some are not, and the popularity may change over time. There is hence a need to decide what to store locally given the local request rates. It is important that such a decision be continuously made over time to take into consideration the dynamic nature of video popularity. In other words, the two systems (i.e., the central repository and local servers) should not work independently and in parallel serving their respective users; instead, there should be constant exchange of movies between the two. For example, the introduction of a recent lecture, or a new or hot movie title can upset the popularity of some movies and make some no longer worth storing locally.

A local server may not have global information about the video contents of the other servers in the network. This may be due to, for example, limited processing capability (for constant and frequent content exchanges and updates), network limitation (e.g., disjoint networks or limited network capacity for frequent updates), or lack of incentive to exchange content information (e.g., due to un-cooperating service providers or security issues). In this case, the local servers can only request their data from the repository and they operate independently from each other. For the case of a cable TV system, multiple head-end servers may serve their local communities through coaxial local drops. The servers operate independently from each other (since they may be run by different providers). They get their movies from the central repository through the use of unicast channels (via a satellite system or through some low-cost channels such as internet). A certain number of network channels may be leased by the video content service providers and always available (already paid for), in which case the issue is the same as for the limited streaming capacity of the repository. Network channels may also be requested on-demand at some cost. This is the case when, for example, satellite or ATM channels are used or some tolls are to be paid in order to use a channel.

Suppose now a service provider operates a group of servers serving a number of regions. If multicast network channels are available, a movie may be delivered to the local servers using such a multicast channel in order to decrease channel requirement and save channel cost. A local server can freely join any existing multicast groups (but they do not communicate with each other). Note that the local servers may be connected through a network and are thus able to exchange their video contents with each other. Therefore, instead of getting their data from the remote repository and incurring long-haul communication cost, servers can exchange content information among themselves and get their data from the other nearby servers. This is possible for servers co-located in a, for example, campus network, entertainment network with cooperative servers/service providers, network with low communication cost, high processing capability, etc. Video content may be streamed from any local server in a group to any other servers through the network. If the streamed data is not copied locally, the remote server would have to be capable of serving all the requests for that movie in the server group. By making a copy of the streamed data locally, an immediate subsequent request in the local server can be served directly from itself (instead of streaming from a remote server through the network); hence decreasing the network bandwidth. The repository may multicast a movie to the group of servers, or it may unicast the movie to a local server which in turn multicast the movie to the other servers through the MAN.

In Chapter 4, we look at, given cost functions for storage and channels, the conditions under which a movie should be stored in order to minimize the cost. We consider a number of cases:

- Independent servers with unicast delivery — In this case, the local servers operate independently of each other and communicate only with the central repository with each request made by a local server getting served by a unicast (dedicated) repository stream to that local server alone. Under this condition, movies are likely to be either entirely stored or not at all and the cut-off point in the request rate for this is well-known;

- Independent servers with multicast delivery — In this case, local servers still operate independently of each other, but the repository multicasts movies to the servers so that they can store them as appropriate for future use. The optimal strategy is no longer to store a movie in its entirety or not at all; instead, a portion of the movie should be stored (corresponding to a moving window). Multicast is able to lower the cost compared with the unicast case, especially when the channel cost is relatively cheap; and

- Communicating servers — In this case, the local servers can exchange video content with each other, so that a movie not cached locally can be obtained from another local server in the network where it is cached. We see that tremendous additional cost reduction may be attained under this condition.

The local servers cache movies of interest to users and stream them to the users. A local server can handle a certain maximum request rate. By putting in more local servers, the streaming capacity can be increased.

A local server may cache a movie for possible future use. (Such buffering is also used to achieve lower user start-up delay as compared to the case of request batching.) If a local server has limited amount of storage, it would be of interest to devise some schemes to make use of the available storage so as to minimize the stream cost.[1] Here we consider that storage comes with a cost depending on the size and length of its usage. Since the service provider pays for the costs of the streams and storage, it is of interest to minimize the total cost of both streams and storage.

[1] Obviously if movies can only be stored either completely or not at all, the optimal strategy is to store the most popular movies in the local servers.

We study a number of schemes in the local servers to cache movies. The schemes allocate a certain amount of buffer for a movie when the movie is brought in from the repository; therefore all requests arriving before the beginning part of the movie is deleted can be served directly from the local cache (known as a *cache group*), hence saving extra repository streams. In this way, the caching scheme keeps a moving window and is able to keep a portion of a movie locally (i.e., *partial* movie caching). Keeping a portion of movie locally also provides interactive flexibility to the users in viewing the movies.

The caching schemes we investigate cache a movie partially, thereof achieving the trade-off between channels and storage. We consider that the network channels can be acquired on demand with a cost, and that the local servers have sufficient bandwidth to serve the requests from the users. We will consider that the latency in the central repository is low (and can be ignored).

Previous work in distributed servers architecture mainly focuses on either storing a file completely in a local server or not storing it at all (we review the previous in the respective chapter). Here we consider schemes in which movie can be partially cached locally, and hence is able to further decrease the system cost. We address in our study when and how much to cache. We have considered using multicasting as a means to deliver data to the local servers (so as to save network bandwidth), which has not been studied before in this context. Note that this system is in fact very similar to the distributed storage architectures discussed in Chapter 2. We consider here network channels can be acquired on a demand basis (as supposed to the leased case there). Furthermore, we consider system design meeting a certain user delay requirement in Chapter 2, while in here we study the effect of costs in storage and streaming on system design and how the total system cost can be minimized.

Chapter 2

Hierarchical Storage Systems

2.1 Introduction

Most video servers reported on in the literature or existing today use magnetic disks to store and stream videos. Magnetic disk technology is used due to its high throughput, low access latency, random data access, and adequate storage capacity. However, magnetic disks are still not ideal to store large volume of video files (e.g., $>$ 1000s video files for some applications), because of reliability and scalability issues. Furthermore, magnetic disks nowadays still suffer comparatively high cost — currently, storing a single 90-minutes movie ($\simeq$ 1 GB) in such a medium can cost more than a hundred dollars. As video files can also differ markedly in their popularities, some of them can be infrequently accessed. Storing all files on-line regardless of their popularities is therefore not very cost-effective [51, 15].

Video servers based on hierarchical storage systems provides a cost-effective solution. They consist of both tertiary and secondary levels. Tertiary storage, commonly referred to as library or “jukebox,” is used to store all video files. The files are transferred or “staged” on demand into the secondary level to be displayed. The secondary level in the hierarchical storage system therefore acts as a “caching” or “staging” platform for video display [52, 53].

Table 2.1 shows the typical performance characteristics in secondary and tertiary levels.

Table 2.1: Typical performance characteristics for the secondary and tertiary storage levels

	Secondary level	Tertiary level	
	Magnetic disks	Optical disks	Tapes
Media cost ($/GB)	\$200–\$300	\$5 – \$50	\$5
Typical capacity	$\leq$ 100GB	$\geq$ 1TB	$\geq$ 1TB
Bandwidth (MB/s)	2 – 10	1 – 5	1 – 5
Total access time[a]	10 ms	$\sim$ 20 sec.	$\sim$ 1 min.

[a] In the case of tertiary level, total access time includes the robotic exchange time.

From the table, we see that, as the media cost of tertiary storage can be orders of magnitude lower than that of secondary storage, storing videos in the tertiary level is much cheaper. Furthermore, as a tertiary library can shelve hundreds or even thousands of removable tapes or disks, with each of which offering high storage capacity ($\sim$10 – 100 GB), a tertiary system can hold large storage capacity in the range of terabytes. Therefore, such a hierarchical storage system offers low cost high capacity storage as compared with a system based on magnetic disks only.

In this chapter, we consider the design of a hierarchical storage system for video-on-demand to meet certain performance goals, taking into account the application characteristics (such as file sizes and popularity, the requirements in user interactivities like fast-forward (FF), rewind (RW), pause, etc). The design of a server based on a hierarchical storage system actually involves many architectural parameters such as secondary level bandwidth, secondary level storage capacity, tertiary level bandwidth, number of drives, etc., and many operational procedures including admission control, request scheduling and replacement policies.

We address our design problems using simulation and analysis. With simulation, we are able to incorporate different levels of complexity and details in our system so as to study their effects in system performance. Analysis complements simulation in that it is able to capture explicitly parameter interdependency, identify system bottlenecks and instabilities, and draw meaningful performance trends, asymptotes and limits. The model we develop is simple and allows us to address many performance and design issues of a video server of such kind.

Previous work on hierarchical storage systems has been on justifying its cost advantage [54, 53, 55], or studying the possible operational procedures and their performance [56, 57, 58, 59, 60]. Our work differs in addressing how the system parameters in a hierarchical storage system can be dimensioned so as to satisfy a certain user delay requirement. We have developed a simple model which can be used to solve many design issues related to a hierarchical storage system.

This chapter is organized as follows. We first briefly describe previous work in Sect. 2.2. In Sect. 2.3, we describe the hierarchical server architecture and its operational procedures. In Sect. 2.4, we describe the design problem, and then discuss how bandwidth and storage can be specified to meet specific performance goals. Finally in Sect. 2.5, we present how our model can be extended and used to address the design issues in different hierarchical storage systems. We conclude in Sect. 2.6.

2.2 Previous Work on Hierarchical Storage Systems

Tertiary storage is one of the system components. Efficiently using automated tertiary systems to deliver data has been discussed quite extensively in the literature. Nemoto et. al. in [61] address load-balancing issues among multiple tape libraries, by using a migration wagon which moves tapes among the libraries so as to relieve hot spots or drive contention. Video popularity is measured by its access frequency, while the heat (temperature) of a library is measured by the total of the access frequency of its tapes.

Drapeau and Katz extends the idea of RAID to tape drives in a library system so as to improve the performance of a tertiary system [62]. They consider striping data among the tapes so that tape bandwidth can be used in parallel, but with an increase in exchange latency. They show that, given a certain stripe size, when the load is low and the requested data size is large, striping improves the performance of the system; while if the load is high, striping would not improve much the performance of the system due to the exchange overhead.

Ford et. al. introduce “Redundant Array of Independent Libraries (RAIL),” in which multiple small and inexpensive libraries are configured as an array (similar to the RAID concept) [63]. The paper discusses the reliability issues in such a system (time to failure and server available time), and shows that (with 5 data library with 1 parity library) a RAIL system can have very long mean-time-to-failure (more than 50 years).

Lau and Lui in [64] study using a tape library to deliver continuous media to the users. Video files are divided into equal-sized blocks and delivered to the users in a block-by-block round-robin fashion. The main performance measure is the user’s start-up delay. A system with large file size and low drive bandwidth is investigated. It is found that round-robin scheduling as compared to file-by-file scheduling achieves lower delay when the arrival rate is low, but much higher delay as the arrival rate increases.

Using hierarchical storage systems (consisting of tertiary libraries and staging disks) as a cost-effective solution for data storage was discussed as early as in 1978 [54]. Early analytic work on hierarchical storage systems focused on non-continuous medium (data files). Misra in [52] analyzes IBM 3850 hierarchical storage system. Using a machine repairmen model, contention between resources (data paths, robotic arms, etc.) and various delay paths are studied. Pentakalos et. al. in [65] modeled the Unitree mass storage system. The model takes into account the workload based on actual measurements. The model comprehensively takes into accounts various job classes (characterized by their file sizes and access frequencies), striping disks, and tape server. Our work differs from theirs in addressing the dimensioning and optimization of the bandwidth and storage in the tertiary and secondary levels to meet a certain user delay requirement. We are also dealing with continuous media and much larger file size (which in effect increases users’ start-up delay due to file staging).

Using hierarchical storage system in VOD has been studied recently. Kienzle et. al. in

[53] compare three operations for a VOD system: i) video files are permanently stored on disks, ii) video files are directly streamed from the tertiary level (with no round-robin request multiplexing), and iii) video files are first staged onto and then streamed from the disks. Using a cost function of storage and bandwidth, it shows that for low video popularity, video should be directly streamed from the tertiary level, and for very popular video, they should be permanently stored on disk.

Chervenak et. al. in [55] study three storage systems for movies-on-demand application: i) one movie per disk, ii) striped disk farm, and iii) hierarchical storage system. Using the metric as average cost per video stream achieving a certain criteria in the user start-up delay, they find that striped disk farm performs the best due to load-balancing, and streaming videos directly from the tertiary storage is likely to have the worst performance.

Suzuki et. al. in [66] consider a hierarchical storage system with some files permanently stored in the secondary level, some have to be streamed directly from the tertiary level, while some have the initial "leading" parts resident in the disk (so as to decrease start-up latency). The performance of concern is average user's start-up delay. Given a certain disk size, if there are too many resident files, then there would be too little space for leaders and hence misses would experience longer start-up delay. If there are too few resident files, the high miss rate would load the tertiary level leading to high start-up delay. The paper shows that there is an optimal number of resident files to minimize the average start-up delay. We differ from the work in terms of system operation, and address how to *size* the secondary level according to the user delay requirement (which was not addressed in the work).

Merchant et. al. in [67] consider a design problem similar to ours here, and similar observations are obtained. Their work is parallel and independent to ours (see [68, 69]), and extended to the case of block-by-block round-robin staging. We have conducted extensive simulation to validate our model, and applied the model in different application areas (such as distributed storage systems, different staging schemes, and issues related to bandwidth sharing and partitioning).

Won and Srivastava in [56] consider a hierarchical storage system in which users are rejected if they cannot be immediately served (either by the tertiary or secondary level). The performance of interest is user rejection rate, and they study the replacement algorithms in the system performance. The secondary level is assumed to be storage-bound rather than bandwidth-bound. The authors first study that a video file in the secondary level is deleted once it finishes display and find that it leads to high user rejection rate since files are deleted too soon. By imposing a minimum residence time for each file in the disk, user rejection rate is shown to be decreased. Our work differs in that we consider a system in which requests are queued and consider user delay requirement. We are also more interested in dimensioning system parameters than replacement schemes. We find that the secondary level in the efficient design should not be bound by either storage or bandwidth. We also consider how the requirement in user interactivity affects the design of the system.

Bianchi et. al. in [57] evaluate a number of replacement policies in a hierarchical storage system consisting of local caches and a central server. The policies are stated as follows (In some of these policies, the popularity of the files is known): i) No release, in which the cache only keeps the most popular set of video files; ii) Lessprob overwrite, in which the least popular file in the cache is replaced; iii) random overwrite, in which the file to be replaced is chosen at random; iv) LRU overwrite, in which the least recently used file is replaced; v) mincall overwrite, in which the file with the minimum number of requests in a sliding window is replaced; vi) conditional LRU overwrite, in which a file from the central server is directly streamed to the users if the file is less recently-used than all the files in the cache, or else the file will replace the LRU file in the cache; vii) conditional mincall overwrite, in which a cached file is replaced if its number of requests in a sliding window is less than the requested video, or else the video would be directly streamed from the central server. It is found that, in terms of hit probability, the no release policy performs the best, and the random replacement policy performs the worst (their hit probabilities differ by about 10% in their examples).

A pipelining mechanism to stage video file from the tertiary level to the secondary level is discussed in [58, 59] as a means to reduce user's start-up delay. In such a system, video files can be displayed while being staged onto the secondary level. Three data flow sequences are examined in terms of their buffer requirements, the disk space requirements, and the conditions for continuity: i) sequential data flow in which the blocks of a video file are first transferred from the tertiary level to the memory, then from the memory to the disk, from the disk back to the memory, and finally from the memory to the user; ii) parallel data flow in which the file blocks are first transferred from the tertiary level to the memory, then from the memory to the user and the disk simultaneously; iii) incomplete data flow, in which the data flows directly from the tertiary level to the memory and then to the user (without writing to the disks).

Wang and Hua in [60] study a pipelining mechanism for VOD in which the initial part of the files (leaders) are permanently stored in the disks (hence reducing the initial delay to get the pipeline going). The displayed portion of a file is immediately deleted. Video popularity is assumed to be known. It shows that if the leader and staging bandwidth of a file is properly designed (with the result that the leader for a popular file is larger than that of a less popular file, and its staging bandwidth as being lower), the requirements in both tertiary bandwidth and disk storage can be reduced (under certain conditions, by more than 2 orders of magnitudes) compared with the case in which staging bandwidth and leader size are the same for all video files.

Federighi and Rowe address in [70] the database issue in a hierarchical storage system: the design of its file manager, the structure of the file system, search engine to locate and load videos, and cache management algorithms.

Our study differ from [56, 57, 58, 59, 60, 70] in that we focus more on dimensioning architectural parameters than studying different operational procedures.

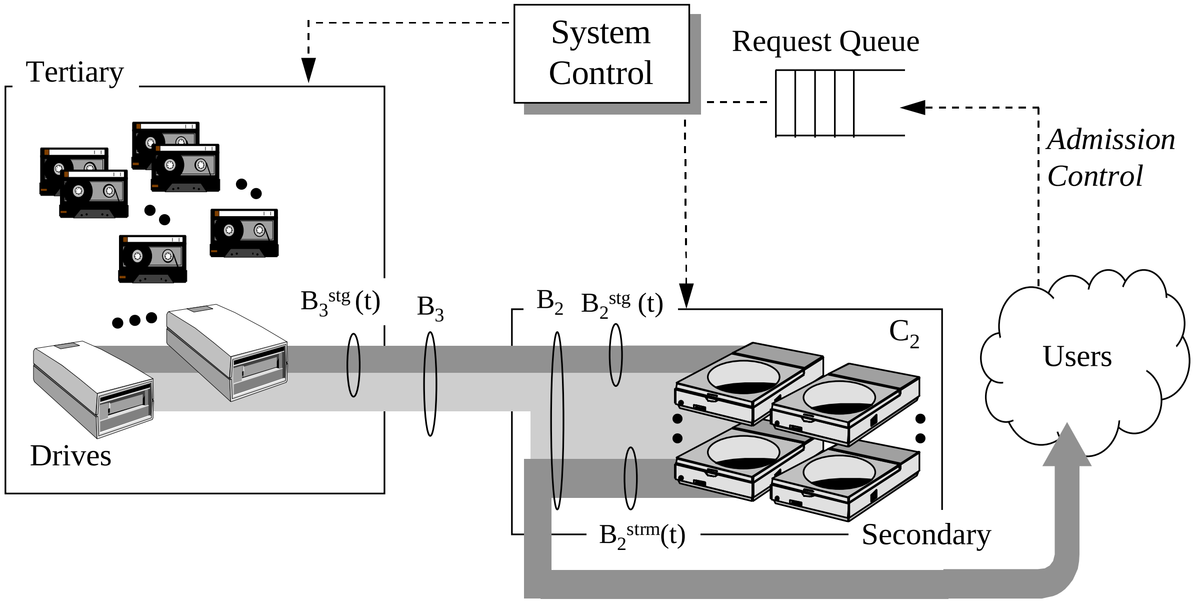


Figure 2.1: A hierarchical storage system for an on-demand video server.

2.3 The Architecture and Operation of a Hierarchical Storage System

2.3.1 System architecture

Figure 2.1 shows a hierarchical storage system for an on-demand video server. The request queue holds the users with their videos not displayed yet. The system control dispatches the accepted requests to the respective levels for service. The levels are described as follows:

Secondary level: The secondary storage level consists of multiple magnetic disks, though rewritable optical disks may also be used. The level is characterized by high throughput and fast data access, and hence is particularly suitable for video streaming. A number of disks in this level can be configured to form a single logical unit known as a disk array, or RAID (Redundant array of inexpensive disks) [16, 18, 37]. In this way, data can be accessed in the array as if it were a single disk volume. There are different RAID levels, each corresponding to how data is laid out and accessed in an application [20, 21, 19].

We characterize the secondary level by its total storage capacity, C_2, and its total aggregate "effective" bandwidth, B_2. The bandwidth B_2 has taken into account the access overheads in the disks (which is related to assess latency and block size of the disks), and hence the maximum number of streams that can be opened at any time in the secondary level is,

$$s_2 = \lfloor B_2/b_0 \rfloor, \tag{2.1}$$

where $\lfloor x \rfloor$ denotes the largest integer less than or equal to x.

If a requested video is not in the secondary level, it has to be staged from the tertiary level, during which or after which it can be streamed. At any particular time t, a portion of B_2 is hence used for video streaming ($B_2^{strm}(t)$), and another portion is used for video staging ($B_2^{stg}(t)$). Obviously, we must have,

$$B_2^{stg}(t) + B_2^{strm}(t) \quad \leq \quad B_2. \tag{2.2}$$

$B_2 - B_2^{stg}(t) + B_2^{strm}(t)$ is hence the idle bandwidth in the secondary level at time t.

Secondary bandwidth B_2 can be either "partitioned" or "shared" for staging and streaming purposes. In the "partitioned" case, a certain amount of secondary bandwidth is permanently set aside for staging or streaming purpose. The bandwidth allocated is therefore left unused if it is not used for the intended purpose. On the other hand, in the "shared" policy, there is no such strict bandwidth partitioning — the unused secondary bandwidth can be allocated for either staging or streaming purpose. (We see that when B_2 and B_3 are not under-utilized, there is no much difference between the shared and partitioned policies.)

As the secondary level has limited storage capacity C_2, not all video titles can be stored at this level simultaneously. Using file replacement and staging mechanisms, video files are able to "share" the limited storage space in order to be displayed to the users. We denote N_2 the maximum number of video files that can be stored in the secondary level, i.e.,

$$N_2 = \frac{C_2}{C_f}, \tag{2.3}$$

and $N_3 \triangleq N_v - N_2$ the remaining number of files in the tertiary level.

Tertiary level: Tertiary storage, also commonly known as the "automated library" or "jukebox," is characterized by three components:

- A cabinet of removable media — A tertiary library shelves many ($\geq\sim$ 100) removable storage media, such as tapes (magnetic tapes or optical tapes) or optical disks (e.g., CD, WORM disk, MO disk, or Phase-change disk, etc.). As each tape or disk has high storage capacity (in the range of 10 – 100 GB), the total storage capacity in this level can be greatly enhanced by shelving more tapes or disks. Hence, the tertiary level can offer highly scalable storage in excess of terabytes.

- Drives — The tapes or disks in the library share a number of drives. This number, $N_{dr}^{(3)}$, typically ranges from 2 – 16.

  Each drive comes with a certain "effective" bandwidth (taking into account file access overhead and block size), b_3; hence at any time the maximum bandwidth the tertiary storage level can deliver is given by $B_3 = b_3 N_{dr}^{(3)}$ (B_3 is called the tertiary bandwidth).

- An automated mechanism — The tapes or disks in the library have to share the limited number of drives by being swapped in and out of the drives. Such swapping or exchange is done by some robotic or automated mechanisms. There are N_{rbt} robots in the library. The total time it takes for an exchange is called exchange latency, T_{ex}. Note that T_{ex} does not include file seek time, which has been taken into account when defining the effective drive bandwidth, b_3.

All video files are stored in the tertiary level. Video files which need to be displayed but are absent from the secondary level have to be staged from this level by using a certain amount of tertiary bandwidth B_3 ($B_3^{stg}(t)$) and also a portion of the secondary bandwidth B_2 ($B_2^{stg}(t)$). We obviously must have

$$\begin{aligned} B_3^{stg}(t) &\leq B_3, && (2.4) \\ B_3^{stg}(t) &= B_2^{stg}(t). && (2.5) \end{aligned}$$

Obviously, it is only necessary to consider, $B_3 \leq B_2$.

2.3.2 Operational procedures

Requests coming into the system are first put into a request queue waiting to be served. If the requested file is in the secondary level, the request is a hit and is served upon the availability of a stream. Otherwise the request is a miss and the data has to be brought from the tertiary level. Upon the availability of staging bandwidth and storage space in the secondary level, the file will be staged and streamed to the user.

There are a number of request-scheduling policies pertaining to how to schedule the requests in the request queue and in the miss queue in the tertiary level. There are also policies pertaining to which file is to be replaced in the secondary level. From our simulation study, we find that architectural parameters (B_2, C_2 and B_3) are more crucial than scheduling policies in determining the system performance. Therefore in this chapter we will focus on the dimensioning of these parameters so as to achieve a certain user delay goal.

There are in fact a multitude of schemes according to which a hierarchical storage system can be operated. This is stated as follows:

- Staging — In terms of how files can be staged from the tertiary level to the secondary level, we may consider two modes of operations: i) *File-by-file staging*, in which a file in a tertiary drive is staged to its completion before it is swapped out of the drive and replaced by another file (and hence incurring minimal exchange latencies); ii) *Round-robin staging*, in which a tertiary file is divided into blocks. Files take turns to be staged into the secondary level one block at a time (Despite more exchange overheads, this potentially decreases user start-up delay [64]).

- Tertiary drives operation — The tertiary drives can be operated independently, hence achieving high request concurrency (serving multiple requests at the same time) but low bandwidth parallelism (using all the bandwidths at the same time). They can also be configured as if they formed a single drive (similar to a disk array) through fine-grained striping [62, 63]. In this way, the bandwidth of the drives can be used in parallel to deliver a file and hence the delivery bandwidth of a file is increased. As the drives are no longer independent in this case, it is as if $N_{dr}^{(3)} = 1$. Such "parallel-drive" configuration achieves maximal bandwidth parallelism but minimal request concurrency.

- Video display time — A video can be displayed once it is completely staged into the secondary level (hence its user can interact with the video from its beginning to its end), or it can be displayed once it starts to be staged.

- Video data deletion — We may keep the displayed portion of a video. We may delete the displayed portion and keep only the portion to be displayed later, in which case the user cannot revisit the previous part of the video. This is called "trail deletion."

Figure 2.2 shows for a given file the total data delivered from the tertiary level to the secondary level as a function of time, with different combination of the operation schemes. A miss arrival first wait for an available tertiary drive. After some time, the file begins to be staged. If a file is displayed only after it is completely staged, the start-up delay is the time from the request arrival until the total data delivered is equal to the file size C_f; otherwise, the display time of the video is the time at which staging begins. In any case, after the video finishes displaying to the user, the file can be deleted. Also shown in the figure is the data produced when round-robin staging is used. We also indicate in the graph the total data consumed (i.e., displayed).

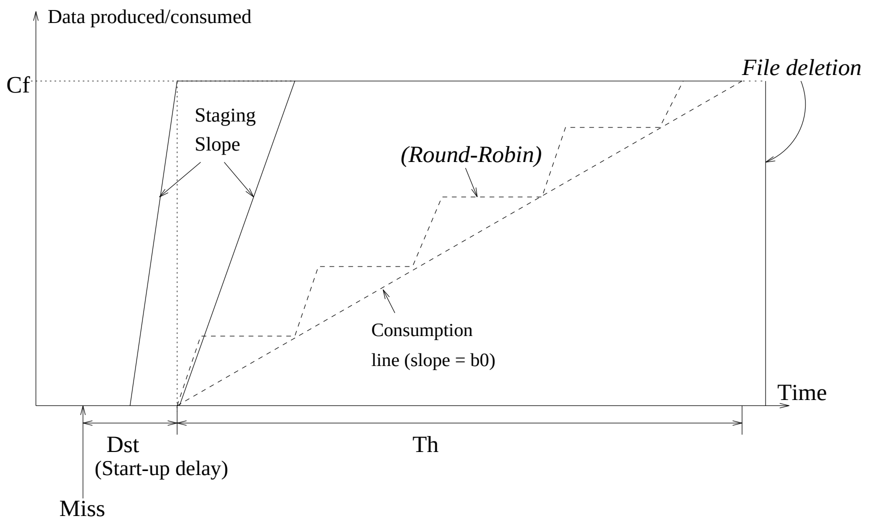


Figure 2.2: Total data staged to the secondary level from the tertiary level, with different operation schemes

Note that with trail deletion, the difference between the data consumption line and the data production line is the actual storage occupied by the file in the secondary level. This is shown in Fig. 2.3. We see that round robin staging can greatly decrease the storage requirement of a file, and with trail-deletion, such storage can be further decreased. However, since a user can only interact with the data in the secondary level, the lower storage also means that there is less flexibility in interacting with the displaying video. Consequently, combination of the operation schemes has implications in the storage requirement and user interactivity offered.

2.4 Storage and Bandwidth Requirements

2.4.1 Design problem

We investigate the performance of the hierarchical storage system with the following application characteristics:

- Demand characteristics:

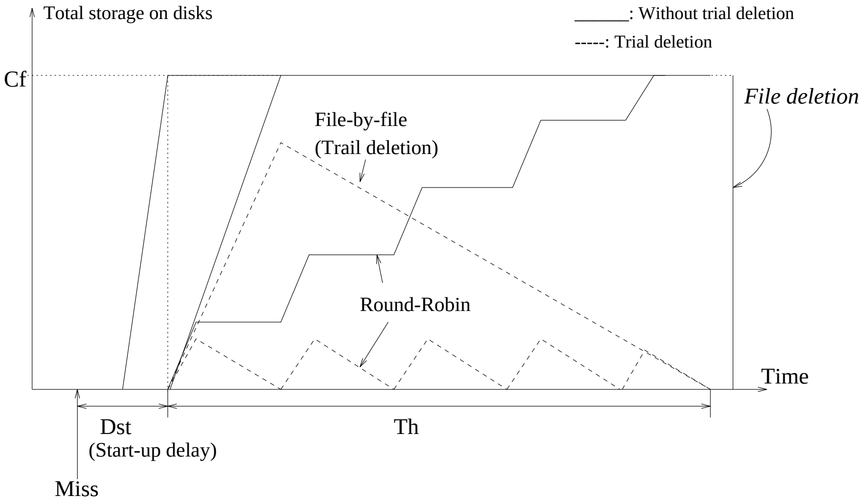


Figure 2.3: Total storage occupied by a staging file in the secondary level with different operation schemes

  - Poisson arrivals with rate λ (requests/minute);[1]
  - Constant request holding time, T_h (minutes);[2]
  - All users are admitted and there is no balking or reneging, i.e., all arrivals wait until their videos are displayed.

- Video characteristics:

  - Constant streaming bandwidth, b_0, for all video files at all times;
  - All video files are of similar sizes C_f;
  - All files in the secondary level may be replaced;[3] and
  - Unmodifiable files, i.e., we assume read-only applications.

[1] Note that for presentation purpose, λ is shown in our graphs as requests/hour.

[2] Through simulation, we find that a system with deterministic holding time has similar performance as a system with arbitrary holding time distribution with the same mean. Therefore, the deterministic condition here is not so restrictive as it appears. Deterministic holding time, however, lends itself to simpler and tractable analysis.

[3] The extension of our analysis to the case where some files can never be deleted is straightforward.

Our design problem can be posed as follows: Given a certain target traffic rate λ_0, how should a hierarchical video server be designed (in terms of C_2, B_2, and B_3) so as to satisfy a certain delay requirement? The delay requirement can be average start-up delay for all users ($\hat{D}$), average start-up delay for all misses ($\hat{D}^{Miss}$), or according to some statistical guarantees (e.g., no more than a certain percentage of requests suffer delay higher than a certain value).

We summarize our design problem as follows:

Given: Application characteristics,
(user holding time, file characteristics,
and interactive characteristics) and
target arrival rate λ_0 (We consider $\lambda_0 T_h \ll N_v$,
i.e., large video base);

Select appropriate: Operational procedures,
C_2 (secondary level storage),
B_2 (secondary level bandwidth),
B_3 (tertiary level bandwidth), and
$N_{dr}^{(3)}$ (number of tertiary drives);

In order to: Satisfy a delay requirement, e.g.,
(Start-up delay requirement) $\overline{D}_{st} \leq \hat{D}$, or
(Miss delay requirement) $\overline{D}^{Miss} \leq \hat{D}^{Miss}$, or
(Delay guarantee) $P(D_{st} > \hat{D}) \leq \varepsilon$.

Note that there can be a range of parameter values satisfying the delay requirement. In reality, specific value can be chosen if i) there is a cost associated with each value and the total cost of the system is to be minimized; ii) the parameters may not take on continous values and hence given today's technology, there is only a few feasible sets of values (For example, if a disk comes with a certain storage and bandwidth, the secondary storage and bandwidth would then be scaled accordingly as the number of disks increases. A tertiary drive may also have a certain amount of bandwidth, making B_3 take on discrete values); or iii) there is a "knee" in the trade-off curve, making the point likely to be the appropriate design choice.

There are many parameters to be specified in the design formulation. Simulation may be used to search for the appropriate parameter values, but it would be very slow and inefficient. To facilitate the design process, we have used analytic modelling. The model developed is then validated with simulation.

We are also interested in the bandwidth utilization at both the tertiary and secondary levels. We define the utilization of the tertiary bandwidth, ρ_3, as the time-averaged fraction of the tertiary bandwidth used for staging purpose. If λ_3 is the miss request rate at the tertiary level, and suppose that each new miss results in a new file to be staged (approximately true for $N_v \gg \lambda_0 T_h$), then

$$\rho_3 = \left(\frac{C_f}{B_3}\right) \lambda_3. \tag{2.6}$$

In the secondary level, there are three measures for bandwidth utilization: i) the total utilization for both streaming and staging, ρ_2; ii) utilization for streaming, ρ_2^{strm}; and iii) utilization for staging, ρ_2^{stg}. Note that since each arrival would need a stream b_0 and on average there are $\lambda_0 T_h$ concurrent users, ρ_2^{strm} is given by,

$$\rho_2^{strm} = \left(\frac{T_h b_0}{B_2}\right) \lambda. \tag{2.7}$$

Obviously, by continuity, we must have

$$\rho_2^{stg} = \rho_3^{stg} \frac{B_3}{B_2}. \tag{2.8}$$

Hence,

$$\rho_2 = \rho_2^{strm} + \rho_2^{stg}. \tag{2.9}$$

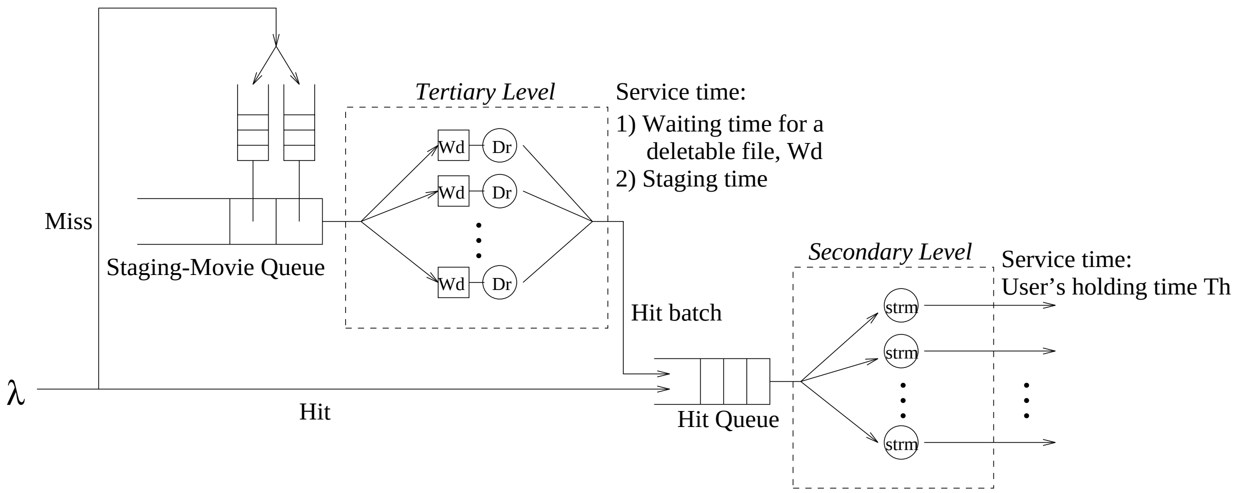


Figure 2.4: Request flow in a hierarchical storage system

2.4.2 System model

We show in Fig. 2.4 the system model of the hierarchical storage system external arrival rate λ (The design point is then given by $\lambda = \lambda_0$). Each arrival independently chooses movie i with probability p_i ($1 \leq i \leq N_v$). For simplicity, we will first discuss the case with uniform video popularity (hence $p_i = 1/N_v$) and file-by-file staging. We extend our model to cover non-uniform video popularity in Sect. 2.5; note that the performance with uniform video popularity is pessimistic if video popularity is in fact skewed. Since we will consider uniform video popularity first, it does not matter which replacement algorithm we use. For concreteness, we choose the Least-Recently-Used (LRU) algorithm, in which we delete the file which has been closed for the longest time.

An arrival is put into the hit queue if it is a hit, and wait for an available stream to be served. If it is a miss, it joins the staging-movie queue, which keeps track of the name of movies to be staged. The movies are scheduled according to the First-Come-First-Serve (FCFS) algorithm. After a certain waiting time, all the misses corresponding to a movie will become hits and join the hit queue.

The secondary level allocates its bandwidth as follows: it first schedules the hits according to their arrival order, by assigning a streaming bandwidth to each of them. If there is still unused bandwidth left (i.e., $B_2 - B_2^{strm}(t) > 0$ at that time t), it is assigned for staging purpose. Note that the arrival of a hit may take away some bandwidths from an on-going staging session.

Since the hit and miss events are state-dependent, the analysis of the hierarchical storage system is difficult. This is illustrated as follows: let $\mathbf{N} = \{N_{res}, N_{stg}, N_v^{stg}\}$, $\mathbf{N} \geq 0$, be the state of the system at any time, where N_{res}, N_{stg} and N_v^{stg} are the number of resident files in the secondary level, files being staged and total number of files waiting to be staged (including those being staged), respectively. Obviously, we have $N_{stg} \leq \min\{N_{dr}^{(3)}, N_v^{stg}\}$, $N_{res} + N_{stg} \leq N_v$, $N_{res} \leq N_2$ and $N_v^{stg} \leq N_v - N_{stg}$. With large N_v (typically the case), the state space is large (of order $O(N_v^2)$, with N_v typically greater than or equal to 100) and the transitions can be very complex. Furthermore, since the staging time and waiting time for a deletable file are not exponential, the transition between states are not Markov. These make the analysis difficult.

Due to the above difficulties, we assume that the probability that an arrival finding its file being staged is negligible (i.e., we assume $N_2 \gg \overline{N}_{stg}$, justifiable when staging time is much less than the movie length) so that hit and miss probabilities become constant in time, i.e., $\alpha_2 = N_2/N_v$ and $\alpha_3 = 1 - N_2/N_v$, respectively. This is in fact observed in simulation. Therefore, a random arrival would enter the tertiary level with probability α_3, i.e., the traffic of tertiary level is Poisson with rate $\lambda_3 \triangleq \alpha_3 \lambda$.

The tertiary level and the secondary level are coupled through the waiting time for a deletable file: it may happen that at a time all the files in the secondary level are not deletable. In this case, staging cannot proceed. This "all-files-undeletable" phenomenon may persist for a while until a file is closed, and hence rendering it deletable. If C_2 is small, such waiting time can be rather long.

If B_2 is shared between staging and streaming (i.e., the unused secondary bandwidth can be freely allocated for either staging or streaming purpose), the staging process in the tertiary level then depends on the excess bandwidth in the secondary level and hence the two levels are coupled together (through the staging bandwidth $B_3^{stg}(t)$): the maximum

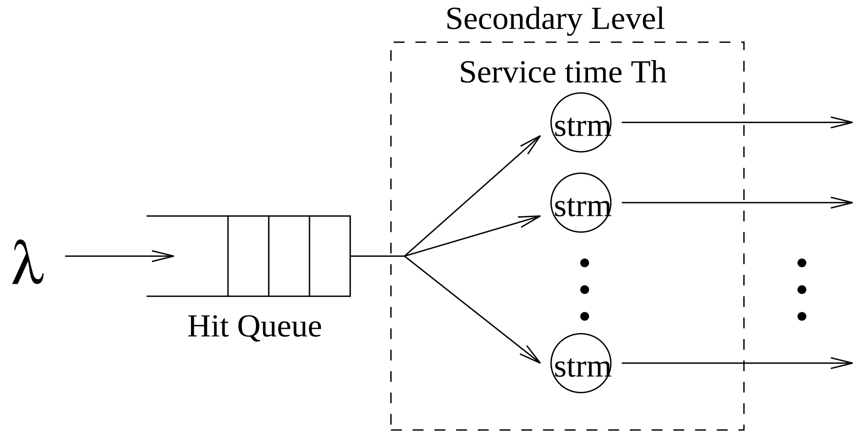


Figure 2.5: Secondary sub-system

staging bandwidth from the tertiary level at any time t depends on the bandwidth used for streaming in the secondary level by $B_3^{stg}(t) = \min\{B_2 - B_2^{strm}(t), B_3\}$. Obviously, there would be no such coupling if bandwidth is partitioned.

Note that coupling leads to staging inefficiency and hence low performance (this is later demonstrated in Fig. 2.15). Under the target arrival rate λ_0, such inefficiency can be relieved by having large enough C_2 and B_2. The resultant system is thus a “decoupled” system in which the two levels operate independently most of the time. We show in Figs. 2.5 and 2.6 the resultant models for the two levels, namely the secondary sub-system and the tertiary sub-system, respectively. Clearly, when $\lambda \leq \lambda_0$, the system can be analyzed using the decoupled models. However, when $\lambda > \lambda_0$, the decoupling may no longer be true and we have used simulation to obtain its performance.

The secondary level consists of a hit queue with request rate λ, with a number s_2 of streams available. Under the condition of large video base (i.e., $\lambda_0 T_h \ll N_v$), each request would hold a stream and a storage equal to C_f for duration T_h before leaving the system.

The tertiary subsystem consists of $N_{dr}^{(3)}$ drives (“server stations”), each with processing time $C_f/b_3 + T_{ex}$. Note that in general, $T_{ex} \ll C_f/b_3$ and hence, for clarity, we will consider in the following that T_{ex} can be ignored (our results can be easily extended to non-negligible T_{ex}). The arrival rate in the tertiary level is Poisson with rate $\lambda_3 = \alpha_3 \lambda$ (at $\lambda = \lambda_0$, λ_3 is

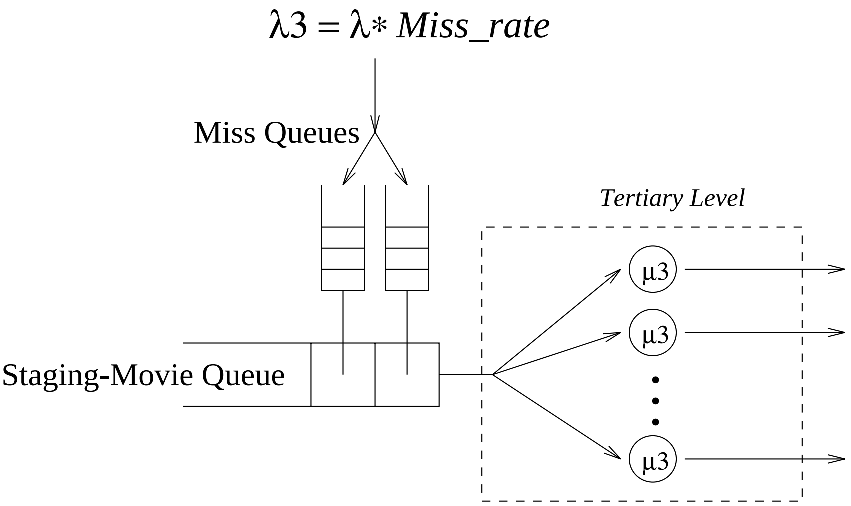


Figure 2.6: Tertiary sub-system

hence given by $\alpha_3\lambda_0$). Since N_v is large, we may make the approximation that each miss chooses a movie which is different from the outstanding ones to be staged. Therefore the system can be modeled as an $M/D/N_{dr}^{(3)}$ system.

Note that the average delay for all users is,

$$\overline{D}_{st} \quad = \quad \overline{D}_{st}^{Hit} + \alpha_3 \overline{D}^{Miss}, \tag{2.10}$$

where $\overline{D}_{st}^{Hit}$ is the average delay in the hit queue, and $\overline{D}^{Miss}$ is the average delay in the tertiary subsystem.

Table 2.2 summarizes some of the important parameters we used in our study of the hierarchical storage system.

2.4.3 Secondary subsystem design

The arrival rate in the secondary system is λ. We show in Fig. 2.7 the interarrival time of the requests in the secondary system, with the discrete points from simulation and the continuous lines corresponding to an exponential distribution with the same rate λ. Despite the joining of the requests from the tertiary level (and hence making the aggregate arrival process in the secondary level not Poisson), the arrival process in the secondary level can

Table 2.2: Important variables considered in the chapter

Variables		Remarks
λ	$=$	External arrival rate to the server
λ_0	$=$	Target arrival rate at which the server is operated under
λ_3	$=$	Arrival rate in the tertiary level
D_{st}^{Hit}	$=$	Start-up delay for hits
N_v	$=$	Total number of video files in the system
N_2	$=$	Total number of video files that can be stored in the secondary level
N_3	$=$	$N_v - N_2$
α_2	$=$	Hit probability for an incoming request $= N_2/N_v$
α_3	$=$	Miss probability for an incoming request $\equiv 1 - \alpha_2$
b_0	$=$	Streaming bandwidth for a video file
C_f	$=$	Size of a video file (GB)
T_h	$=$	User's holding time (constant)
s_2	$=$	Maximum number of streams in the secondary level $= \lfloor B_2/b_0 \rfloor$
C_2	$=$	Total storage capacity in the secondary level (GB) $= N_2C_f$
B_2	$=$	Total bandwidth in the secondary level
B_3	$=$	Total bandwidth in the tertiary level
$1/\mu_3$	$=$	Average service time of the drives in the tertiary level (staging time + exchange time)

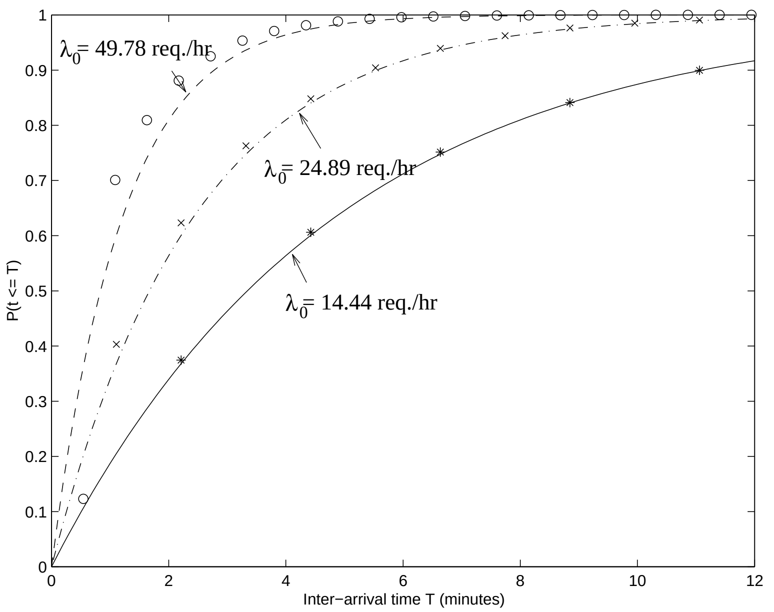


Figure 2.7: Cumulative distribution function (Cdf) for the interarrival time of the requests into the hit queue. The discrete points come from simulation, and the continuous lines is the cdf of a Poisson process with the same λ (C_2 = 78 GB, B_2 = 35 MB/s, B_3 = 19 MB/s, $N_{dr}^{(3)}$ =1, b_0 = 1.5 Mb/s, C_f = 1 GB, N_v = 500, T_h = 90 minutes)

be approximated by a Poisson process. Hence, the secondary level can be modeled as an $M/D/s_2$ system with holding time T_h.

The delay of an $M/D/s_2$ system has been extensively studied (See, for example, [71, 72, 73, 74]). We use here the approximation given in [74] for $M/G/s_2$ systems. The average start-up delay for hits, D_{st}^{Hit}, is therefore given by:

$$\overline{D}_{st}^{Hit} \approx \frac{\lambda^{s_2} E[T_h^2](E[T_h])^{s_2-1}}{2(s_2-1)!(s_2-\lambda E[T_h])^2\left[\sum_{j=0}^{s_2-1}\frac{(\lambda E[T_h])^j}{j!}+\frac{(\lambda E[T_h])^{s_2}}{(s_2-1)!(s_2-\lambda E[T_h])}\right]} \tag{2.11}$$

$$= T_h\frac{(\lambda T_h)^{s_2}}{2(s_2-1)!(s_2-\lambda T_h)^2\left[\sum_{j=0}^{s_2-1}\frac{(\lambda T_h)^j}{j!}+\frac{(\lambda T_h)^{s_2}}{(s_2-1)!(s_2-\lambda T_h)}\right]}. \tag{2.12}$$

Figure 2.8 shows the average waiting time as utilization $\rho \triangleq \lambda T_h/s_2$ increases, given s_2 for T_h = 90 minutes. We see that there is a rather sharp knee as ρ increases. As long as s_2 is not low ($\geq$ 10) and for most ρ of interest ($\leq$ 0.75), the start-up delay for the hits is

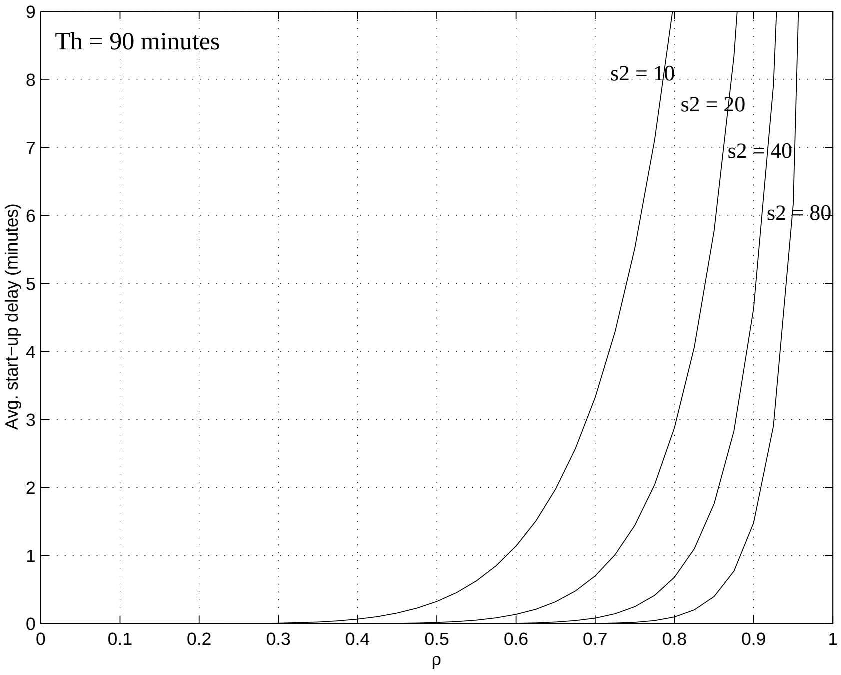


Figure 2.8: Average waiting time (Average start-up delay) for an $M/D/s_2$ system, with $T_h = 90$ minutes and $\rho \triangleq \lambda T_h/s_2$

negligible (compared with the holding time). For example, if $B_2 = 20$ MB/s and $b_0 = 1.5$ Mb/s (giving $s_2 = \lfloor B_2/b_0 \rfloor = 106$), there is hardly any hit delay even for $\rho \simeq 0.9$ (i.e., $\lambda = 64$ req./hr)! Therefore, hits enjoy very low delay under most traffic conditions; and in reality, the tertiary level limits the performance of the server. We may hence approximate the average delay for all users $\overline{D}_{st}$ in Equation (2.10) as,

$$\overline{D}_{st} \simeq \alpha_3 \overline{D}^{Miss}. \tag{2.13}$$

We observe from Eq. (2.11) that even though the delay is expected to be higher for non-uniform holding time (by an approximate factor $E[T_h^2]/(E[T_h])^2$), so long as $s_2 \gg 1$ and ρ is not too high (≤ 0.75), such difference is negligible compared with the delay incurred in the tertiary level. Indeed, we show in Fig. 2.9 $\overline{D}_{st}$ versus λ based on simulation, given the following distribution of the holding time with the same mean: 1) Constant holding time with $T_h = 90$ min.; 2) Uniform holding time $\sim$ U[60 min., 120 min.]; and 3) Exponential holding time $\sim \exp[\mu]$, with $1/\mu = 90$ min. ($C_2 = 100$ GB, $B_2 = 20$ MB/s, $B_3 = 10$ MB/s,

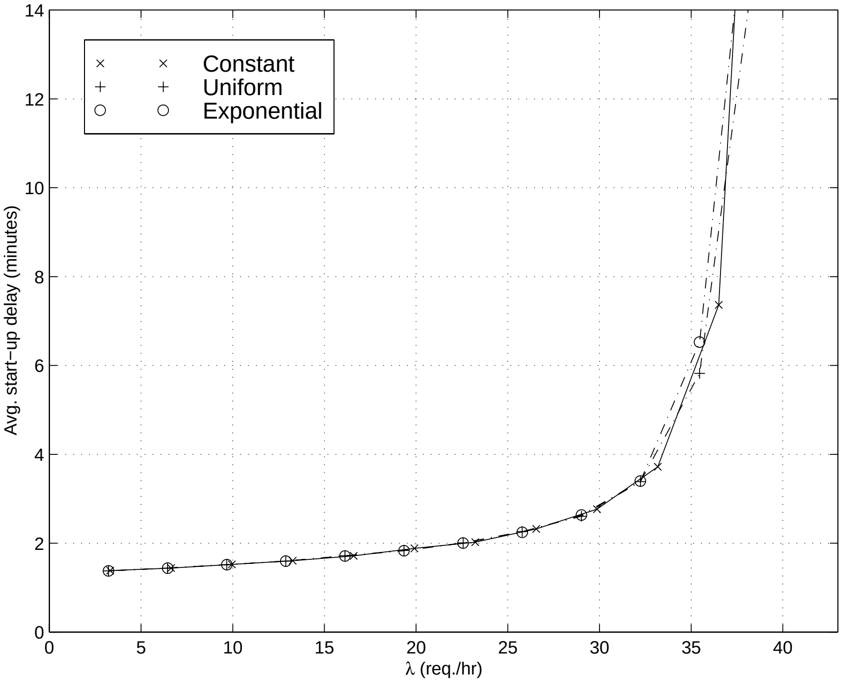


Figure 2.9: Influence of the holding time distribution on the delay performance of a hierarchical storage system (C_2 = 100 GB, B_2 = 20 MB/s, B_3 = 10 MB/s, N_v = 500, and uniform video popularity)

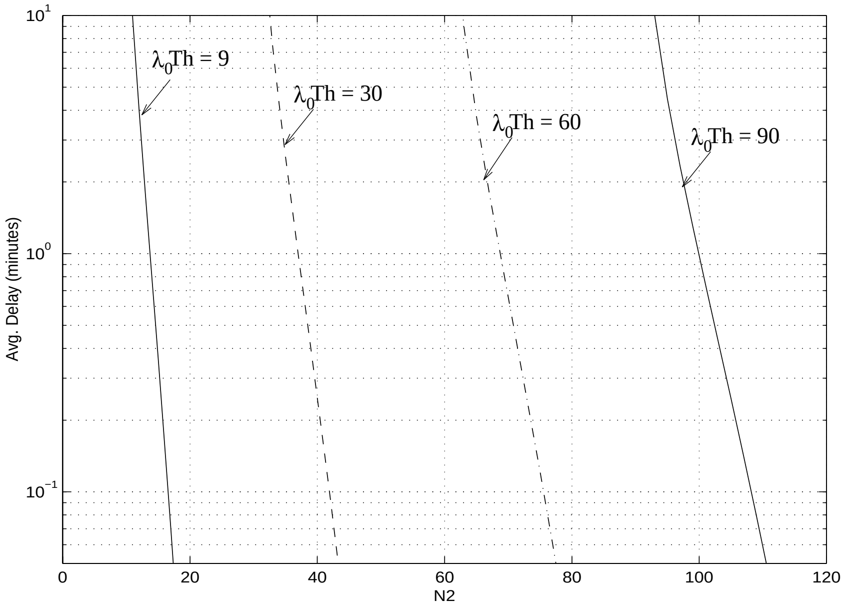


Figure 2.10: Average waiting time for a deletable file vs. N_2, given $\lambda_0 T_h$

$N_v = 100$, and uniform video popularity). Obviously, the holding time distribution does not have much effect in $\overline{D}_{st}$.

We now find the required C_2 given λ_0 so that the average waiting time for a deletable file is negligible. We plot in Fig. 2.10 the waiting time against N_2 ($\triangleq C_2/C_f$), for different values of $\lambda_0 T_h$ (based on simulation). As λ_0 increases, the corresponding C_2 has to be increased so that the waiting time is negligible (say, $\leq\sim 0.1$ minute). Note that since on average $\lambda_0 T_h$ files would be open (assuming that the staging time is negligible compared with T_h), we must have $C_2 \geq \lambda_0 T_h C_f$. Write $C_2 = (\lambda_0 T_h + n_2) C_f$, where n_2 is the additional number of file spaces above the mean required so that the waiting time for a deletable file becomes negligible. We plot in Fig. 2.11 the waiting time against n_2. We see that n_2 can be very small, in the range of only 8 – 18.

Note that the bandwidth needed for streaming in the secondary level can also be read from the same figures above, i.e., B_2 has to be at least $(\lambda_0 T_h + n_2) b_0$ for streaming purpose.

Since B_2 is also used for staging (with maximum bandwidth used B_3), the total B_2 required is $B_2 = (\lambda_0 T_h + n_2) b_0 + B_3 + \Delta$, where Δ is the "sharing" term (if the bandwidth B_2 is partitioned, $\Delta = 0$; while if the bandwidth is shared, $\Delta \leq 0$). Normally B_3 is well-

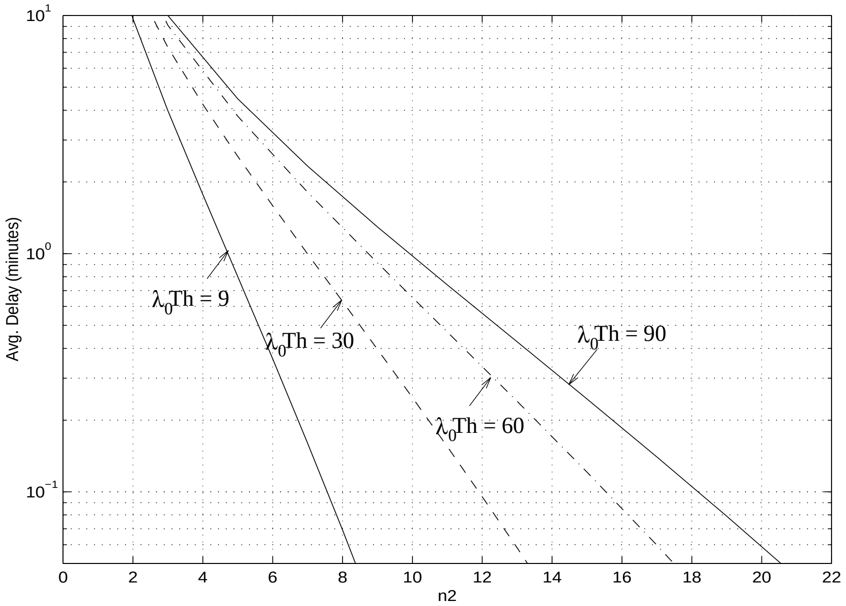


Figure 2.11: Average waiting time for a deletable files vs. n_2, given $\lambda_0 T_h$

utilized and $\lambda_0 T_h \gg n_2$. Therefore, the sharing term is small and $B_2 \approx (\lambda_0 T_h + n_2) b_0 + B_3$. This linear relationship and hard trade-off are illustrated in Fig. 2.12, which shows the trade-off between B_2 and B_3 to achieve a certain $\overline{D}_{st}$ based on simulation, along with the linear line $B_2 = B_3 + \lambda_0 T_h b_0$.

Now the only parameter left to be determined is B_3, which is obtained by examining the tertiary subsystem.

2.4.4 Tertiary subsystem design

Fig. 2.13 shows the typical start-up delay distribution for the hierarchical storage system in which a video is displayed after it has been completely staged into the secondary level (with file-by-file staging). The impulse at the origin corresponds to the hits, which enjoy very little delay. There is another impulse corresponding to those misses without queuing time (and hence its start-up delay is given by C_f/B_3) — the spread of the distribution is due to the queueing time for an available drive in the tertiary level. Since user delay is determined by the misses, B_3 is chosen to satisfy user delay requirement.

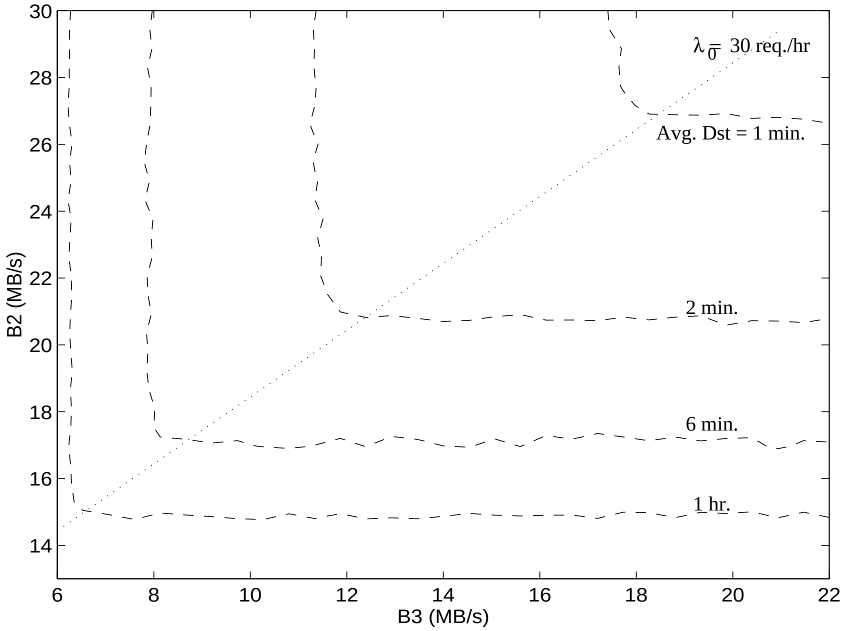


Figure 2.12: Trade-off between B_2 and B_3, for various values of average start-up delays. The linear line is $B_2 = B_3 + \lambda_0 T_h b_0$. ($\lambda_0 = 30$ req./hr, $C_2 = 100$ GB, $N_v = 500$, and uniform video popularity)

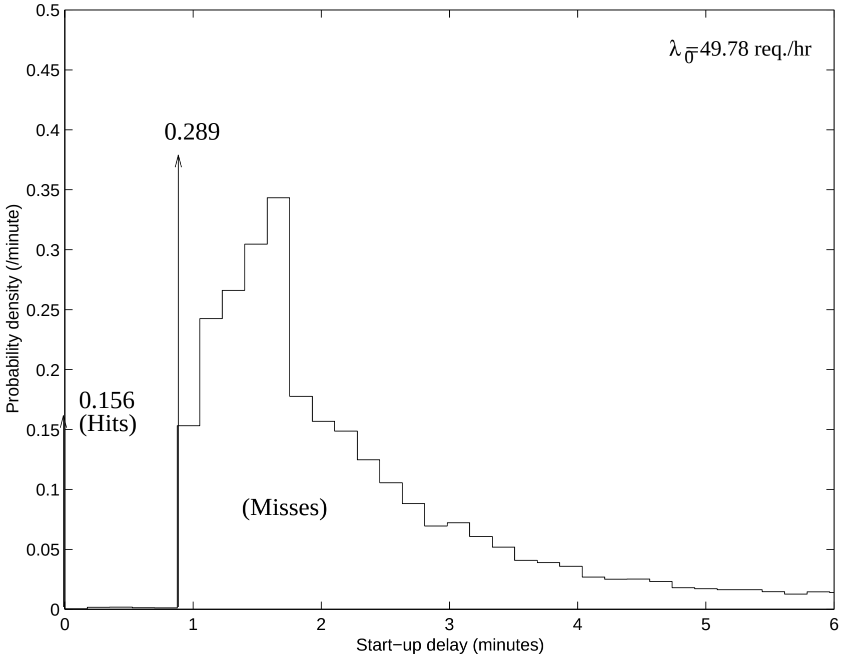


Figure 2.13: Distribution for the start-up delay in a hierarchical storage system with $\lambda_0 \simeq$ 50 req./hr (C_2 = 78 GB, B_2 = 35 MB/s, B_3 = 19 MB/s, $N_{dr}^{(3)}$ =1, b_0 = 1.5 Mb/s, C_f = 1 GB, N_v = 500, T_h = 90 minutes)

We hence may obtain B_3 as follows (given a certain requirement in the average overall start-up delay and λ_0). Recall that the tertiary subsystem can be modeled as an $M/D/N_{dr}^{(3)}$ queue with arrival rate $\lambda_3 = (1 - N_2/N_v)\lambda_0$ and service time $1/\mu_3 = C_f/b_3$. In reality, there would not be much difference if an $M/M/N_{dr}^{(3)}$ model is used. In Fig. 2.14, we compare an $M/M/1$ model with an $M/D/1$ model in terms of the B_3 required to meet a certain average system delay (2 minutes) [10, 9].[4] We see that, as expected, B_3 corresponding to an M/M model is higher, but there is no significant difference between the bandwidth required in the two cases. For simplicity, we hence may use an $M/M/N_{dr}^{(3)}$ model to find B_3.

If a video has to be completely staged before it is displayed, the overall average delay requirement is first translated to the miss delay requirement according to:

$$\hat{D}^{Miss} \simeq \hat{D}/\alpha_3 \tag{2.14}$$

[4] For the $M/D/1$ model, B_3 is given by $C_f/(2\overline{D}^{Miss})(\lambda_3\overline{D}^{Miss} + 1 + \sqrt{\lambda_3^2(\overline{D}^{Miss})^2 + 1})$, while for the $M/M/1$ model, B_3 is given by $C_f(\lambda_3 + 1/\overline{D}^{Miss})$.

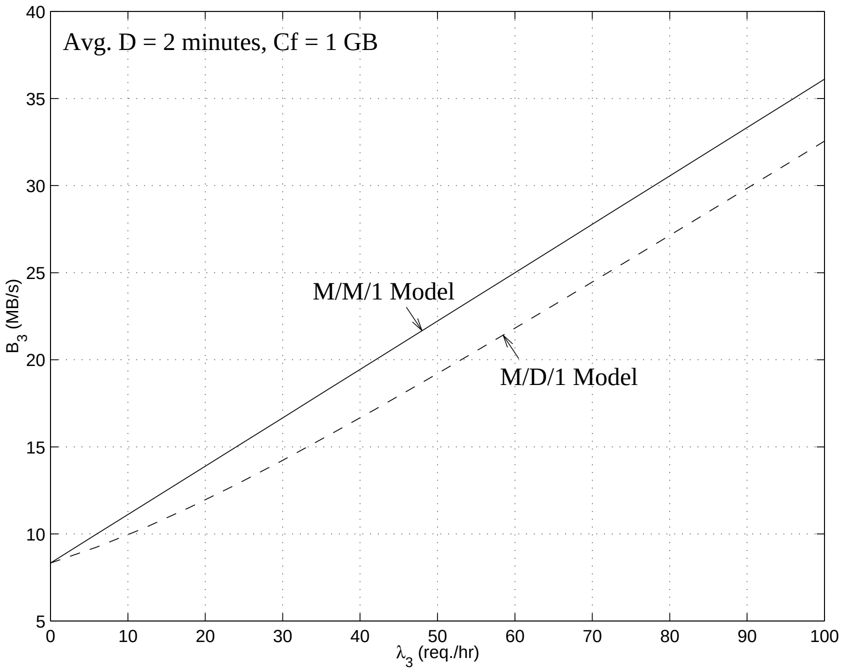


Figure 2.14: Comparison between an M/M model and an M/D model of the B_3 required to achieve a certain $\overline{D}^{Miss}$

$$= \hat{D}\frac{N_v}{N_v - N_2}. \tag{2.15}$$

We equate $\hat{D}^{Miss}$ with the well-known average delay in an $M/M/N_{dr}^{(3)}$ queue (see, for example, [10, 9]) and solve for μ_3, i.e,

$$\hat{D}^{Miss} = \frac{1}{\mu_3} + \frac{P_Q}{N_{dr}^{(3)}\mu_3 - \lambda_3}, \tag{2.16}$$

where P_Q is given by ($\rho^{(3)}$ is the tertiary level utilization given by $\lambda_3/(N_{dr}^{(3)}\mu_3)$),

$$P_Q = p_0 \frac{(N_{dr}^{(3)}\rho^{(3)})^{N_{dr}^{(3)}}}{N_{dr}^{(3)}!(1-\rho^{(3)})}, \tag{2.17}$$

where

$$p_0 = \left[\sum_{n=0}^{N_{dr}^{(3)}-1} \frac{(N_{dr}^{(3)}\rho^{(3)})^n}{n!} + \frac{(N_{dr}^{(3)}\rho^{(3)})^{N_{dr}^{(3)}}}{N_{dr}^{(3)}!(1-\rho^{(3)})}\right]^{-1}. \tag{2.18}$$

While there is a closed-form expression for μ_3 when $N_{dr}^{(3)} = 1$, it has to be solved numerically when $N_{dr}^{(3)} \geq 2$. Given μ_3, we can get $B_3 = N_{dr}^{(3)} C_f \mu_3$.

If a file can be streamed once it starts being staged, the average miss delay then becomes, as opposed to Eq. 2.16,

$$\overline{D}^{Miss} = \frac{P_Q}{N_{dr}^{(3)}\mu_3 - \lambda_3}, \tag{2.19}$$

2.4.5 Performance characteristics

Given our model, we present here some other performance aspects of the hierarchical storage system.

- **Marginal and elasticity of substitution:** *The marginal substitution and the elasticity of substitution between B_3 and C_2 for file-by-file staging and complete staging before display (given a certain average start-up delay $\hat{D}$) are given by,*

  $$\begin{aligned} \frac{\partial B_3}{\partial C_2} &\approx -\frac{N_{dr}^{(3)} + \hat{D}\lambda_0}{\hat{D}N_v}, & (2.20)\\ \frac{\partial B_3/B_3}{\partial C_2/C_2} &\approx -\frac{N_2}{N_v - N_2}, & (2.21)\end{aligned}$$

  where $\hat{D}$ is the average delay requirement in the system.

  Marginal substitution refers to the rate at which two variables can be traded off with each other. In a hierarchical storage system, a higher tertiary bandwidth B_3 can generally be traded with a lower secondary storage C_2, and vice versa. The marginal substitution between these two variables is given by $\partial B_3/\partial C_2$.

  Note that in a system with low waiting time (our case of interest), user delay is comparable with the tertiary staging time. Since the tertiary level may be approximated by an M/M system, the delay is given by $\overline{T}_{M/M/N_{dr}^{(3)}}$ is approximately $N_{dr}^{(3)}\overline{T}_{M/M/1}$ with service rate $N_{dr}^{(3)}\mu_3$. If $\hat{D}$ is the average delay of the system, we have

  $$\begin{aligned} \hat{D} &\approx \frac{N_3}{N_v}\overline{T}_{M/M/N_{dr}^{(3)}} & (2.22)\\ &\approx \frac{N_3}{N_v}N_{dr}^{(3)}\overline{T}_{M/M/1} & (2.23)\\ &= \frac{N_3}{N_v}\frac{N_{dr}^{(3)}}{N_{dr}^{(3)}\mu_3 - \lambda_3}, & (2.24)\end{aligned}$$

which gives,

$$\mu_3 = \frac{1 + \hat{D}\lambda_0/N_{dr}^{(3)}}{\hat{D}(N_v/N_3)}. \tag{2.25}$$

With $B_3 = N_{dr}^{(3)} b_3 = N_{dr}^{(3)} C_f \mu_3$, we have

$$\begin{aligned} B_3 &\simeq \frac{C_f(N_{dr}^{(3)} + \hat{D}\lambda_0)}{\hat{D}N_v}(N_v - N_2) & (2.26)\\ &= \frac{N_{dr}^{(3)} + \hat{D}\lambda_0}{\hat{D}N_v}(C_f N_v - C_2). & (2.27)\end{aligned}$$

Therefore, $\partial B_3/\partial C_2$ is given in accordance to Eq. (2.20). For illustrative purpose, we show in Fig. 2.15 the trade-off between C_2 and B_3 to achieve a certain $\overline{D}_{st}$ (based on simulation) ($\lambda_0 = 30$ req./hr, $B_2 = 20$ MB/s, $N_{dr}^{(3)} = 1$, $N_v = 500$, and uniform video popularity). The linear region indicates the trade-off between C_2 and B_3 (little coupling between the two levels), with a slope given by $\partial C_2/\partial B_3$. There is a limit to such trade-off indicated by the horizontal asymptote due to conflict of resources (i.e., coupling between the two levels) — at low $\overline{D}_{st}$ (e.g., 1–2 minutes in the figure), such limit is due to insufficient B_2; while at higher $\overline{D}_{st}$ (e.g., more than 6 minutes in the figure), such limit is due to, as discussed earlier, the floor in $C_2 \approx (\lambda_0 T_h + n_2)C_f$. We therefore see that when the two levels are coupled (through either bandwidth or storage), inefficiency and low performance result.

We note here that, since B_2 and B_3 cannot be traded off with each other, we expect the trade-off curve between C_2 and B_2 would have similar trends. Indeed, this is the case as shown in Fig. 2.16 (based on simulation) (with $\lambda_0 = 30$ req./hr, $B_3 = 10$ MB/s, $N_v = 500$, $N_{dr}^{(3)} = 1$ and uniform video popularity). We show in Fig. 2.17 the influence of video popularity on the trade-off between C_2 and B_2 based on simulation (with $\lambda_0 = 30$ req./hr, $B_3 = 10$ MB/s, and $N_v = 500$). We have used the geometric video popularity with 80/20 popularity model, i.e., 80% of the requests ask for 20% of the videos (This gives $\sigma = 0.98405$). We see immediately that non-uniform video popularity is able to achieve the same delay requirement with markedly less storage and/or bandwidth.

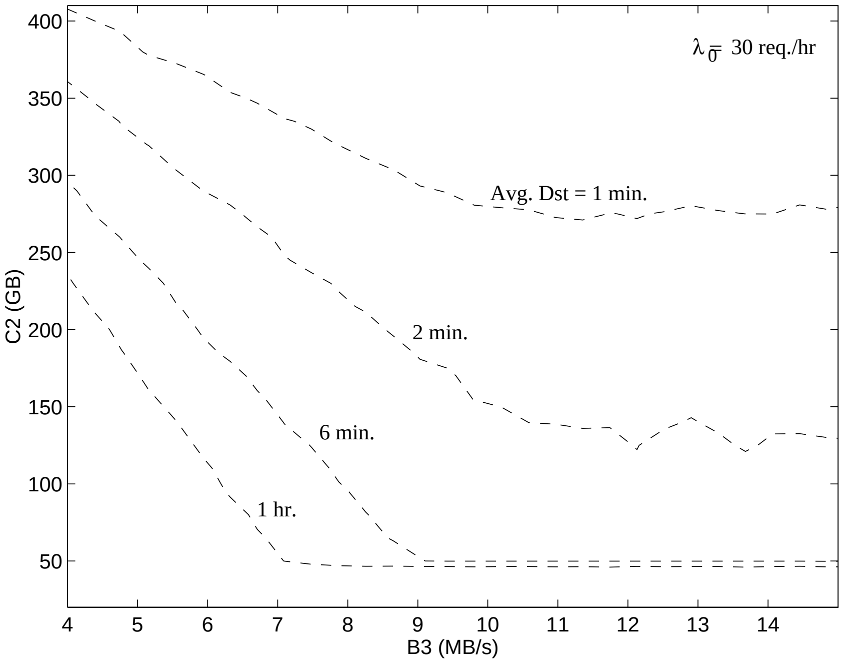


Figure 2.15: Trade-off between C_2 and B_3 to achieve a given $\overline{D}_{st}$ ($\lambda_0 = 30$ req./hr, $B_2 = 20$ MB/s, $N_v = 500$, $N_{dr}^{(3)} = 1$ and uniform video popularity)

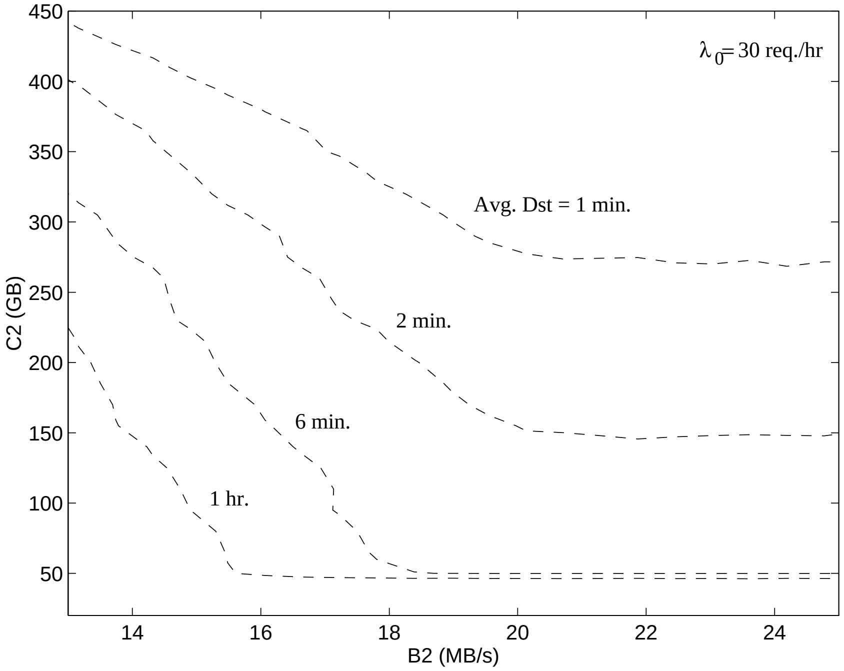


Figure 2.16: Trade-off between C_2 and B_2 to achieve a given $\overline{D}_{st}$ ($\lambda_0 = 30$ req./hr, $B_3 = 10$ MB/s, $N_v = 500$, $N_{dr}^{(3)} = 1$ and uniform video popularity)

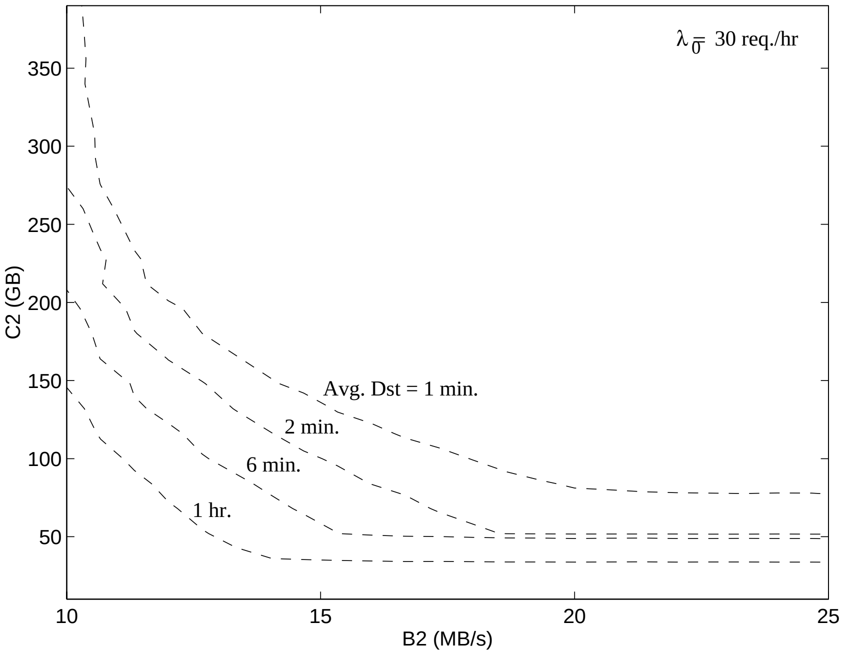


Figure 2.17: Trade-off between C_2 and B_2, with 80/20 video popularity ($\lambda_0 = 30$ req./hr, $B_3 = 10$ MB/s, $N_v = 500$)

The "elasticity" of substitution measures the rate of substitution between two variables in terms of their percentage change. For tertiary bandwidth and secondary secondary, such elasticity is given by $(\partial B_3/B_3)/(\partial C_2/C_2) = \partial \ln B_3/\partial \ln C_2$. Using Eq. (2.27), we get the elasticity of substitution according to Eq. (2.21). Note that since usually $N_2 \ll N_v$, the elasticity is low, meaning that a large percentage increase in C_2 only give a relatively small decrease in B_3. Therefore the optimal operating point for the hierarchical storage system is likely around the trade-off knee. This is evident by referring again to Fig. 2.15.

- **Performance invariance:** *Two hierarchical storage systems with the same user holding time have the same performance if the ratio of their parameters ($b_0(i)$, $C_f(i)$, $C_2(i)$,$B_2(i)$,$B_3(i)$) (where $i = 1, 2$ is the index corresponding to the respective system) scale as follows:*

$$\frac{b_0(2)}{b_0(1)} = \frac{C_f(2)}{C_f(1)} = \frac{C_2(2)}{C_2(1)} = \frac{B_2(2)}{B_2(1)} = \frac{B_3(2)}{B_3(1)}. \tag{2.28}$$

First note that if b_0 is scaled by a certain factor, the total secondary streaming bandwidth would then be similarly scaled. If both C_f and C_2 are scaled by the same factor, the miss probabilities will be the same in both systems. If C_f and B_3 are scaled by the same factor, μ_3 will remain the same and hence is the miss delay. Now if B_2 is also similarly scaled, both systems will have the same performance. Figure 2.18 shows $\overline{D}_{st}$ versus λ based on simulation, for two systems with b_0 changed from 1.5 Mb/s to 3 Mb/s and the other parameters C_f, C_2, B_2, and B_3 scaled accordingly (while maintaining the same holding time $T_h = 90$ minutes). Obviously the two systems have the same performance, showing that if the streaming bandwidth b_0 (and hence the file size) is changed, C_2, B_2, and B_3 all have to be changed by the same factor in order to maintain the performance.

Similarly, if T_h changes while b_0 remains unchanged, system performance will remain the same if $T_h(2)/T_h(1) = C_f(2)/C_f(1) = C_2(2)/C_2(1) = B_2(2)/B_2(1) =$

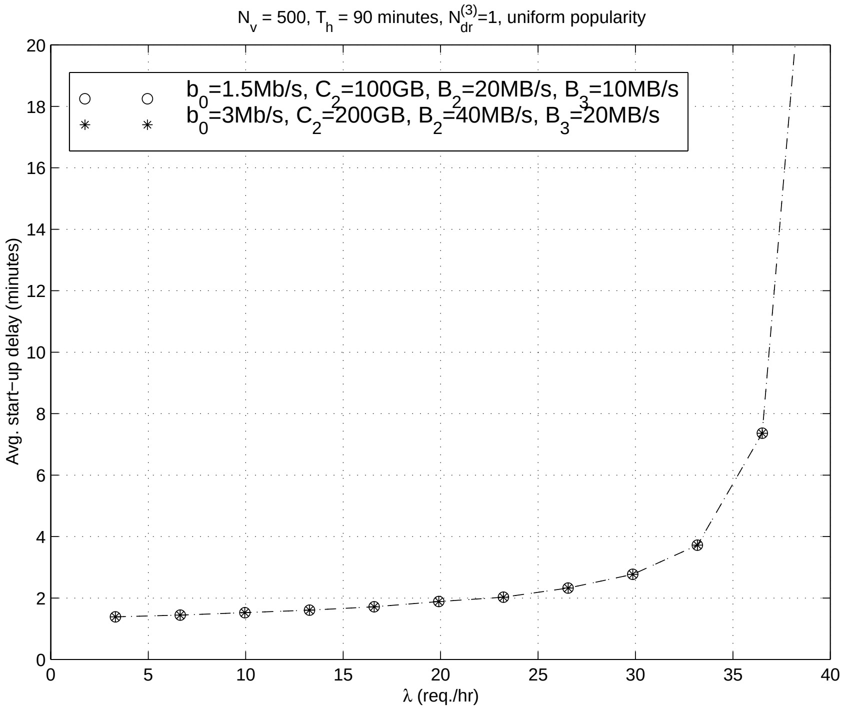


Figure 2.18: Change in b_0 (from 1.5 Mb/s to 3 Mb/s) in the performance of a hierarchical storage system (for $b_0 = 1.5$ Mb/s, we use $C_2 = 100$ GB, $B_2 = 20$ MB/s, $B_3 = 10$ MB/s, $N_{dr} = 1$, $N_v = 500$ and uniform video popularity.)

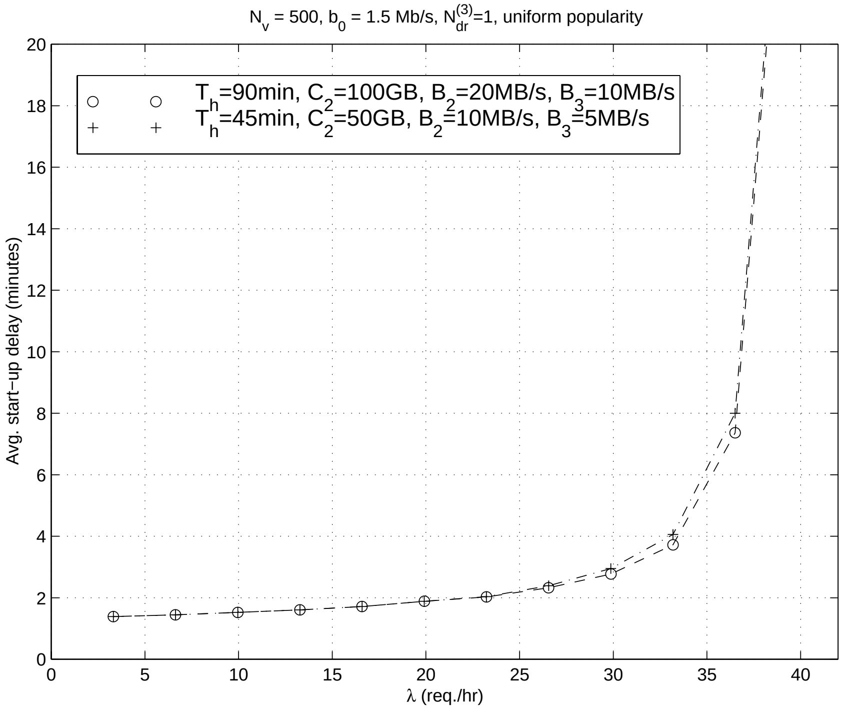


Figure 2.19: Performance of the hierarchical storage system as T_h is changed from 90 minutes to 45 minutes (for $T_h = 90$ minutes, we use $C_2 = 100$ GB, $B_2 = 20$ MB/s, $B_3 = 10$ MB/s, $N_{dr} = 1$, $N_v = 500$ and uniform video popularity)

$B_3(2)/B_3(1)$. We illustrate in Fig. 2.19 the performance invariance (based on simulation) when T_h is changed from 90 minutes to 45 minutes while C_f, C_2, B_2 and B_3 are scaled accordingly (with $b_0 = 1.5$ Mb/s).

- **Number of robotic arms in the tertiary level:** *In order to minimize the waiting time for a free robot, the number of robotic arms should satisfy,*

$$N_{rbt} \gg (1 - \alpha_2)\lambda_0 T_{ex}. \tag{2.29}$$

Note that the utilization of the robotic arms in the tertiary level is given by $\lambda_3 \overline{T}_{ex}/N_{rbt}$. Hence, in order for a staging request to find a free robot whenever there is an idle drive, we should have $\lambda_3 \overline{T}_{ex}/N_{rbt} \ll 1$, hence the condition given above.

2.5 Model Extensions and Design Examples

In this section, we extend our model to solve various design problems in a hierarchical storage system. The simple nature of our model (just two independent queues without computationally difficult procedures) greatly facilitates the design process of such a system. We also provide a number of design examples in this section.

2.5.1 Designing with different staging schemes

In this section, we illustrate how our model is used in specifying the architectural parameters so as to meet user delay requirement $\overline{D}_{st} \leq \hat{D}$. We consider here designing at the trade-off "knees," as these are the likely operating points if bandwidth and storage took on continuous values independently. We first consider the case of file-by-file staging without trail deletion, with complete staging before display, and then with stage-streaming (video streaming while being staged). Finally we consider how trail deletion affects the bandwidth and storage requirements.

Complete staging before display

Recall that if a file is completely staged before it is displayed, users would have complete freedom in interacting with the video from its beginning to its end. Figure 2.20 shows the required B_3 with respect to the target arrival rate λ_0. Here we show delay requirement $\hat{D} = 1$ minute and $\hat{D} = 3$ minutes, with $N_v = 500$ ($C_f = 1$ GB, $b_0 = 1.5$ Mb/s, $T_h = 1.5$ hr., $N_{dr}^{(3)} = 1$ (n_2 is chosen to be 15)). We see that the required B_3 increases rather linearly in λ_0. Though the required B_3 is higher with $\hat{D} = 1$ minute than with $\hat{D} = 3$ minutes, it is not three times as high.

We show in Table 2.3 the design to satisfy $\hat{D} = 2$ minutes. We consider two target arrival rates: $\lambda_0 = 20$ req./hr and $\lambda_0 = 50$ req./hr, and $N_v = 500$ (Other parameters are $C_f = 1$ GB, $T_h = 1.5$ hr., $b_0 = 1.5$ Mb/s, $N_{dr}^{(3)} = 1$, $n_2 = 15$). We see that there is "economy of scale" in terms of bandwidth and storage requirements — higher target arrival rate does

Table 2.3: Design specifications to achieve $\hat{D} = 2$ minutes ($C_f = 1$ GB, $b_0 = 1.5$ Mb/s, $T_h = 90$ minutes, $N_{dr}^{(3)} = 1$, $n_2 = 15$)

	$\lambda_0 = 20$ req./hr.	$\lambda_0 = 50$ req./hr
C_2 (GB)	45	90
α_3	0.91	0.82
λ_3 (req./hr)	18.2	41
$\hat{D}^{Miss}$ (minutes)	2.2	2.44
B_3 (MB/s)	12.63	18.21
B_2 (MB/s)	21.1	35.1

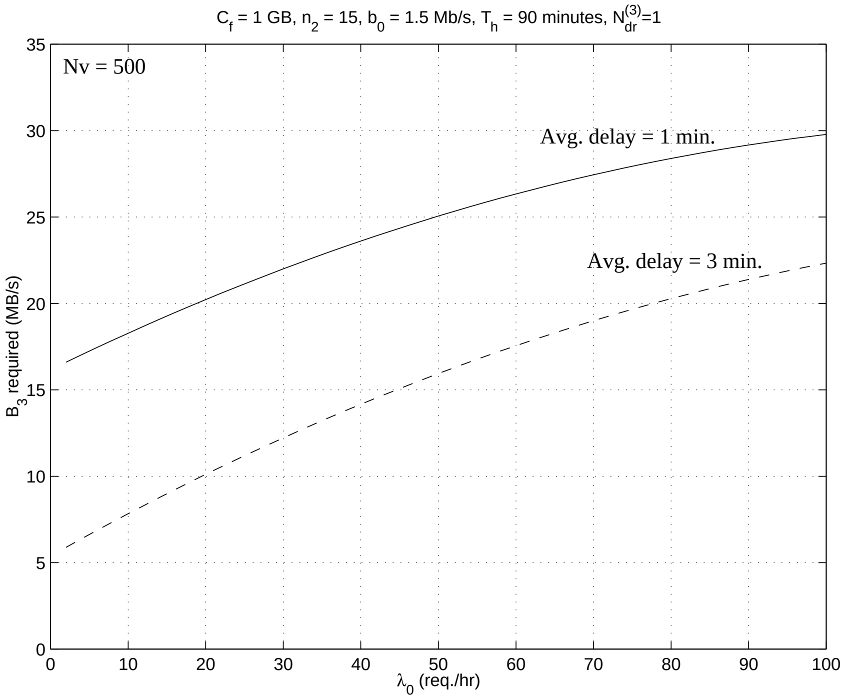


Figure 2.20: B_3 required to achieve $\hat{D} = 1$ minute and 3 minutes, with $N_v = 500$ ($C_f = 1$ GB, $b_0 = 1.5$ Mb/s, $T_h = 90$ minutes, $N_{dr}^{(3)} = 1$, $n_2 = 15$)

not lead to a proportional increase in bandwidth and storage requirements.

We show in Fig. 2.21 the simulation results using the parameters specified in Tab. 2.3. Clearly, the delay requirements at λ_0 are met, showing that our model is accurate. Note that the delay in fact is slightly lower than the requirement, since we have used the simpler M/M model instead of an M/D model to obtain the architectural parameters.

Deployed servers designed for a particular target arrival rate λ_0 may be used at a different rate. Figure 2.21 shows how such an "optimized" design would perform as the arrival rate changes. Though the server satisfies the delay requirement at the given arrival rate, the delay increases rapidly as the arrival rate becomes higher than the target designed rate. Therefore, in designing a hierarchical video server, it is important to design for the maximum arrival rate λ_0 in order to prevent unacceptable delays should arrival rate increases.

In Fig. 2.22 we compare our analytic model with simulation results, using the parameters

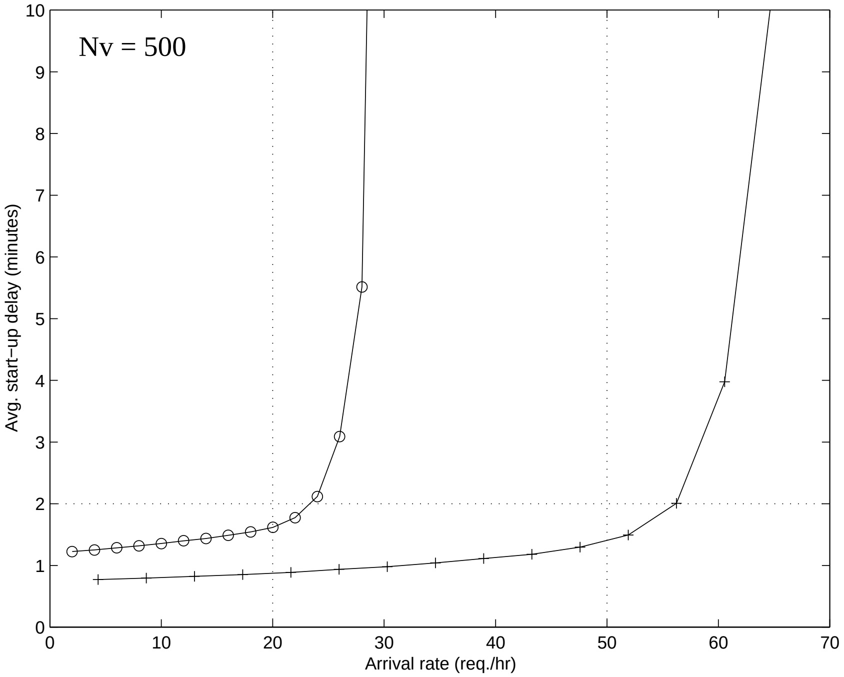


Figure 2.21: Server design satisfying $\hat{D} = 2$ minutes, with $\lambda_0 = 20$ req./hr and 50 req./hr ($N_v = 500$, $C_f = 1$ GB, $b_0 = 1.5$ Mb/s, $T_h = 90$ minutes, $N_{dr}^{(3)} = 1$, and $n_2 = 15$)

satisfying $\hat{D} = 2$ minutes ($N_v = 500$, $C_2 = 45$ GB, $B_2 = 21.1$ MB/s, $B_3 = 12.63$ MB/s, $C_f = 1$ GB, $b_0 = 1.5$ Mb/s, $T_h = 90$ minutes, $N_{dr}^{(3)} = 1$, and $n_2 = 15$). We show two analytic curves, one based on the M/M model (exponentially-distributed staging time) and another one based on the M/D model (deterministic staging time), with the total staging bandwidth available at any time given by $\min\{B_2 - \lambda T_h b_0, B_3\}$ (assuming negligible waiting time for a deletable file). Obviously, the M/D model is accurate up to the knee, beyond which the two levels are coupled and hence performance deteriorates very rapidly. We see that our decoupled model is indeed good up to λ_0. The performance of the M/M model is apparently a bit pessimistic up to the knee, beyond which the performance is optimistic (due to the decoupling assumption in the analysis).

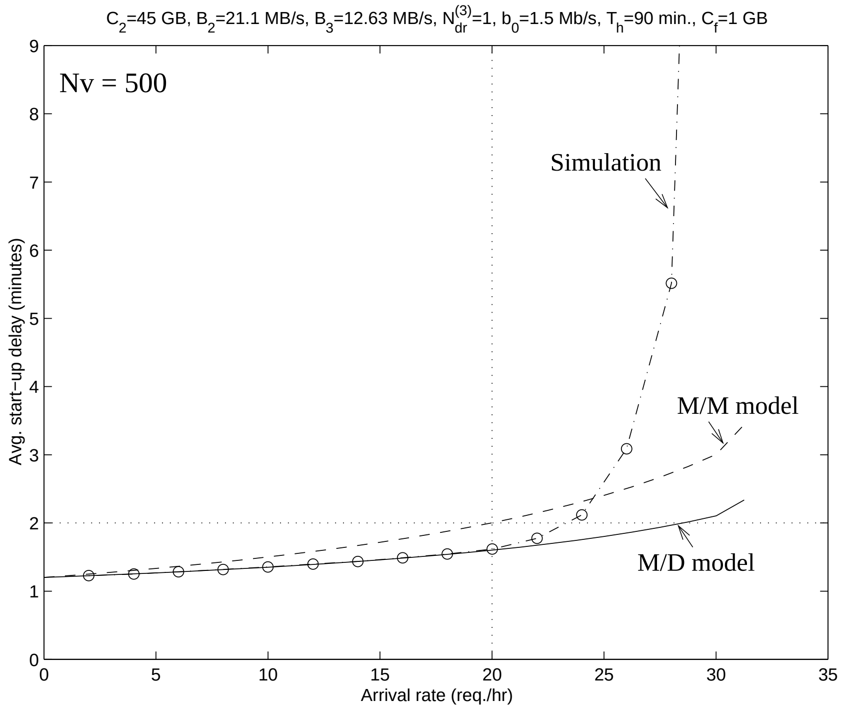


Figure 2.22: Simulation and analysis for a hierarchical storage system ($N_v = 500$, $C_2 = 45$ GB, $B_2 = 21.1$ MB/s, $B_3 = 12.63$ MB/s, $C_f = 1$ GB, $b_0 = 1.5$ Mb/s, $T_h = 90$ minutes, $N_{dr}^{(3)} = 1$, and $n_2 = 15$)

Stage-streaming operation

Here, a video is displayed once it starts being staged. To ensure streaming continuity, the total staged data of a displaying video at any time must be at least equal to the total streamed data up to that time (i.e., $b_3 \geq b_0$, generally satisfied with today's tertiary drive technology). If the staging rate is higher than the streaming rate, eventually the whole file would be staged before the video finishes displaying. Before the whole file is completely staged, a user can only jump/skip to any portion of the video file having been staged at that moment. Users entering the system finding their videos being staged or completely resident would enjoy no delay and better interactive capability. Misses in the system only suffer queueing delay (without staging delay) in the tertiary level.

We first note that the storage capacity in the secondary level is not different from the

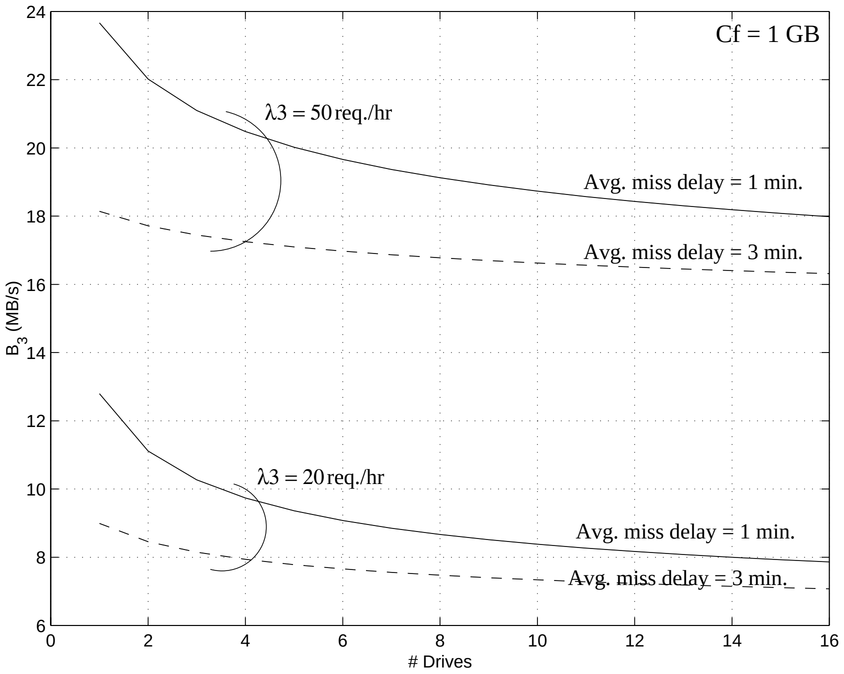


Figure 2.23: B_3 required as $N_{dr}^{(3)}$ increase ($C_f = 1$ GB)

case of complete staging before display, and $B_2 \approx (\lambda_0 T_h + n_2)b_0 + B_3$. The only difference is the required B_3.

We show in Fig. 2.23 the required B_3 as $N_{dr}^{(3)}$ increases, given $\hat{D}^{Miss} = 1$ minute and 3 minutes and $\lambda_3 = 20$ req./hr and 50 req./hr ($C_f = 1$ GB). Since miss delay is the queueing delay, we should operate the tertiary drives independently (as opposed to operate the drives in parallel for the case of complete staging before display). The saving in bandwidth in using independent drives is particularly marked for low delay requirement ($<$ 3 minutes). There is not much incremental benefit in increasing the number of drives beyond a certain point ($\simeq 8$).

We now compare the bandwidth requirement between the operations of complete staging before display and stage-streaming. We compare B_3 and B_2 required to meet $\hat{D} = 1$ minute in Figs. 2.24 and 2.25, respectively, as a function of the target arrival rate λ_0, given $N_{dr}^{(3)}$ ($N_v = 500$, $C_f = 1$ GB, $b_0 = 1.5$ Mb/s, and $T_h = 90$ minutes). Clearly, as λ_0 increases, the bandwidth requirements also increase (rather linearly). Stage-streaming has lower

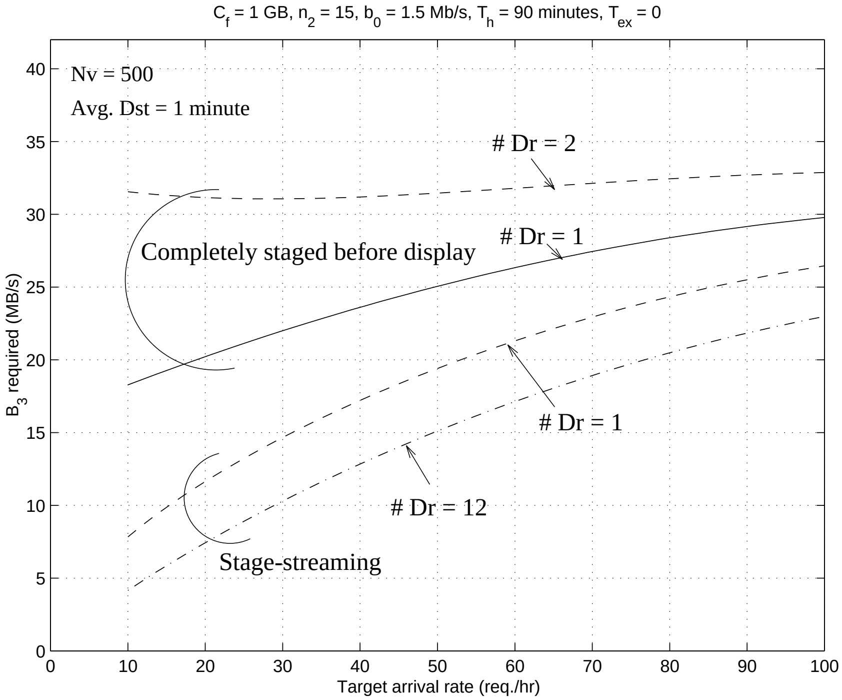


Figure 2.24: Comparison of B_3 requirement between the operations of "complete staging before display" and stage-streaming given $N_{dr}^{(3)}$, with $N_v = 500$ and $\hat{D} = 1$ minute, $C_f = 1$ GB, $b_0 = 1.5$ Mb/s, $T_h = 90$ minutes $(n_2 = 15)$

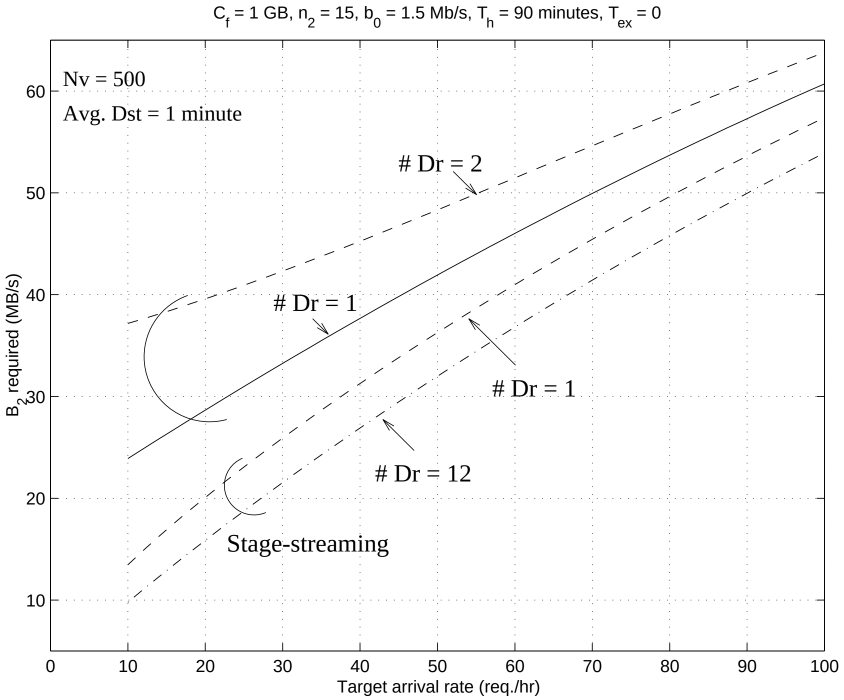


Figure 2.25: Comparison of B_2 requirement between the operations of "complete staging before display" and stage-streaming given $N_{dr}^{(3)}$, with $N_v = 500$ and $\hat{D} = 1$ minute, $C_f = 1$ GB, $b_0 = 1.5$ Mb/s, $T_h = 90$ minutes $(n_2 = 15)$

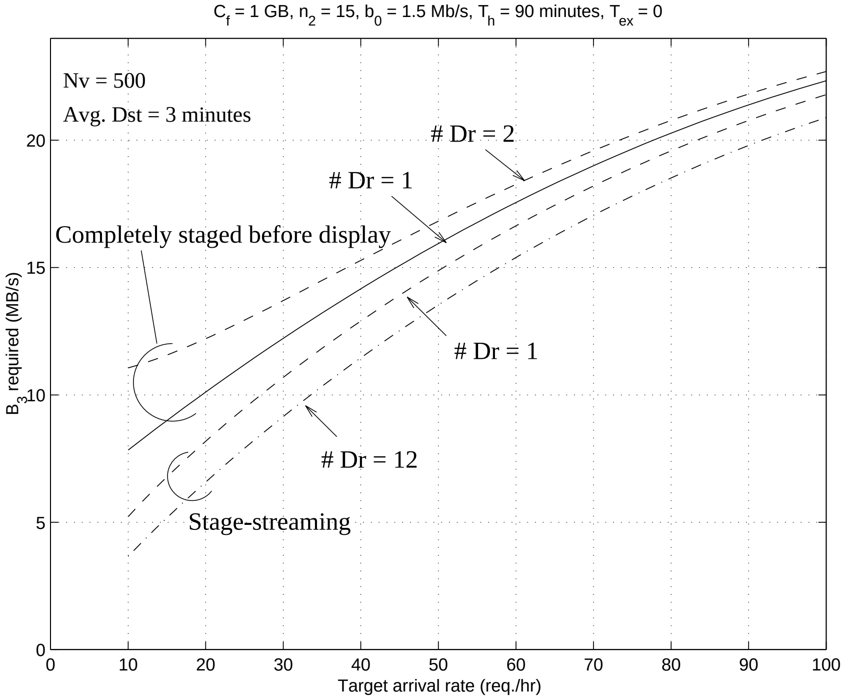


Figure 2.26: B_3 requirement for "complete staging before display" and stage-streaming, given $N_{dr}^{(3)}$ ($\hat{D} = 3$ minutes, $N_v = 500$, $C_f = 1$ GB, $b_0 = 1.5$ Mb/s, $T_h = 90$ minutes, $n_2 = 15$)

bandwidth requirement at the expense of some losses in user interactivity, and requires lower bandwidth as the number of tertiary drives increases.

We show in Figs. 2.26 and 2.27 the similar plots but with a higher delay requirement ($\hat{D} = 3$ minutes).

As $\hat{D}$ increases, the difference between the two schemes decreases. This is because the delay of the misses now mainly comes from the queueing time in the tertiary level, hence yielding the staging time of a file relatively insignificant in the total user delay.

Stage-streaming with trail deletion

We now consider stage-streaming with trail deletion, in which the streamed data in the secondary level is deleted; i.e., the secondary level keeps only the future data of a displaying video. (We see that if staging bandwidth for a video file is comparable to streaming

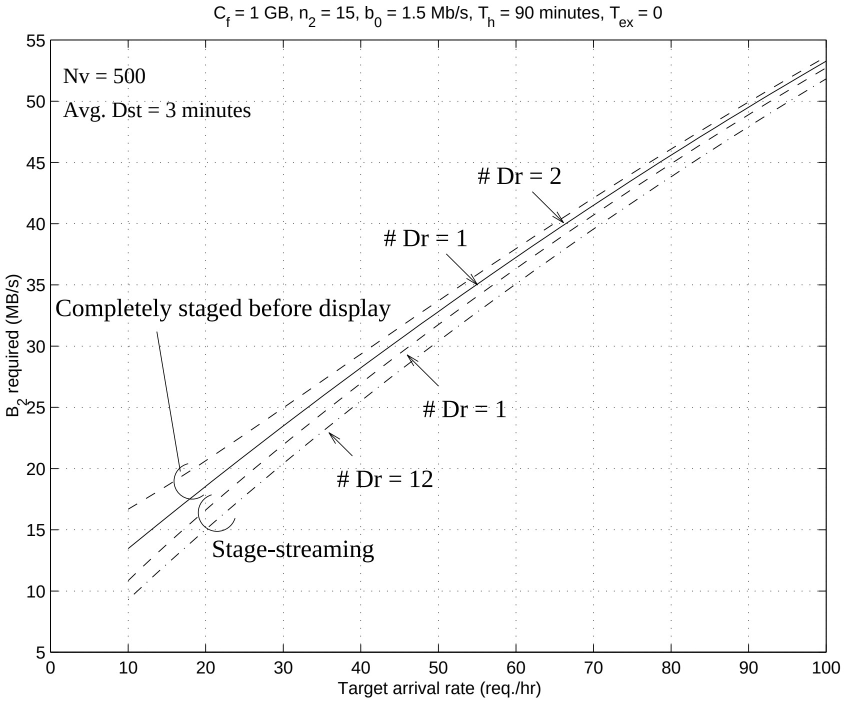


Figure 2.27: B_2 requirement for "complete staging before display" and stage-streaming given $N_{dr}^{(3)}$ ($\hat{D} = 3$ minutes, $C_2 = C_f(\lambda_0 T_h + n_2)$, $n_2 = 15$, $N_v = 500$, $C_f = 1$ GB, $b_0 = 1.5$ Mb/s, and $T_h = 90$ minutes)

bandwidth, the required C_2 would be very low). Users in the system hence can only interact with the video through start/stop or some very localized operations. This design can be applied in some VOD applications characterized by very low and localized user interactivities (e.g., movie-on-demand). As portions of video files are deleted once they are displayed, all requests are misses. We will see that trail deletion can greatly reduce the storage requirement of a video server (generally in the range of $\sim$50%, though a reduction of up to 100% is possible[5]), at the expense of additional bandwidth due to the higher miss rate.

Since all requests are misses, the arrival rate in the tertiary level λ_3 is the same as the external arrival rate,

$$\lambda_3 = \lambda_0. \tag{2.30}$$

As mentioned before, since λ_3 is higher than the previous cases, the required B_3 would be higher; while B_2 is still related to B_3 by $B_2 \approx B_3 + (\lambda_0 T_h + n_2) b_0$. We compare in Fig. 2.28 the required B_3 with and without trail deletion in order to achieve $\hat{D} = 1$ minute. Without trail deletion, the hit rate increases as λ_0 increases; and hence the required B_3 increase very sub-linearly with λ_0. However, with trail deletion, B_3 increases with λ_0 rather linearly. The difference in the requirements widens as λ_0 increases.

In order to find the required C_2, we need to know the process corresponding to the start-of-staging. We show in Fig. 2.29 the hierarchical storage system with $N_{dr}^{(3)}$ tertiary drives, the time line for various processes. Requests arrive in the tertiary system at rate λ_0. After queueing for some time, the video is loaded into a tertiary drive and starts to be staged (start of staging process) and displayed. After a certain time, the video will be completely staged and a new file can be loaded. The storage requirement of each video file in the process of being displayed is shown, with the aggregate sum of which at any time is the total secondary storage requirement at that time.

Note that the process of "start-of-staging" corresponds to service-entrance process in a multi-server queue. We simulate the "service-entrance" process in $M/D/R$ and $M/M/R$

[5] This corresponds to the case when the tertiary level directly streams videos to the users.

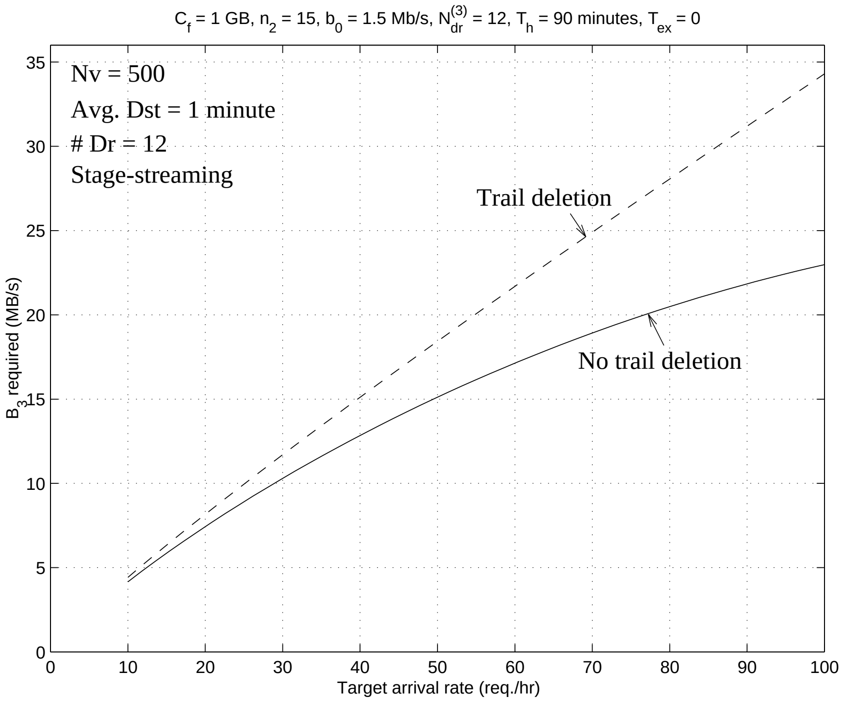


Figure 2.28: Comparison of B_3 requirement with and without trail deletion ($\hat{D} = 1$ minute, $N_v = 500$, $C_f = 1$ GB, $b_0 = 1.5$ Mb/s, $T_h = 90$ minutes, $n_2 = 15$)

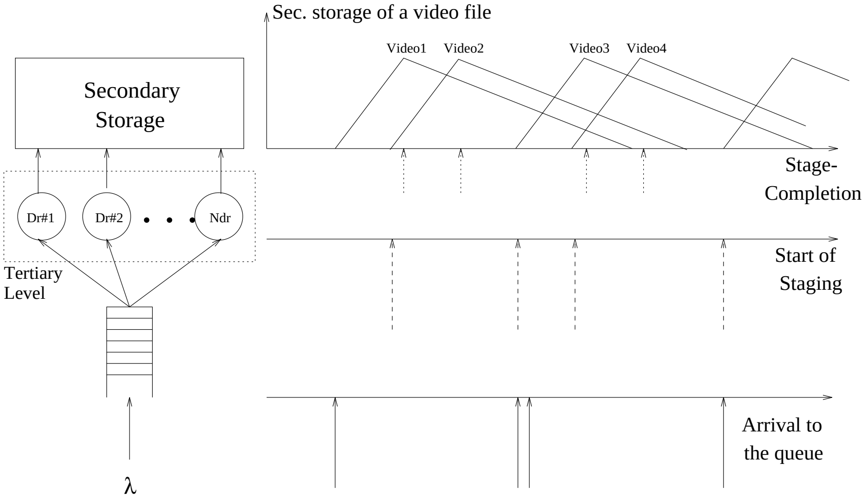


Figure 2.29: Operation of a stage-streaming with trail deletion

queues, with $R = 1$, 4, and 8 and $\rho \equiv \lambda_0/(R\mu) = 0.2$, 0.5, and 0.7. Let t be the inter-entrance time. If the service-entrance process is Poisson, then $P(t \leq T) = 1 - \exp(-\lambda_0 T)$, or $-\ln(1 - P(t \leq T)) = \lambda_0 T = \rho(TR\mu)$. In Figs. 2.30 and 2.31, we present the inter-

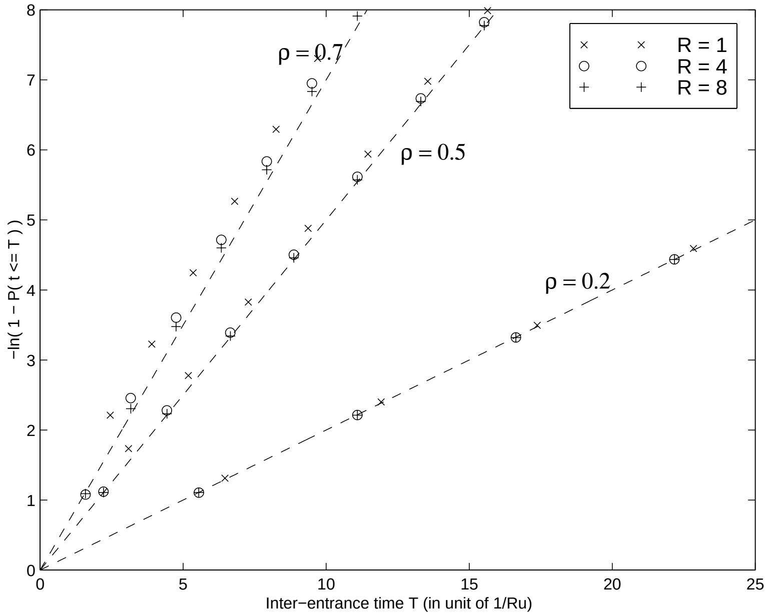


Figure 2.30: Service-entrance process in an $M/D/R$ queue

entrance time t (normalized to $1/(R\mu)$) based on simulation in discrete points, and the lines corresponding to a Poisson process with the same rate. We see that the inter-entrance process can be approximated by the Poisson process.

The storage requirement of the secondary level can then be found as follows. Figure 2.32 shows the storage occupied for each displaying video file with time. The storage of a file first increases with rate $(b_3 - b_0)$ for a time C_f/b_3 (when the whole file is delivered), then decreases with rate b_0 for a period $(1 - b_0/b_3)T_h$. The maximum storage space occupied by a displaying file is

$$C_{max} = \left(1 - \frac{b_0}{b_3}\right) C_f. \tag{2.31}$$

The total secondary storage required at any time t is the aggregate sum of the storage of all open files at time $[t - T_h, t]$. The number of open files in the time interval T_h is Poisson with rate λ_0.

Given that a start-of-staging event occurs in $[t - T_h, t]$, the time of its occurrence (the "start" time) T_s is uniformly distributed within the range. If C is the storage space of the

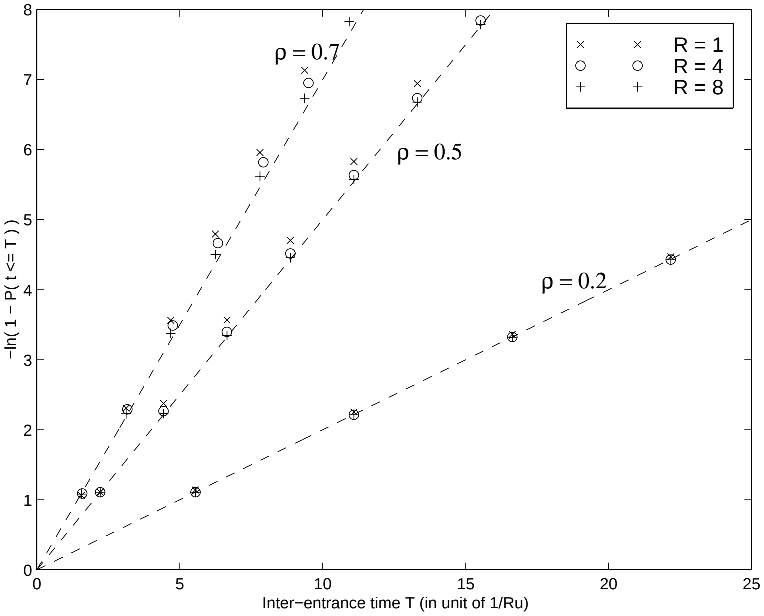


Figure 2.31: Service-entrance process in an $M/M/R$ queue

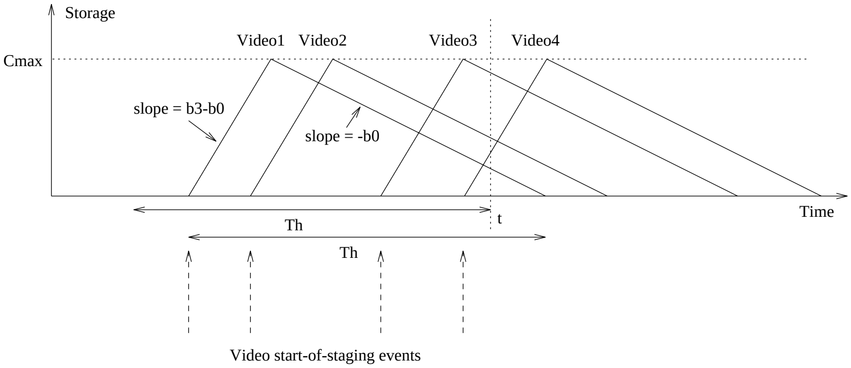


Figure 2.32: Storage requirement of any video file (from the start-of-staging till request departure) in an non-interactive server as a function of time

file occupied at time t, then

$$
\begin{aligned}
P(C \leq c) &= P\left(T_s \in [t - \frac{c}{b_3 - b_0}, t] \bigcup T_s \in [t - T_h, t - T_h + \frac{c}{b_0}]\right) && (2.32)\\
&= \frac{c}{(b_3 - b_0)T_h} + \frac{c}{b_0 T_h} && (2.33)\\
&= \frac{c}{C_{max}}. && (2.34)
\end{aligned}
$$

Therefore, given a staging event occurs within $[t - T_h, t]$, the storage of a video file at time t is uniformly distributed between 0 and C_{max}.

Let X be the total secondary storage space required at any time t. Then $X = \sum_{i=1}^{N} C(i)$, where $C(i)$ is the storage required for the ith opened file ($i = 1 \ldots N$). For large N ($\geq$ 5–10) (by central limit theorem),

$$
\begin{aligned}
\frac{1}{N} X \mid N &= \frac{1}{N} \sum_{i=1}^{N} C(i) \mid N && (2.35)\\
&\sim \mathcal{N}(\frac{C_{max}}{2}, \frac{C_{max}^2}{12N}), && (2.36)
\end{aligned}
$$

where $\mathcal{N}(\mu, \sigma^2)$ is a normal distribution with mean μ and variance σ^2. Therefore,

$$
\begin{aligned}
P(X \geq C_2) &= P(\sum_{i=1}^{N} C(i) \geq C_2) && (2.37)\\
&= \sum_{n=1}^{\infty} P(\sum_{i=1}^{n} C(i) \geq C_2 \mid N = n) P(N = n) && (2.38)\\
&\approx \sum_{i=1}^{\infty} \left\{ 1 - \Phi\left(\frac{C_2 - C_{max} n/2}{C_{max}\sqrt{n/12}} \right) \right\} \frac{(\lambda_0 T_h)^n}{n!} e^{-\lambda_0 T_h} && (2.39)\\
&= \sum_{i=1}^{\infty} \left\{ 1 - \Phi\left(\frac{C_2/C_{max} - n/2}{\sqrt{n/12}} \right) \right\} \frac{(\lambda_0 T_h)^n}{n!} e^{-\lambda_0 T_h}, && (2.40)
\end{aligned}
$$

where

$$
\Phi(x) \triangleq \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\xi^2/2} d\xi, \qquad (2.41)
$$

is the cumulative Gaussian distribution.

If C_2 is the secondary storage capacity, $P(X \geq C_2)$ is the probability of running out of storage space. We show in Fig. 2.33 such probability with respect to C_2 (normalized to C_{max}), given λ_0 ($T_h = 1.5$ hour). From the graph, we see that to keep the probability low

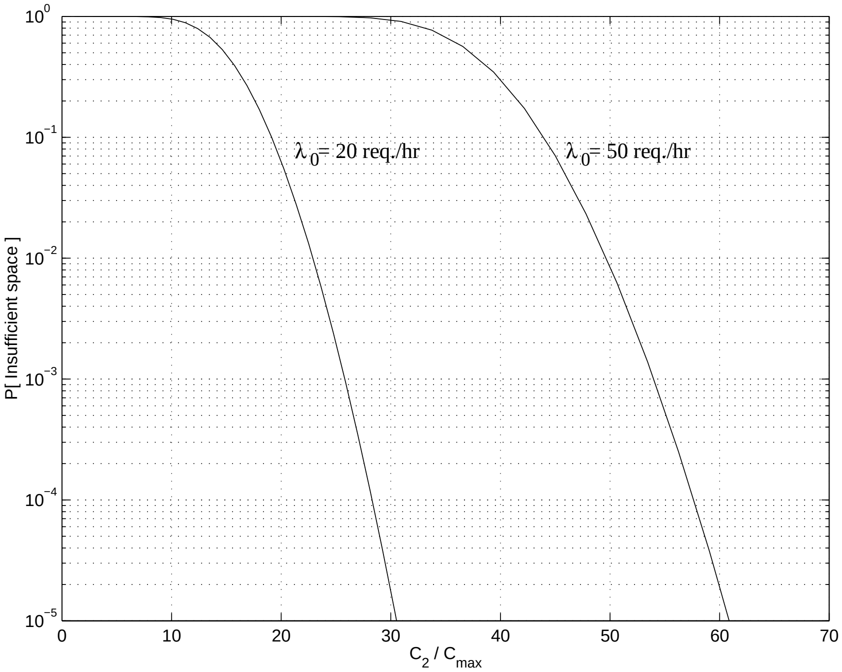


Figure 2.33: Probability of insufficient space in a non-interactive server if the secondary storage space is C_2, for $\lambda_0 = 20$ req./hr and 50 req./hr, and $T_h = 90$ minutes

(i.e., 10^{-2} – 10^{-4}), we need $C_2 \approx \beta(\lambda_0 T_h)C_{max}$, where $\beta \simeq 0.7$–1. Hence,

$$\begin{aligned} C_2 &= (\beta\lambda_0 T_h)C_{max} \\ &= (\beta\lambda_0 T_h)(1 - \frac{b_0}{b_3})C_f, \end{aligned} \tag{2.42}$$

and hence $C_2 \approx 0.5(\lambda_0 T_h C_f)$. For the case of $b_3 \approx 2b_0$, more than 50% reduction in C_2 as compared with the case of no trail deletion is possible.

2.5.2 Design examples given storage and bandwidth relationship

We have shown in Fig. 2.16 the trade-off between C_2 and B_2 to achieve a certain delay requirement (complete staging before display). From the figure we see that given a certain target arrival rate and B_3, one can generally find a range of values (B_2, C_2) in order to meet a certain average delay requirement. If disk capacity and disk bandwidth can take on continuous values independently, a hierarchical storage system would be likely operated around the trade-off knee. Current storage technology, however, can only allow some feasible combinations of bandwidth and storage. In this section, we will show how specific parameter values can be chosen given current disk technology. We will see that such consideration can lead to a design away from the trade-off knee, leading to some excess resources.

A magnetic disk comes with its storage capacity C_{dsk} and a certain effective disk bandwidth B_{dsk}. Several magnetic disks can be put together to achieve a higher total capacity, while at the same time increasing the total bandwidth (e.g., as in disk array). If we let the total storage and bandwidth be proportional to the number of disks, and let B_2 be the total effective bandwidth required in the secondary level, we then have the following "disk technology" relationship:

$$C_2 = \left(\frac{B_2}{B_{dsk}}\right) C_{dsk}. \tag{2.43}$$

Consider that $B_{dsk} = 2$ MB/s, and given that currently $C_{dsk} \leq 10$ GB, we therefore have $C_2 \leq (B_2/2)10 = 5B_2$. The line is shown as the "disk technology" line in Figure

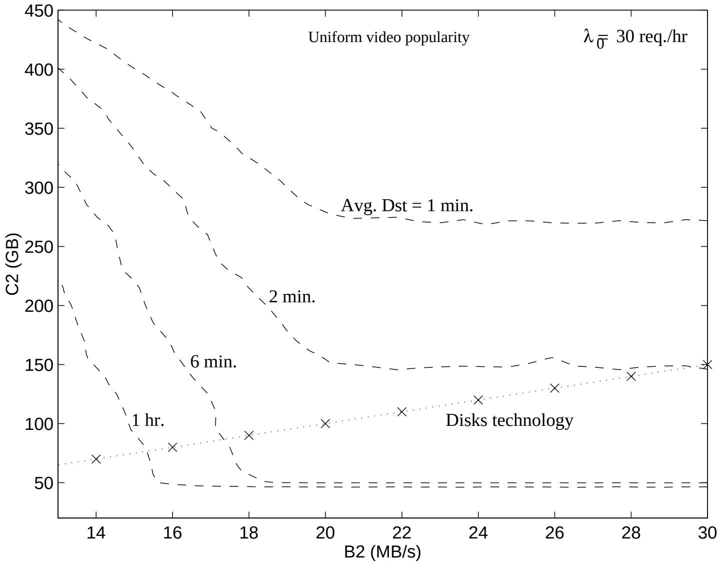


Figure 2.34: Trade-off between C_2 and B_2 with the "disk technology" line included ($\lambda_0 = 30$ req./hr, $B_3 = 10$ MB/s, $N_v = 500$ and uniform video popularity)

2.34, reproduced from Fig. 2.16. The region below the line is the current "technologically feasible" region. Clearly, we see that the feasible operating points of a server are not necessarily around the knees of the trade-off curves, after taking into account our current storage technology.

We give examples of how a hierarchical storage system can be designed to satisfy a certain delay requirement, say $\hat{D} = 2$ minutes or even 30 seconds, under arrival rate $\lambda_0 = 30$ req./hr. We show in Figure 2.35 specifically the delay contours achieving the same $\overline{D}_{st} = \hat{D}$. We have also considered that each tertiary drive comes with $b_3 = 2$ MB/s, and we show in the figure three different values of B_3: 8 MB/s, 10 MB/s and 12 MB/s (corresponding to 4, 5, and 6 drives, respectively).

From the interception between the "disk technology" line and the delay contours, we can deduce the requirements of the secondary level in order to satisfy the average delay requirement. Such requirements in terms of storage capacity and bandwidth are shown in Tab. 2.4. For the case of $\hat{D} = 2$ minutes and $B_3 = 10$ MB/s, since the "technology" line intercepts the delay contour well into the horizontal asymptote region at $(B_2, C_2) = (30$

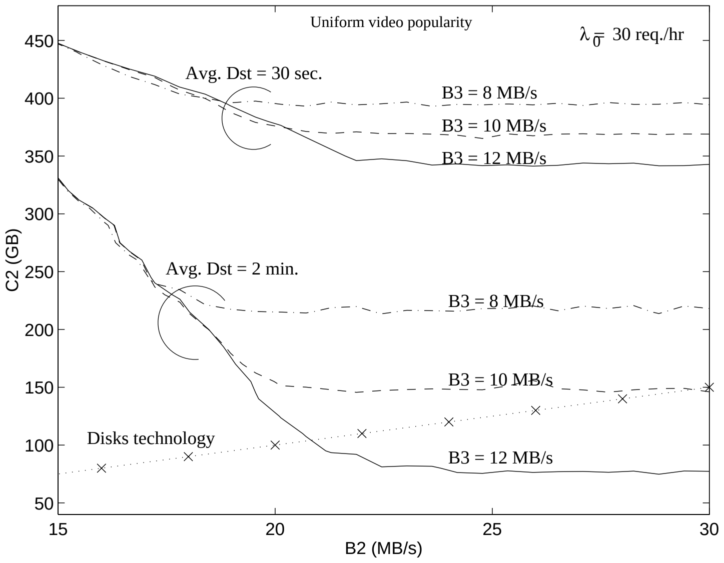


Figure 2.35: The design of a hierarchical storage system with $\overline{D}_{st} = 30$ seconds and 2 minutes, with $\lambda_0 = 30$ req./hr ($N_v = 500$ and uniform video popularity)

MB/s, 150 GB), we have a lot of excess B_2. We also see that if B_3 is increased to 12 MB/s, the operating point becomes (B_2, C_2) = (22 MB/s, 80 GB), saving a total of 4 disks. To decide between these two points, we have to take into consideration the cost of a tertiary drive. If its cost is less than 4 magnetic disks, the second points would be of lower system cost and would be the choice.

Note that given our current technology, the hierarchical storage system can hardly satisfy $\hat{D} = 30$ seconds without incurring excessive storage and bandwidth.

How does non-uniform video popularity affect our design? Figure 2.36 shows the similar plot as Fig. 2.35, but with non-uniform video popularity (80/20 geometric video popularity). From the figure, we can obtain the storage and bandwidth requirements as shown in Tab. 2.4, in which the resources in excess (not at the knee) are highlighted.

From the table, we see that non-uniform video popularity can tremendously reduce the requirements in both B_2 and C_2 compared with the uniform case. While designing a hierarchical storage system with 30-seconds delay under uniform video popularity is almost impossible, it is very feasible with non-uniform popularity. We see that when B_3 = 12

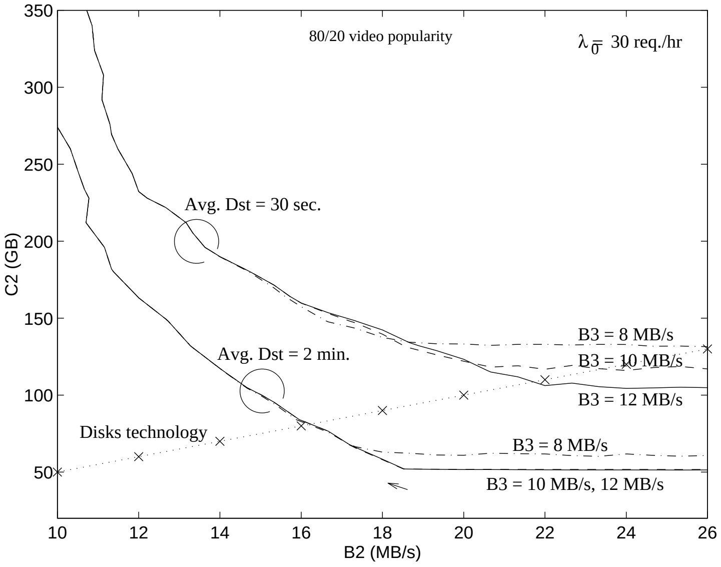


Figure 2.36: Trade-off between C_2 and B_2, with 80/20 video popularity ($\lambda_0 = 30$ req./hr and $N_v = 500$)

Table 2.4: Storage and bandwidth requirements for a server under a certain delay constraint constraint ($\lambda_0 = 30$ req./hr, $B_3 = 10$ MB/s, and $N_v = 500$)

$\overline{D}_{st}$	B_3	Uniform popularity				80/20 video popularity			
		B_2 (MB/s)	C_2 (GB)	#disks	GB/disk	B_2 (MB/s)	C_2 (GB)	#disks	GB/disk
30 sec.	8	*78*	390	39	10	*26*	130	13	10
	10	*72*	360	36	10	*24*	115	12	9.6
	12	*68*	340	34	10	22	105	11	9.55
2 min.	8	*44*	220	22	10	18	60	9	6.7
	10	*30*	150	15	10	18	55	9	6.1
	12	22	85	11	7.73	18	≈55	9	6.1

MB/s, the C_2 necessary with non-uniform video popularity is only about 20% of the total storage. When $\overline{D}_{st}$ is increased to 2 minutes, under non-uniform popularity, we see that there is no much gain in C_2 and B_2 as B_3 is increased from 10 MB/s to 12 MB/s. Therefore, using $B_3 = 10$ MB/s is sufficient.

2.5.3 Non-uniform video popularity

Our model can be extended to design a hierarchical storage system with multiple video popularity classes, and within each class video popularity is uniform. Let A be the number of popularity classes, each of which have $N_{v,a}$ videos, where $1 \leq a \leq A$. Obviously, the total number of videos is,

$$N_v = \sum_{a=1}^{A} N_{v,a}. \tag{2.44}$$

Let P_a be the probability of a random arrival choosing popularity class a, $1 \leq a \leq A$ (the request rate for each class is hence $\lambda_a = P_a \lambda_0$). We have $\sum_a P_a = 1$ and the popularity of the ith movie in class a is $p_{i,a} = P_a / N_{v,a}$, $1 \leq i \leq N_{v,a}$.

The design of such a hierarchical storage system follows the same steps as given in the previous section. For file-by-file staging without trail deletion, C_2 required is still $(\lambda_0 T_h + n_2) C_f$. Since the extra n_2 space in the secondary level is shared among the popularity classes, the hit probability for class a $(1 \leq a \leq A)$ can be estimated as,

$$\alpha_{2,a} = \frac{\lambda_a T_h + P_a n_2}{N_{v,a}}. \tag{2.45}$$

Therefore overall miss probability is,

$$\alpha_3 = \sum_{a=1}^{A} P_a (1 - \alpha_{2,a}), \tag{2.46}$$

and the arrival rate to the tertiary level is given by $\lambda_3 = \alpha_3 \lambda_0$. The required B_2 is given by $B_3 + (\lambda_0 T_h + n_2) b_0$.

As an example, let's design a server meeting an average delay of $\hat{D} = 2$ minutes with complete staging before display, for $\lambda_0 = 20$ req./hr or 50 req./hr. There are $A = 3$ popularity classes, with class popularities $P_1 = 0.05$ (unpopular movies), $P_2 = 0.25$, and

Table 2.5: Design specifications to achieve $\hat{D} = 2$ minutes, with $A = 3$ popularity classes ($C_f = 1$ GB, $b_0 = 1.5$ Mb/s, $T_h = 90$ minutes, $N_{dr}^{(3)} = 1$ and $n_2 = 15$)

	$N_v = 500$	
	$\lambda_0 = 20$ req./hr.	$\lambda_0 = 50$ req./hr
C_2 (GB)	45	90
$\alpha_{2,1}$	0.015	0.03
$\alpha_{2,2}$	0.05625	0.1125
$\alpha_{2,3}$	0.21	0.42
α_3	0.8382	0.6764
λ_3 (req./hr)	16.76	33.82
$\hat{D}^{Miss}$ (minutes)	2.39	2.96
B_3 (MB/s)	11.64	15.03
B_2 (MB/s)	20.07	31.9

$P_3 = 0.7$ (popular movies), and number of movies in each class being $N_{v,1} = 150$, $N_{v,2} = 200$, and $N_{v,3} = 150$ ($N_v = 500$). Table 2.5 shows the bandwidth and storage requirements. Note that the requirements in Tab. 2.5 is not much different from the uniform popularity case (ref. Tab. 2.3). This is expected for large number of video files since each arrival would likely request a different movie to be displayed. We present in Fig. 2.37 the system delay versus λ given the above specifications, showing that the performance goal is met at λ_0.

2.5.4 Bandwidth partitioning

So far, we have considered a system in which B_2 is shared between streaming and staging and hits are scheduled at a higher priority than misses. Therefore we found that hits enjoy negligible delay and hence the delay comes from misses.

If B_2 is partitioned (i.e., a fraction of B_2 is used solely for staging purpose, while

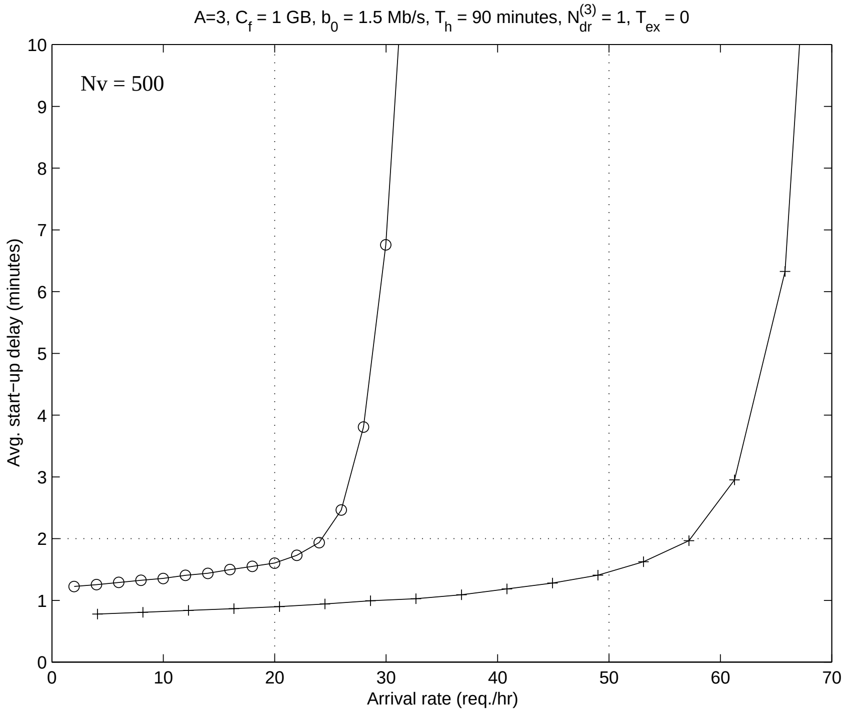


Figure 2.37: Simulation results of designing a hierarchical storage system with non-uniform video popularity ($\hat{D} = 2$ minutes, $\lambda_0 = 20$ req./hr and 50 req./hr, $A = 3$, $N_v = 500$, $C_f = 1$ GB, $b_0 = 1.5$ Mb/s, $T_h = 90$ minutes, and $N_{dr}^{(3)} = 1$)

the remaining is used solely for streaming), hits may now suffer delay due to insufficient bandwidth allocated to streaming. In fact, to satisfy an average overall delay requirement of $\overline{D}_{st} \leq \hat{D}$, we may partition the bandwidth so that the start-up delay for hits is $\beta\hat{D}$, where $0 \leq \beta \leq 1$, and the miss delay in the tertiary level is $\overline{D}^{Miss} = (1-\beta)\hat{D}/\alpha_3$ (so that the overall delay is still $\hat{D}$). Since storage requirement is not expected to be much different, the question then becomes how the delay requirement should be "partitioned" between the hits and misses so as to minimize the bandwidth requirements.

We show here that in fact to minimize the bandwidth requirements, it is most likely that hits should enjoy minimal delay. Recall that (for $N_{dr}^{(3)} = 1$),

$$B_3 \;=\; C_f\left(\frac{1}{\overline{D}^{Miss}} + \lambda_3\right)$$

$$
\begin{aligned}
&= C_f \left(\frac{\alpha_3}{\hat{D} - \overline{D}_{st}^{Hit}} + \lambda_3 \right) \\
&= \alpha_3 C_f \left(\frac{1}{(1-\beta)\hat{D}} + \lambda_0 \right), \qquad (2.47)
\end{aligned}
$$

and hence,

$$
\frac{\partial B_3}{\partial \beta} = \frac{\alpha_3 C_f}{\hat{D}(1-\beta)^2}. \qquad (2.48)
$$

Therefore, as β (corresponding to the hit delay) increases, B_3 increases, first quite slowly and then very quickly when β approaches 1. This point is illustrated in Fig. 2.38, in which we plot the B_3 required as a function of β given $\hat{D}$ ($N_v = 500$ and $\lambda_0 = 50$ req./hr ($C_f = 1$ GB, $\lambda_0 = 50$ req./hr, $n_2 = 15$, $N_{dr}^{(3)} = 1$, and $b_0 = 1.5$ Mb/s). Note that an increase of a minute or two in the hit delay ($\beta = 0.5$) leads to a saving of about 10 streams (ref. Fig. 2.11), corresponding to about $n_2 b_0 = 2$ MB/s. For $\hat{D} = 2$ minutes, this is lower than the increase in B_3 (which is about 5 MB/s in this case). Therefore, both B_3 and B_2 requirements are increased when β is increased in this case. Due to the steep increase in B_3 with respect to β, the hierarchical storage system with partitioned B_2 would be likely designed with little hit delay (with miss delay meeting the delay requirement). As $\hat{D}$ is increased, the increase in B_3 is no longer as steep and hence a partitioned B_2 with non-negligible hit delay may lead to a lower B_2 (but with higher B_3).

2.5.5 A hierarchical storage system for distributed video services

In order to serve a large number of users and further increase the storage capacity in the system, video system can consist of multiple storage and streaming nodes as shown in Fig. 2.39. Our model developed can be used to design such a distributed video system in which S video servers serving local communities are connected through a switch or network to L sites, where files are stored. A video file to be displayed can be obtained from one of the L sites. The local servers cache and stream the file to their respective users.

We want to specify the bandwidth and storage required to meet a certain user delay requirement given a target aggregate arrival rate λ_0. Note that since out of the request

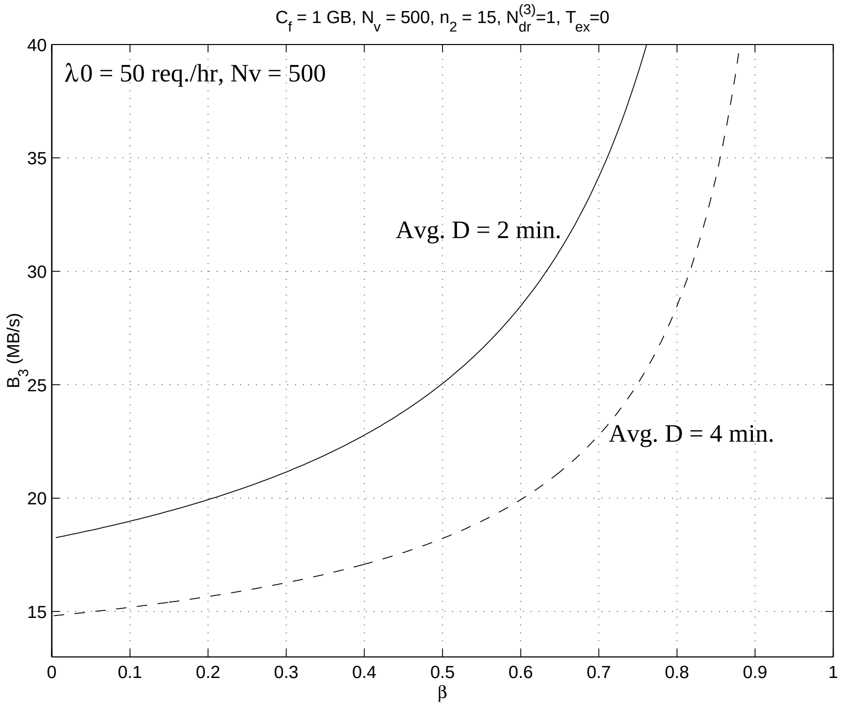


Figure 2.38: B_3 requirement vs. partitioned hit delay with $\hat{D} = 2$ minutes ($C_f = 1$ GB, $\lambda_0 = 50$ req./hr, $n_2 = 15$, $N_{dr}^{(3)} = 1$)

rate in a library only a small fraction comes from a particular local server, using bandwidth partitioning for each library and local server pair would not be economical. By the same token, this also holds true for a local server: since it is unlikely for all the libraries concurrently deliver files to a particular local server, partitioning bandwidth for each library in a local server is not economical. Hence the library bandwidth should be shared among the local servers, and the server bandwidth should be shared among all the libraries.

This is best illustrated by an example. Consider the design of a distributed video system with $\overline{D}_{st} \leq \hat{D} = 2$ minutes, $\lambda_0 = 200$ req./hr uniformly distributed among all the local servers, $L = 8$, $S = 10$, and $N_v = 4000$ titles (uniform popularity) with each permanent site storing 500 video titles ($C_f = 1$ GB, $b_0 = 1.5$ Mb/s and $T_h = 90$ minutes, $N_{dr}^{(3)} = 1$, and negligible exchange time). Let's consider complete staging before display. Since each local server stores $(\lambda_0/S)T_h + n_2 = 45$ GB (yielding the required miss delay 2.02 minutes), the local miss request rate is 19.78 req./hr and hence the corresponding request rate to

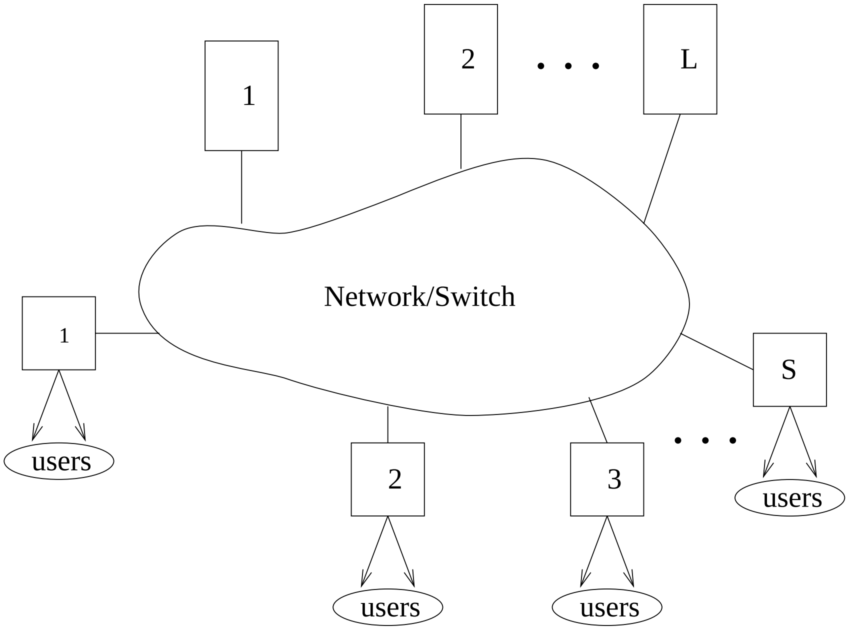


Figure 2.39: A distributed hierarchical storage system with multiple video servers and video permanent sites

a library from a local server is 2.47 req./hr. Since the average miss delay should be 2.02 minutes, if the library bandwidth were partitioned among the local the total required library bandwidth is 89.3 MB/s! This is very large, and since the bandwidth is only less than 10% utilized, much bandwidth is wasted. If a local server partitions its staging bandwidth for each library, the total staging bandwidth needed locally is 71.4 MB/s, and again such bandwidth is less than 10% utilized!

We hence should consider shared bandwidth, in which the library bandwidth is shared among the local servers, and vice versa. We summarize the important notations in this section in Tab. 2.6.

Denote $B_2^{(s)}$ and $C_2^{(s)}$ the secondary bandwidth and storage for server s ($s = 1 \ldots S$), and $B_3^{(l)}$ the bandwidth for site l ($l = 1 \ldots L$). Let $p_s \lambda_0$ be the request rate for server s, with $\sum_{s=1}^{S} p_s = 1$. Further let $\alpha_2^{(s)}$ be the hit probability for server s. With trail deletion, $\alpha_2^{(s)} = 0$; while without trail deletion, we have

$$\alpha_2^{(s)} = \frac{n_2 + T_h \lambda_0 p_s}{N_v} \tag{2.49}$$

Table 2.6: Important variables considered in the distributed server environment

Variables		Remarks
L	$=$	Number of libraries in the system, indexed $l = 1, \ldots, L$
S	$=$	Number of local servers in the system, indexed from $s = 1, \ldots, S$
λ_0	$=$	Aggregate target arrival rate in the system
$\lambda_3^{(l)}$	$=$	Aggregate request rate for library l
p_s	$=$	Fraction of the total arrival in local server s
$\alpha_2^{(s)}$	$=$	Hit rate for local server s
$\eta_l^{(s)}$	$=$	Fraction of misses in the local server s directed to library l
$\delta_l^{(s)}$	$=$	Fraction of staging files in library l for local server s (related to the above parameters through Eq. (2.54))
$\sigma^{(l)}$	$=$	Utilization of bandwidth for library l
$C_2^{(s)}$	$=$	Total storage in server s
$B_2^{(s)}$	$=$	Total bandwidth in server s
$B_3^{(l)}$	$=$	Total bandwidth in library l

$$= a + bp_s, \tag{2.50}$$

where $a \triangleq n_2/N_v \ll 1$, and $b \triangleq \lambda_0 T_h/N_v \ll 1$. Let $\eta_l^{(s)}$ be the fraction of misses in server s requesting a file from site l. Obviously, $\sum_{l=1}^{L} \eta_l^{(s)} = 1$.

The design procedure of the system is as follows. Let $\overline{D}_l^{Miss}$ be the average delay for site l. The average delay for all misses is then

$$\overline{D}^{Miss} = \frac{\sum_{s=1}^{S} \sum_{l=1}^{L} p_s (1 - \alpha_2^{(s)}) \eta_l^{(s)} \overline{D}_l^{Miss}}{\sum_{s=1}^{S} p_s (1 - \alpha_2^{(s)})}. \tag{2.51}$$

As hit delay is negligible, the average start-up delay in server s is hence given by $\overline{D}_{st}^{(s)} \simeq (1 - \alpha_2^{(s)})\overline{D}^{Miss}$, and the average overall delay is $\sum_{s=1}^{S} \sum_{l=1}^{L} p_s (1 - \alpha_2^{(s)}) \eta_l^{(s)} \overline{D}_l^{Miss}$.

Let $\lambda_3^{(l)}$ be the rate of arrival of staging requests in the permanent site l given by,

$$\lambda_3^{(l)} = \sum_{s=1}^{S} \eta_l^{(s)} (1 - \alpha_2^{(s)})(p_s \lambda_0). \tag{2.52}$$

The aggregate staging rate is given by $\lambda_3 = \sum_{l=1}^{L} \lambda_3^{(l)}$. Define $\delta_l^{(s)}$ as the fraction of staging files in site l delivered to server s. We have

$$\begin{aligned} \delta_l^{(s)} &= \frac{\eta_l^{(s)} (1 - \alpha_2^{(s)})(p_s \lambda_0)}{\lambda_3^{(l)}} \qquad (2.53) \\ &= \frac{\eta_l^{(s)} (1 - \alpha_2^{(s)}) p_s}{\sum_{s=1}^{S} \eta_l^{(s)} (1 - \alpha_2^{(s)}) p_s}. \qquad (2.54) \end{aligned}$$

The distributed video system can be designed as follows (with homogeneous file size and holding time): We first obtain $C_2^{(s)}$ by $(\lambda_0 p_s T_h + n_2) C_f$. Using Eq. (2.52), we obtain $\lambda_3^{(l)}$. We then obtain $B_3^{(l)}$ satisfying the miss delay requirement.

Let $\sigma^{(l)}$ be the resultant utilization of $B_3^{(l)}$ for site l (given by $\lambda_3^{(l)} C_f / B_3^{(l)}$), and hence $\delta_l^{(s)} \sigma^{(l)}$ is the utilization of $B_3^{(l)}$ to deliver files to server s. Further denote $B_3^a = \sum_{l=1}^{L} B_3^{(l)}$ the maximum aggregate bandwidth used for staging in the system.

What is left is to specify $B_2^{(s)}$. We first find the bandwidth necessary in server s for staging, $B_*^{(s)}$. Obviously $B_*^{(s)} \leq B_3^a$. In fact, since concurrent video staging from all L permanent sites to a certain server is highly unlikely, $B_*^{(s)}$ can be much less than B_3^a. We design $B_2^{(s)}$ so that the total staging bandwidth exceeding such value almost never happens.

$B_*^{(s)}$ is obtained as follows: At any time, permanent site l is staging a file to server s with probability $\delta_l^{(s)}\sigma^{(l)}$. Consider the following generation function for server s,

$$g_s(y) \;=\; \prod_{l=1}^{L}\left((1-\delta_l^{(s)}\sigma^{(l)})+\delta_l^{(s)}\sigma^{(l)}y^{B_3^{(l)}}\right), \tag{2.55}$$

$$\equiv \; \sum_{i=0}^{2^L} c_i^{(s)} y^{B(i)}, \tag{2.56}$$

where $0 = B(0) < B(1) < \ldots < B(2^L) = B_3^a$. The coefficient $c_i^{(s)}$ is the probability that staging bandwidth in server s equals $B(i)$. We can therefore take $B_*^{(s)} = B(I)$, where $\sum_{i=I+1}^{2^L} c_i^{(s)} \ll 1$. $B_2^{(s)}$ is then obtained as $(p_s\lambda_0 T_h + n_2)b_0 + B_*^{(s)}$.

Much bandwidth is saved with bandwidth sharing. Let's consider our previous example again. We have $p_s = 0.1$ (each server serves 20 req./hr) and note that $\eta_l^{(s)} = 0.125$ for any server s. With $C_2^{(s)} = 45$ GB (which gives $\overline{D}_l^{Miss} = 2.0228$ minutes), we get $\lambda_3^{(l)} = 24.72$ req./hr. Given $\lambda_3^{(l)}$ and $\overline{D}_l^{Miss}$, $B_3^{(l)} = 15$ MB/s, and hence $\sigma^{(l)} = 0.4545$. The generator function for server s is therefore $g_s(y) = (1 - 0.4545/10 + 0.4545y^{15}/10)^8 = 0.689 + 0.2626y^{15} + 0.04376y^{30} + 0.004167y^{72} + \ldots$, from which we take $B_*^{(s)} = 15$ MB/s. Therefore, $B_2^{(s)} = 23.44$ MB/s. Note that both B_3 and B_2 are greatly reduced compared with the partitioned case.

2.6 Conclusions

Video-on-demand (VOD) encompasses services such as movie-on-demand, home-shopping, news-on-demand, various distributed/interactive training programs, etc. Video servers based on hierarchical storage systems offer high-capacity, low-cost and scalable video storage. The system consists of a secondary level (characterized by fast file access and high throughput) and a tertiary level (characterized by cheap and large storage). Video files stored in the tertiary level are staged into the secondary level to be displayed. The design of such system includes its architectural parameters (bandwidth and storage in each level), and operational procedures (e.g., request scheduling and data replacement policies) for different level of user interactivities.

There are two types of users in a hierarchical system, hits and misses. We find that hits enjoy negligible delay; therefore the tertiary bandwidth should be designed to meet user delay requirement. An efficient tertiary level is very important in a high-performance hierarchical storage system, and hence if a video file is displayed after it is completely staged, we should parallelize the drive bandwidth. On the other hand, if a video can be displayed while it is being staged we should use multiple independent drives.

We have developed a simple model for a hierarchical storage system, from which we can specify the required bandwidth and storage in both secondary and tertiary levels to meet a certain user delay requirement, given specific application characteristics and a target request rate. We find that user delay increases rapidly when the arrival rate increases beyond the target arrival rate. Therefore, admission control should be used so that a hierarchical server should not be operated beyond the arrival rate under which it is designed.

We have seen that the number of files stored in the secondary level can be much lower than the total number of files in the system, and hence a hierarchical storage system is able to achieve much lower system cost compared to a system with secondary storage only. With large number of files, the storage and bandwidth requirements do not depend sensitively on the skewness of video popularity. The simple model we developed can also be used to specify bandwidth and storage requirements in a distributed storage system, in which multiple geographically-distributed servers get their video files from some remote repositories and streams the videos to their local users.

Chapter 3

Request Batching and Multicasting in VOD

3.1 Introduction

In pure-VOD, each user is assigned its own dedicated unicast stream. Hence users enjoy great flexibility in interacting with the server while viewing their videos. However, pure-VOD does not scale up well with the user population and becomes very expensive when a large number of concurrent requests have to be accommodated.

When the video content is popular, the use of a single multicast stream serving many users simultaneously becomes more cost-effective. This is accomplished by grouping (i.e., batching) many requests for a given content arriving over a period of time and serving them with a single multicast stream. This is referred to as near-VOD [75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87]. However, the bandwidth saving as compared to pure-VOD is achieved at the expense of user start-up delay. Since each user no longer has a dedicated stream, such technique is effective when user interactivity does not have to be flexible or is not essential, as is the case with such applications as movie-on-demand. Providing interactivity in the multicast environment has been discussed in [88, 89, 90, 91].

Today, request batching is practically used in many satellite and cable network-based

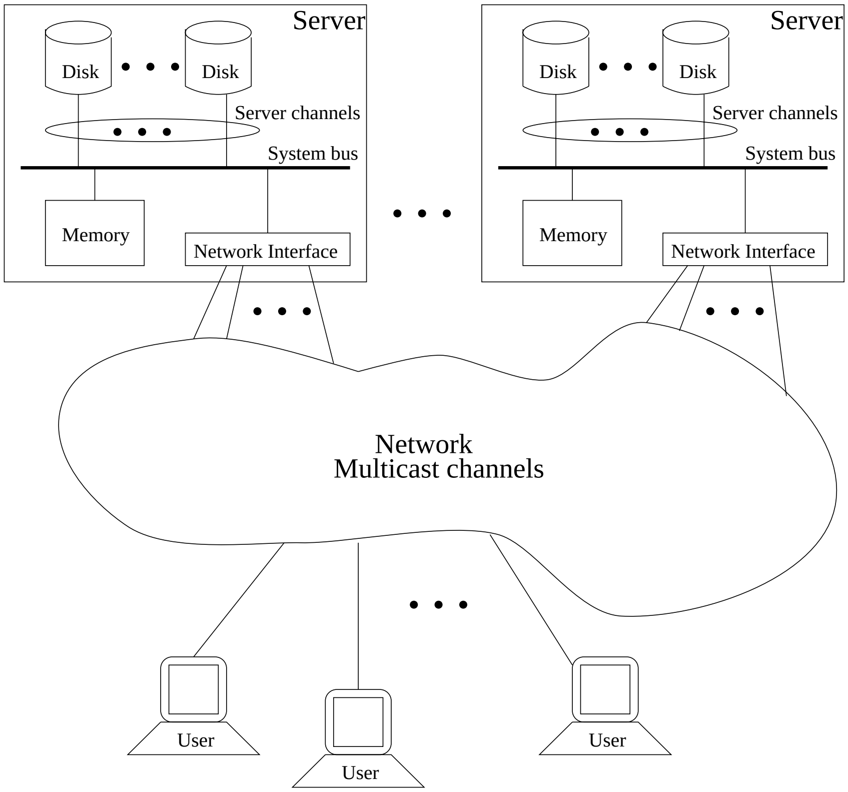


Figure 3.1: A near video-on-demand system

movie-on-demand services. It consists of having the same movie shown at specific pre-scheduled points in time, with the time between consecutive showings (referred to as the batching window) equal to some fraction of the movie's duration. Clearly, for a given request arrival process, the larger the batching window is, the smaller is the number of channels (and hence network bandwidth) used and the larger the batch size is (which, in turn, translates to higher revenue per channel), but the longer is the delay experienced by a user.

In Fig. 3.1, we show the three components constituting a near video-on-demand system:

- Video servers — The video servers store a number of movies (characterized by their duration, popularity and streaming data rate) accessible by the users. Each server has finite storage and streaming capacities [35, 34, 32, 33, 29, 30, 31, 26, 15, 37, 38, 50, 49]. Such resources are considered to be always available and in a sense already paid for. The available streaming capacity may be partitioned or shared among the movies. In a near VOD system, the main issue is to appropriately assign the limited streaming capacity to the various requests by means of batching.

- Network — The network is considered to offer the multicast channels needed. In a near VOD system, there may be a certain number of channels that are leased by the service provider and always available (already paid for), in which case the issue is the same as for the streaming capacity of the server. On the other hand, multicast channels may be requested by the near VOD service provider on-demand at some cost. This is the case when, for example, satellite channels are used or some tolls are to be paid in order to use a multicast stream. In this case, the issue from the service provider's point of view is to amortize the cost of the channels and guarantee a certain level of profit.

- Users — The users make requests to view certain movies. These requests are characterized by the stochastic process representing the arrival in time of a request, which depends on a number of factors (the time of the day, the occurrence of some news/events, etc.), and the choice of movies, which depends on the movie's popularity (the probability of selecting the movie). What is important for the users is the waiting time, the time from when the user places a request until the movie display is started. Depending on the waiting time incurred, a user may cancel its request and leave the system (i.e., renege). The reneging behavior of the users is an important consideration in the design of a near VOD system and the underlying request batching schemes. Many user reneging models have been considered in the literature, in which the time a user is willing to wait before reneging is distributed according to some cumulative distribution functions; namely, an exponential function [75, 76, 78, 82], a truncated Gaussian function [77] or a linear function (uniform distribution) [76]. However, in practice, there is no real data on the user's reneging behavior, and the use of any specific model has been either arbitrary, or driven by the need to keep the analysis tractable. In the absence of any information about user's reneging behavior, it may be appropriate to use a simple model in which users are willing to wait for a certain amount of time, beyond which they would not be satisfied and may be considered reneging with very high probability (i.e., reneging function is a step function with a delay limit). In some situations, the delay limit may be a design choice that the service provider makes and advertises for the service. A batching scheme in this case therefore should be designed so that user's delay is bounded by this time.

Previous work on near video-on-demand concentrated on the streaming capacity of the servers (which is given and already paid for), and therefore addressed the issue of how the available server channels are to be assigned to requests so as to achieve maximum throughput; more specifically, they considered a certain design goal pertaining to the loss of requests such as minimizing the loss rate, guaranteeing that loss is uniform across all movies, or some trade-off between the two [75, 76, 77, 78, 79, 92]. These studies would be applicable to the case where network channels are leased and therefore are available (in limited number) and paid for prior to service offering.

We consider a near VOD system in which the network channels are acquired on-demand. Associated with the use of each channel is a certain cost. Under this circumstance, request loss rate may no longer be the only measure of performance, and minimizing loss rate may no longer be the only design goal. From the service provider's point of view, profit is an important consideration; and maximizing profit while offering acceptable user delay (and by the same token acceptable request loss rate) becomes the important design goal. In this chapter, we primarily consider the case in which the number of on-demand channels that can be acquired is unlimited. Accordingly, the servicing of requests pertaining to a given movie is independent of the servicing of requests for other movies, and hence it is sufficient to consider the single movie case. We do also consider the case in which the number of available on-demand channels for a given movie is limited and study the effect of such limitation on the system design and performance. (However, we do not consider the case where there is a limited number of on-demand channels that are shared among multiple movies, but simply note that the performance of a system with sharing is always better than that in which the available bandwidth is partitioned among the movies.)

We first examine two well-known basic batching schemes, namely, the window-size based schemes and the batch-size based scheme. We first analyze these schemes under the condition that users do not renege and compare them in terms of the number of required concurrent channels (also referred to as streams), the number of users served per stream (which translates to revenue per stream), and the delay experienced by a user. In the window-size based batching schemes, a maximum user delay is guaranteed. This maximum delay is equal to the batching window size; conversely, if the user reneging behavior is a step function, then the window-size based schemes with the window size equal to the delay limit would lead to no user loss and thus maximum throughput. Obviously this also corresponds to maximum profit. With the batch-size based scheme, per stream revenue (and thus per stream profit) can be guaranteed. The performance characteristics of these schemes without user reneging is useful to design a system with a certain delay or profit objective. For example, providers may want to provide a service in which the probability of the user delay exceeding a certain value $\hat{d}$ is very low.

We then introduce a new adaptive scheme which combines the key advantage of the window-size based schemes (namely guaranteed delay) and the advantage of the batch-size based scheme (namely guaranteed per stream revenue) by ensuring that when the arrival rate is sufficiently high (and hence profit can be easily achieved), the system guarantees fairly low delay D_{min} to the users by batching them according to a window size equal to D_{min} (for service competitiveness); but when arrival rate is not so high, the system guarantees a certain level of profit as long as users' delay does not exceed a certain bound D_{max}. The scheme therefore balances service quality (in terms of the user delay experienced) and system profit adaptively.

We then examine the design of near VOD systems with user reneging, driven by the goal to maximize the profit rate (which also corresponds to the profit achieved over a long period of time). System profit depends on the length of the batching period: if the period is too short, many network channels would be used with a few users per channel leading to high usage cost; as the period increases, the batch size increases, but due to the reneging behavior of the users, the batch size reaches a limit; thus if the batching period gets too long, the profit rate decreases due to lost opportunities. We find the optimum values for the parameters of the batching schemes to maximize the profit rate.

The chapter is organized as follows. We first describe previous related work in Sect. 3.2. In Sect. 3.3, we first describe the two basic batching schemes, and analyze their performance in terms of the batch-size, number of concurrent streams, and user delay experienced. We then present the adaptive scheme; and finally, we address the minimum number of channels required to meet a certain user delay requirement.[1] In Sect. 3.4, we find the optimum values for the parameters of the batching schemes to maximize the profit rate with user reneging. We conclude in Sect. 3.5.

3.2 Previous Work

Gelman and Halfin consider that channels are partitioned for the movies (so that each movie is assigned a number of channels), and analyze user loss rate (because it reflects the throughput of the system, i.e., rate of the requests served per unit time) for a particular movie with the following three batching schemes [75]:

- An arrival finding an idle channel would be served immediately, otherwise it would join a queue waiting to be served. Users may renege while they are waiting in the queue. Whenever a channel is available, it is assigned to all the requests in the queue. The paper analyzes the multiple channels case in which users can wait indefinitely (i.e., no reneging) or cannot wait at all (i.e., an $M/D/n/n$ queue). With user's reneging function being exponential, it considers the single channel case, and shows that as users become more patient, the throughput increases.

- The movie is displayed once M or more requests are collected (the batch-size based

[1] Part of the work in this section has appeared in [84, 85].

scheme, with $M = 1$ as the special case of the above scheme). The paper analyzes the single channel case with exponential reneging function, and shows that there is an optimal M to achieve minimal loss rate.

- The movie is scheduled at regular intervals (fixed scheduling). Multiple channels and exponential user reneging function are considered. It shows that as users are more willing to wait, an increase in the number of channels does not bring much improvement in the loss rate.

Asit Dan et. al. consider multiple server channels in [76], with the channels shared among multiple movies (of fixed length) of non-uniform popularity (Zipf distribution with parameter 0.271). The performance measures considered are user loss rate, unfairness (indicated by the variance of the loss rates across the movies) and average user delay. Users in the system are willing to wait for at least a fixed amount of time, beyond which their waiting time is exponentially distributed (only in one case that they consider user's reneging time as uniformly distributed). Using simulation, they study the following batching schemes:

- FCFS: Incoming requests are time-stamped. Whenever a server channel is released, the oldest request in the system, along with all the requests of the same movie, is served.

- FCFS-n: The most popular n movies are schduled according to the fixed scheduling (all with the same window size). For these n movies, users are assumed never reneging. The remaining server channels are shared among the remaining movies according to FCFS batching scheme.

- MQL (Maximum queue length): Incoming requests join seperate queues according to the movies they request. Whenever a channel is released, the longest queue is served first.

It is shown that MQL achieves lower user loss rate than FCFS when server channels are very limited (by starving some of the requests for non-popular movies). However, when there are plenty of channels, MQL may have higher loss rate since it pays no regards of how long each user has been waiting in the system while FCFS serves the the one most likely to renege (the oldest request) first. The examples also show that MQL also has less fairness and lower mean average delay. Concerning FCFS-n, they find that there is an optimal n such that user loss rate is minimized. They also study the minimum number of channels required in order to achieve a certain loss rate (5%), for the schemes studied, and find that FCFS-n, if designed properly, potentially saves much bandwidth compared with FCFS or MQL schemes.

Aggarwal et. al. in [77] follow up on [76] and propose the maximum-factored-queue (MFQ) batching scheme, which tries to balance the fairness of the FCFS with the low loss rate of the MQL. In their scheme, the movie queue-length is weighted by a factor biased against its popularity, and an idle channel is assigned to the queue with the highest factored queue length. The weighting factor was estimated to be inversely proportional to the square-root of the movie popularity. The authors proposes two heuristics to assess video popularity, one based on counting the number of requests for a particular movie in a past fixed period of time, and another based on the number of arrivals since the last showing of the movie. They simulate the system assuming that the user reneging behavior is a truncated (on the left) Gaussian distribution, with multiple movies (of the same length) of non-uniform popularities (according to the Zipf distribution with parameter 1). It is shown that the scheme strikes a balance between the performance of FCFS and MQL, in terms of user loss rate, fairness, and average user delay.

In [78], Almeroth et. al. consider non-stationary arrival process and examine the adaptibility of a number of batching schemes in terms of the loss rates and average user delay. They simulate a system with multiple server channels, and multiple movies (with movie length uniformly distributed between 105 minutes and 135 minutes) of non-uniform popularity (according to the Zipf distribution with parameter 0.271). A user is willing to wait for a certain amount of time, beyond which its waiting time is a truncated exponential distribution. Incoming requests join a single queue, and whenever a channel is allocated, the request at the head of the queue, along with all the requests of the same movie, will be served. The following channel allocation policies are simulated:

- FCFS — as in [76]. It is found that when there is a temporary burst in the request arrival, the idle channels are exhausted too quickly (and hence channels are run out too fast). This leads to high user loss rate, especially when the arrival rate is high. The variance of the user delay is expected to be quite large in this scheme.

- Forced wait (Auto-gating) — The request at the head of the queue has to wait for at least a certain amount of time (window size W) before it is served ($W = 0$ is the FCFS case). Channel assignment is hence regulated. For a given arrival rate, it is shown that there is an optimal window size to achieve minimum loss rate.

- Rate control schemes — In this scheme, the time is slotted into regular intervals, and a certain maximum number of channels can be assigned within each interval. In this way, the channel assignment is paced (so that channels would not be allocated too quickly). Channel assignment is accelerated if there are still many channels unused towards the end of an interval. The authors study two ways of pacing the channel allocation and show that such schemes are able to dynamically adapt to load changes to keep the loss rate low.

As in [76], the paper also specifies the minimum number of channels in order to achieve a certain low loss rate (5%), for the cases of the FCFS, forced wait, and pure rate control schemes. It is shown that if the arrival rate and user reneging behaviour are known, the forced wait is able to achieve similar bandwidth requirement as the rate control schemes.

In [79], Abram-Profeta and Shin consider the minimization of user loss rate given that the user's reneging time is exponentially distributed. They study the following two batching schemes:

- Fixed scheduling — Movies are scheduled at regular intervals. They consider the optimization problem of how a certain fixed number of channels can be partitioned among a number of movies (of the same length) with non-uniform popularity (Zipf distribution with parameter 0.271). They solved the problem by integer programming. They have also considered other assignment policies such as assigning each movie a number of channels proportional to its popularity or square-root of its populariy, or in such a way to minimize the average window size, and compared them in terms of throughput, average window size, and the standard deviation of the window size. The paper also analyzes the throughput with non-stationary arrival process (with arrival rate being sinusoidal).

- Batch-size based scheme — Using simulation on a single movie with a certain number of multiple channels, they found that there is an optimal batch size to minimize user loss rate.

The paper compares the throughput of the fixed-scheduling and optimized batch-size based scheme for one movie title, and shows that when channels are not so limited and users are quite patient, the throughput of the fixed-scheduling and batch-size based scheme are similar.

Gelenbe and Shachnai consider channel assignment problem in which arrivals join seperate queues corresponding to the movies requested [92]. The objective is to maximize the throughput of the system given a fixed number of server channels. They compare two batching schemes. In the first scheme, the time is slotted at regular interval T_h/N, where T_h is the movie length and N is the total number of server channels in the system. Queues are examined in a round-robin fashion in each slot, and a maximum of one queue is served in each slot (If all the queues are empty in a slot, no channel would be assigned). In the second scheme, the N server channels is partitioned for each movie according to its popularity, with the slot time for movie i equal to T_h/N_i, where N_i is the number of channels assigned. By choosing N_i optimally, throughput can be maximized. The maximum waiting time of a user in the system is exponentially distributed.

The authors model the system based on G-network (i.e., with "positive" and "negative" customers corresponding to arrival and reneging), and show the close agreement between analysis and simulation. They find that the partitioned scheme has a higher throughput, but requires knowledge in video popularity.

Tsiolis and Vernon consider in [93] a batching scheme (group-guaranteed server capacity) which permanently assigns certain numbers of channels to groups of movies in the system, and the channels are used by these movies according to the FCFS batching scheme. The authors first estimate the number of server channels assigned to each group of movies so so as to achieve minimum average user delay under the condition of no user reneging, given the total server channels, the number of movies and their respective popularities. Then they study the performance of such a scheme with user reneging (through simulation), and compare it with other proposed schemes (FCFS, FCFS-n, MFQ). It is shown that there is no much difference between the scheme and FCFS.

While all the above work consider fixed number of channels, this work considers on-demand channels with a cost incurred each time a multicast channel is used. Under this circumstance, minimizing request loss rate may no longer make sense since it would mean serving each request with a dedicated stream, and hence leading to low profit or unprofitable services. A more important issue is how batching can be done so that channel cost is amortized. We analyze a number of schemes in this light, and derive many measures related to system profitability, such as batch size, number of concurrent streams, and user delay distribution. We also consider the optimization of the schemes in order to ensure high profit, while satisfying user delay requirement or a certain low user loss rate.

When channels can be acquired on-demand, some of the batching schemes mentioned above are no longer applicable (e.g., FCFS, MQL, and MFQ). While the window-based schemes and batch-size based scheme have been mentioned previously [75, 78, 79, 86], we present a comprehensive analysis on the schemes, deriving all the important profitability and delay measures.

There is some other previous work on more advanced batching techniques. Golubchik et. al. consider in [80] that video frames can be repeated or dropped, so that adjacent streams can catch up with each other and merge into a single multicast stream ("adaptive piggybacking"). They study three schemes of merging adjacent streams in terms of their bandwidth saving. Aggarwal et. al. in [81] formulate a stream merging scheme as a dynamic program and explore its performance and complexity issues. Sheu et. al. study buffering part of the videos in the client sites so that video data can be streamed directly from the clients (instead of from the servers); hence decreasing the bandwidth requirement of the server [82]. They study, in terms of the measures such as throughput and average user delay, two ways to do such "chaining" operation — in the standard chaining, new arrivals may be served immediately while in the extended chaining, new arrivals are forced to wait for a period of time (so as to batch more users) before they are served. Yu et.al. and Liao et. al. study using additional memory in the server to cache some of the streamed data, so that the requests closely following each other can retrieve their data directly from the memory instead of from the disks [88, 83]. Our work can be combined with theirs to achieve higher system profit and lower user loss rate.

3.3 Batching Schemes and Their Analysis Without User Reneging

In this section, we describe the two basic batching schemes considered in this chapter and analyze them assuming users do not renege. We derive the number of channels required, the batch-size, and the delay experienced by a user. We compare the two schemes with each other. We then introduce the adaptive scheme and present its performance. We also consider what should the minimum number of streams be for a video file so that the probability of a user's delay D exceeding a certain value $\hat{d}$ is less than ε, i.e., $P(D > \hat{d}) < \varepsilon$.

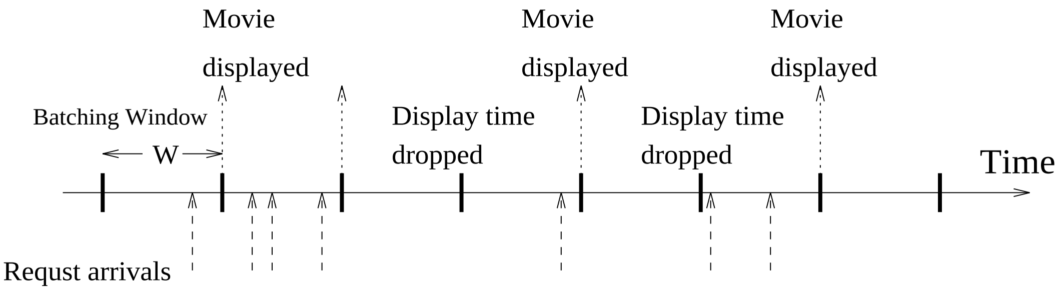


Figure 3.2: Fixed gating

3.3.1 Scheme description

A. Window-size based schemes

Consider a movie of duration T_h (minutes). The *fixed scheduling* scheme is the simplest of all window-size based batching schemes whereby a movie is shown once every exactly W minutes ($W \leq T_h$). The number of required concurrent streams is deterministic and is simply given by $\lceil T_h/W \rceil$. A user which makes a request to view the video between two such showings (the batching window) is served by the next showing following the request. If the average rate of arrival of requests is λ requests/minute, then the average number of users served by a stream is simply λW.

In the fixed scheduling, a stream is used even if no request has been made in the batching period preceeding it. As shown in Fig. 3.2, it is possible to omit a showing if no user has requested the movie in its corresponding batching window. This scheme is referred to here as *fixed gating*. The number of concurrent streams is no longer deterministic, and depends on the arrival process. In both fixed scheduling and fixed gating, the user delay is bounded by W, and the show time may be published ahead of time.

A third batching scheme which also guarantees a delay bound of W is shown in Fig. 3.3, in which the batching window is started with the first arrival in a batch and extends for a duration equal to W. We refer to this scheme as the *auto-gated scheduling*. While in this scheme, the show time cannot be announced ahead of time, each such show time is determined and may be advertised as soon as the start of the corresponding batching

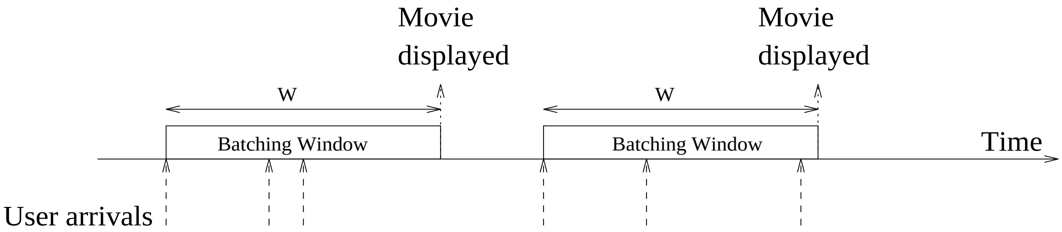


Figure 3.3: Auto-gated scheduling

window is determined.

B. Batch-size based scheme

In general, the concern of service providers is to have as many users in a batch as possible, since the system profitability is directly related to the average batch size. Let P_V (dollars) be the pay-per-view for a movie, and the total cost in using a channel be the sum of a fixed cost C (dollars) and a variable cost proportional to the number of users in the batch (with the proportionality constant α dollars/user). If $\overline{N}$ is the average number of users in a batch, the average per-batch profit is given by $\overline{N}P_V - (C + \alpha\overline{N})$. Define $P \triangleq P_V - \alpha$ (the provider's net revenue per user). A batching scheme is profitable if,

$$\overline{N} \geq \frac{C}{P} \triangleq K, \tag{3.1}$$

i.e., the average batch size has to be no less than a number K.

Note that in the auto-gating scheme, the number of users in a batch is high when the arrival rate is high. However, when the arrival rate is low, system profitability suffers. In the batch-size based scheme, we secure the profitability by batching $M \geq K$ users together before multicasting the movie, regardless of how long it takes to collect the M users. Note that with such a scheme, users suffer uncertain waiting time; i.e., the time at which the movie is displayed cannot be advertized to the users when they request service.

C. Moving-average scheme

This scheme addresses the objective of satisfying a certain average user delay requirement $\hat{D}$. Such a requirement may be met by dynamically adjusting the batching window so that the mean delay of the users in each batch is equal to $\hat{D}$. Given a set of known request arrival times $A_1, A_2, \ldots, A_i$ in a batch (counted from the time of the first arrival in the batch, and hence $A_1 = 0$), in order to keep the average delay in the batch equal to $\hat{D}$, the (future) tentative display time T_D has to satisfy,

$$\frac{1}{i}\sum_{j=1}^{i}(T_D - A_j) = \hat{D}, \tag{3.2}$$

which gives,

$$T_D = \hat{D} + \frac{\sum_{j=1}^{i} A_j}{i}, \tag{3.3}$$

i.e., the video display time is the average delay requirement $\hat{D}$ plus the average arrival time in the batch. Therefore the show time of a video has to be re-calculated at each arrival in a batch.

3.3.2 Analysis

Let again T_h denote the duration of a movie in minutes. We consider the request's arrival process to be Poisson with rate λ. In evaluating the various batching schemes, we are interested in the following closely-related performance measures:

- *The required number of concurrent streams, both its distribution $f_S(s)$ and its mean value $\overline{S}$:* Let $S(t)$ be the number of concurrent streams used at time t. We are interested in the average number of concurrent streams allocated for the video file, defined as $\overline{S} \triangleq \lim_{T\to\infty} \frac{1}{T}\int_0^T S(t)dt$. Since the average number of concurrent streams required in pure-VOD is λT_h, the bandwidth saving η is given by $\eta = 1 - \overline{S}/(\lambda T_h)$.

- *The batch size (the number of users served together in a batch) N, both its distribution $f_N(n)$ and its mean $\overline{N}$:* $\overline{N}$ is related to $\overline{S}$ by $\lambda T_h = \overline{S}\ \overline{N}$. Indeed, by the Little's formula, the average number of user requests which arrive during the video display time T_h is simply λT_h, which is also equal to $\overline{S}\ \overline{N}$.

- *The delay D of a user, both its distribution $f_D(t)$ and its mean $\overline{D}$.*

A. Window-size based schemes

Fixed scheduling: The distribution of the batch size is Poisson with mean $\overline{N} = \lambda W$; we also have $\overline{S} = T_h/W$ and $\eta = 1 - 1/(\lambda W)$. D is uniformly distributed between 0 and W, and $\overline{D} = W/2$.

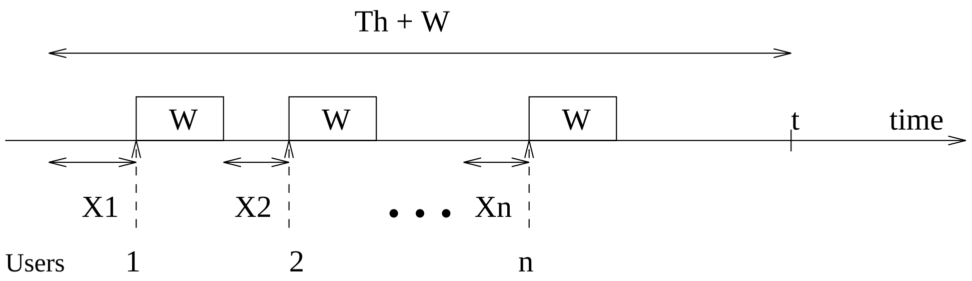


Figure 3.4: Time diagram for analyzing stream distribution at time t

Fixed gating: The probability that there are i $(i \geq 1)$ requests in a batch is given by $(\lambda W)^i e^{-\lambda W}/(i!(1-e^{-\lambda W}))$; hence, $\overline{N} = \lambda W/(1-e^{-\lambda W})$,

$$\overline{S} = \frac{T_h}{W}(1-e^{-\lambda W}), \tag{3.4}$$

and $\eta = 1-(1-e^{-\lambda W})/(\lambda W)$. As expected, when $\lambda \to 0$, $\overline{S} \to \lambda T_h$ (the number corresponding to pure-VOD); and $\overline{S} \leq T_h/W$ with $\lim_{\lambda\to\infty} \overline{S} = T_h/W$.

Given that there is an arrival within a batching window W, its arrival time is uniformly distributed within the window; hence, $f_D(t) = 1/W$, for $0 \leq t \leq W$, and $\overline{D} = W/2$.

Auto-gated scheduling: The distribution of the batch size is $P(N = i) = (\lambda W)^{i-1}$ $e^{-\lambda W}/(i-1)!$, for $i \geq 1$. Hence $\overline{N} = 1 + \lambda W$ (which is exactly one more than that of fixed scheduling), and

$$\overline{S} = \frac{\lambda T_h}{1+\lambda W}, \tag{3.5}$$

and $\eta = \lambda W/(1+\lambda W)$. The normalized profit rate is given by $\hat{\theta} = \lambda(1-K/(1+\lambda W))$.

In terms of the delay distribution, since the first user in each batch has a delay of W while the remaining ones in the batch have a delay uniformly distributed between 0 and W,

$$f_D(t) = \tfrac{1}{1+\lambda W}\delta(t-W) + \tfrac{\lambda}{1+\lambda W} \quad \text{for } 0 \leq t \leq W, \tag{3.6}$$

where $\delta(t)$ is the usual impulse function with $\delta(t) = 0$ for $t \neq 0$ and $\int_{-\infty}^{\infty}\delta(t)dt = 1$. $\overline{D}$ is hence given by,

$$\overline{D} = \frac{\lambda W + 2}{2(1+\lambda W)}W. \tag{3.7}$$

We derive the (approximate) expression of the stream distribution for the auto-gated scheduling. Refer to Fig. 3.4. Consider an arbitrary time t, and denote S the (random) number of concurrent streams at that time (we assume that S is stationary and hence drop the subscript t). The event of $S = s$ ($s \leq \hat{S} \triangleq \lceil T_h/W \rceil$) means that exactly s movies have started their showings in $[t - T_h, t]$. Consider the period $[t - T_h - W, t]$, and assume that at time $t - T_h - W$, no batching window has been started (and hence the first arrival after time $t - T_h - W$ would start a batching window. Note that the assumption is good if λW is not high[2]). Denote X_i the interarrival time from the last showing of a movie to the arrival of the next request in time $[t - T_h - W, t]$. Obviously, X_i is exponentially distributed with rate λ. Note that the condition for $S \geq s$ is $(X_1 + W) + (X_2 + W) + \ldots + (X_s + W) \leq T_h + W$, i.e., $\sum_{i=1}^{s} X_i \leq T_h - (s-1)W$. Since $W = \sum_{i=1}^{s} X_i$ is the sum of s exponentials, its distribution is,

$$\sum_{i=1}^{s} X_i \sim g_W(x; s) = \frac{\lambda(\lambda x)^{s-1}}{(s-1)!} e^{-\lambda x}, \tag{3.8}$$

and hence, $P(S \geq s) = \int_0^{T_h - (s-1)W} g_W(x; s) dx$. Therefore,

$$\begin{aligned} P(S = s) &= P(S \geq s) - P(S \geq s+1) \\ &= \begin{cases} \int_0^{T_h - (s-1)W} g_W(x; s) dx - \int_0^{T_h - sW} g_W(x; s+1) dx, & \text{for } s < \hat{S}, \\ \int_0^{T_h - (\hat{S}-1)W} g_W(x; \hat{S}) dx, & \text{for } s = \hat{S}. \end{cases} \end{aligned} \tag{3.9}$$

B. Batch-size based scheme

The movie is displayed each time a new batch of M requests is collected. Therefore, N is deterministic and equal to M, $\overline{S} = \lambda T_h / M$, and $\eta = 1 - 1/M$. Clearly, for a given λ, the larger M is, the smaller $\overline{S}$ is; and in contrast with what we have seen for the window-based schemes, $\overline{S}$ keeps increasing as λ increases. Therefore, if M is fixed, when λ gets large, the batch-size based scheme in fact consumes more bandwidth than the window-based schemes. Let W denote the batching period; it is a random variable equal to the sum of $(M - 1)$

[2] Note that if λW is high, then the stream distribution would be so skewed towards the maximum stream T_h/W that $\overline{S} \approx T_h/W$ and the stream distribution would not be as interesting.

exponential variables, and its distribution $g_W(w)$ is given by,

$$g_W(w) = \frac{\lambda(\lambda w)^{M-2}}{(M-2)!} e^{-\lambda w}. \tag{3.10}$$

The user delay distribution may be obtained by conditioning on W. Given $W = w$, the first user in the batch has delay w, the last user has delay equal to 0, and the remaining $M - 2$ users have a delay uniformly distributed between 0 and w, i.e.,

$$f_D(x|w) = \tfrac{1}{M}\delta(x-w) + \tfrac{M-2}{M}\tfrac{1}{w} + \tfrac{1}{M}\delta(x), \quad \text{for } 0 \leq x \leq w. \tag{3.11}$$

By removing the condition on w, the user delay distribution is given by $(x \geq 0)$,

$$f_D(x) = \tfrac{1}{M} g_W(x) + \tfrac{M-2}{M} \int_x^\infty \tfrac{g_W(w)}{w} dw + \tfrac{\delta(x)}{M}, \tag{3.12}$$

and $\overline{D} = (M-1)/(2\lambda)$, which is equal to half of the average batching period.

C. Moving-average scheme

The analysis for this scheme has proven to be rather difficult. Accordingly, we have used simulation to study its performance.

3.3.3 Numerical results and comparisons

Window-based schemes

The auto-gated scheduling is chosen as a representative of window-based schemes. We consider $T_h = 90$ minutes.

In Fig. 3.5, we plot $\overline{S}$ versus W, given λ. Also shown in the figure is the maximum T_h/W which is attained when $\lambda \to \infty$, and corresponds to the number required in the fixed scheduling case. In Fig. 3.6, we plot $\overline{S}$ versus λ, given W. Given λ, when $W = 0$, $\overline{S} = \lambda T_h$ corresponds to pure-VOD. As W increases, $\overline{S}$ decreases from λT_h, first rapidly and then slowly; most of the bandwidth reduction as compared to pure-VOD is attained for a relatively low value of W, in the range of 6–8 minutes, and beyond this range the rate of reduction becomes relatively small. Furthermore, this gain is significantly more important

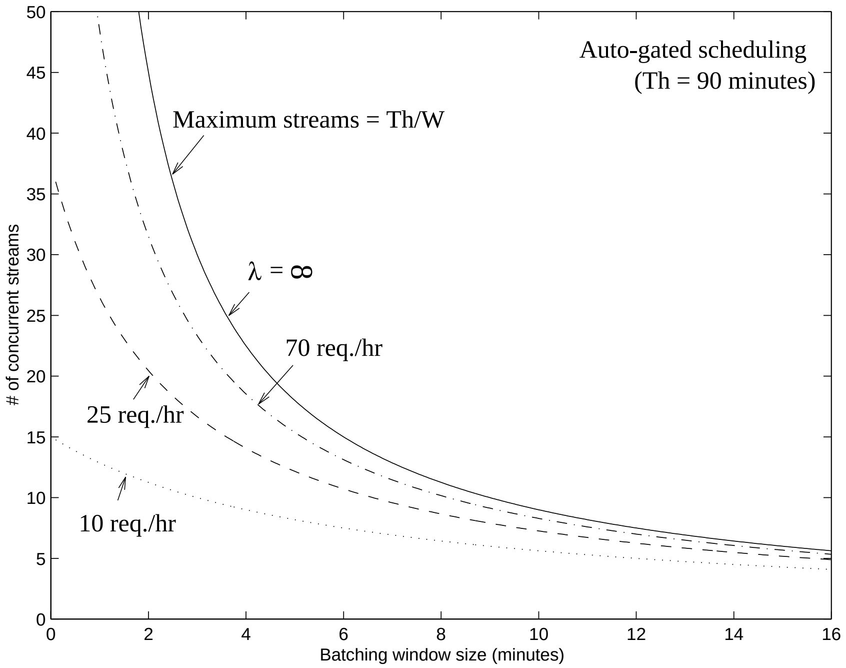


Figure 3.5: $\overline{S}$ versus W for the auto-gated scheduling

for larger λ than for smaller λ. We also note from both figures that the bandwidth reduction as compared to the fixed schedule (T_h/W) is more significant for smaller W and smaller λ. Indeed, the average number of streams which can be expressed as $T_h/(W + 1/\lambda)$ is small compared to T_h/W when $(W + 1/\lambda)$ is large as compared to W; this is the case when W is small given λ, and it is more pronounced when $1/\lambda$ is large.

The average batch size $\overline{N} = 1 + \lambda W$ is linear in both W and λ. In Fig. 3.7, we plot $\overline{N}$ versus W given λ. In order to achieve profitability, we need $\overline{N} \geq K$, i.e., $\lambda W \geq K - 1$; thus given λ, profitability is achieved as long as W is greater than $(K-1)/\lambda$. For example, when $K = 5$, for $\lambda = 120$ req./hr, W can be as low as 2 minutes to maintain profitability; however, if $\lambda = 25$ req./hr, W has to be increased to 12 minutes.

In Fig. 3.8, we plot the delay distribution for the auto-gated scheme for various combinations of (λ, W). By comparing Fig. 3.8(a) and 3.8(b), we note that as λ increases, there results a decrease in the impulse at W and an increase in the probability density from 0 to W, indicating that the delay distribution gets closer to the uniform distribution and the average delay decreases. Figure 3.8(c) as compared to Fig. 3.8(a) shows how the delay

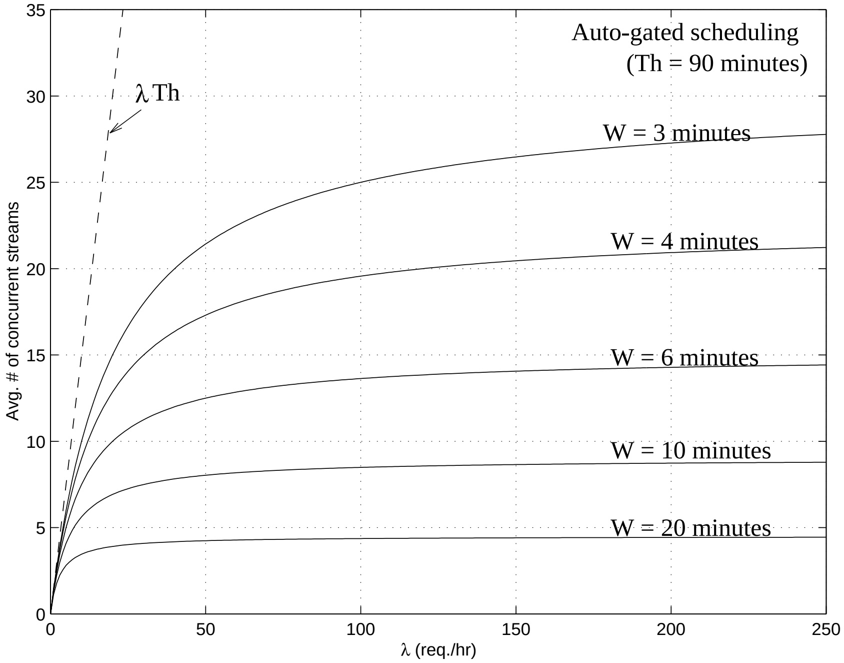


Figure 3.6: $\overline{S}$ versus λ for the auto-gated scheduling

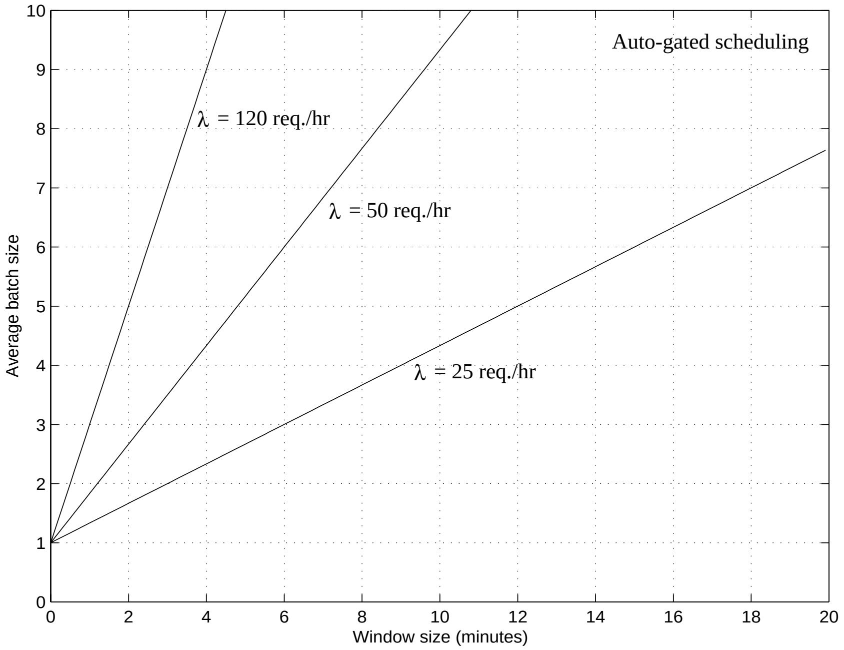


Figure 3.7: $\overline{N}$ versus W given λ

distribution changes as W increases; both the impulse and the density level decrease, and there is an increase in the average delay.

We now examine $\overline{D}$ and plot in Fig. 3.9 $\overline{D}$ versus W for given λ. From Eq. (3.7), $\overline{D}$ can be re-written as

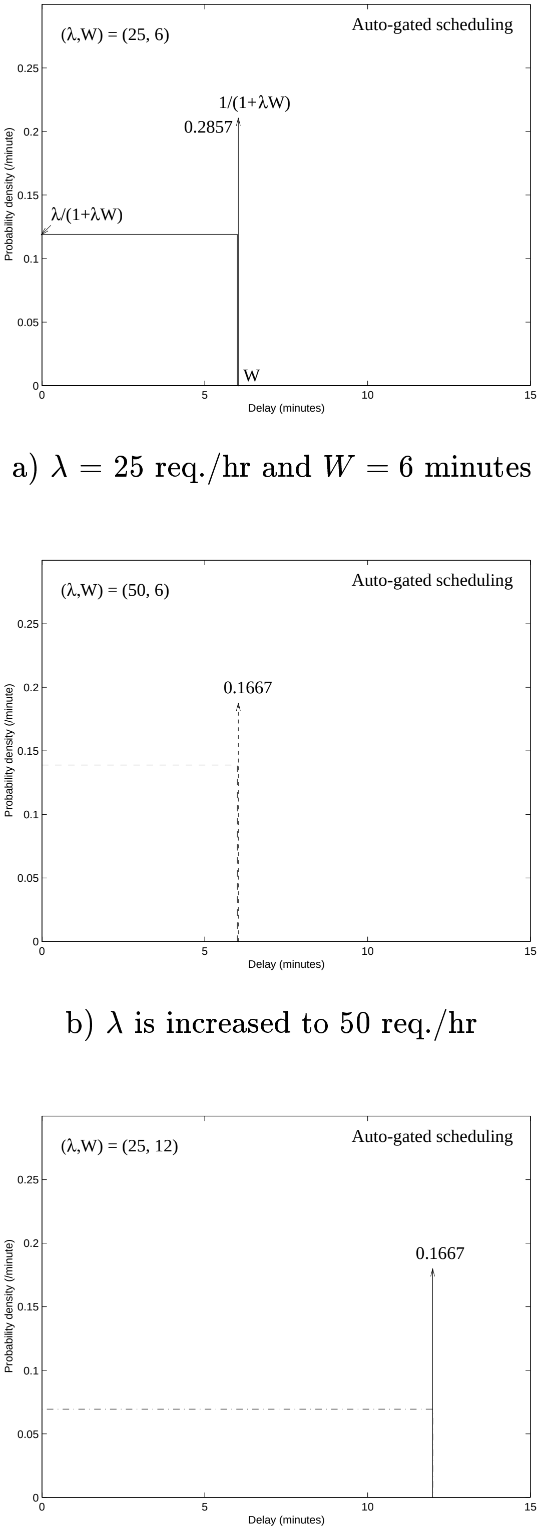
$$\overline{D} = \frac{W}{2} + \frac{1}{2(1/W + \lambda)} \leq \frac{W}{2} + \frac{1}{2\lambda}. \tag{3.13}$$

The upper bound is reached when W gets large so as $1/W$ gets small as compared to λ; for large λ, this is the case for even small value of W (e.g., for $\lambda = 120$ req./hr, $W \geq 5$ minutes).

In Fig. 3.10, we plot $\overline{D}$ versus λ, given W. The user delay at $\lambda = 0$ is equal to W with probability 1 (each batch would only have one user). As λ increases, $\overline{D}$ decreases from W, reaching asymptotically $W/2$. We observe that the drop in $\overline{D}$ as λ increases is sharper with larger values of W than with smaller value of W. This is expected since more users can be batched with a larger window. It is encouraging to note that for window size of only 12 minutes, the users experience an average delay of 8 minutes for an arrival rate as low as 10 req./hr., and an average delay close to 6 minutes for an arrival rate equals to 40 req./hr.

We consider now that W is selected appropriately for a given λ so as to guarantee a given average batch size M, and thus a given level of profitability (where M is selected to be larger than K). (Clearly, this assumes that the rate λ of the arrival process is known *a priori* and remains constant over relatively long periods of time.) In Fig. 3.11, we plot the minimum window size W necessary to guarantee $1 + \lambda W = K$, and the resulting average delay incurred, as a function of λ. Clearly, for a given K, larger values of λ require smaller values of W, and the higher K is, the larger W must be. For $\lambda = 40$ req./hr and $K = 10$, W has to be 14 minutes in order to maintain profitability; and if K is now 2 as opposed to 10, W may be decreased to 1.5 minutes.

Consider now that there is a certain level of profitability $M \geq K$ that needs to be guaranteed, and that the maximum delay requirement needs not be lower than a certain value $D^{(0)}_{max}$. Then for $\lambda > \lambda_0$ where λ_0 is such that $K = 1 + \lambda_0 D^{(0)}_{max}$, one may use the value $W = D^{(0)}_{max}$ and for $\lambda < \lambda_0$, W is selected so as to guarantee the batch size $M = K$. We



a) $\lambda = 25$ req./hr and $W = 6$ minutes

b) λ is increased to 50 req./hr

c) W is increased to 12 minutes

Figure 3.8: Delay distribution for the auto-gated scheme

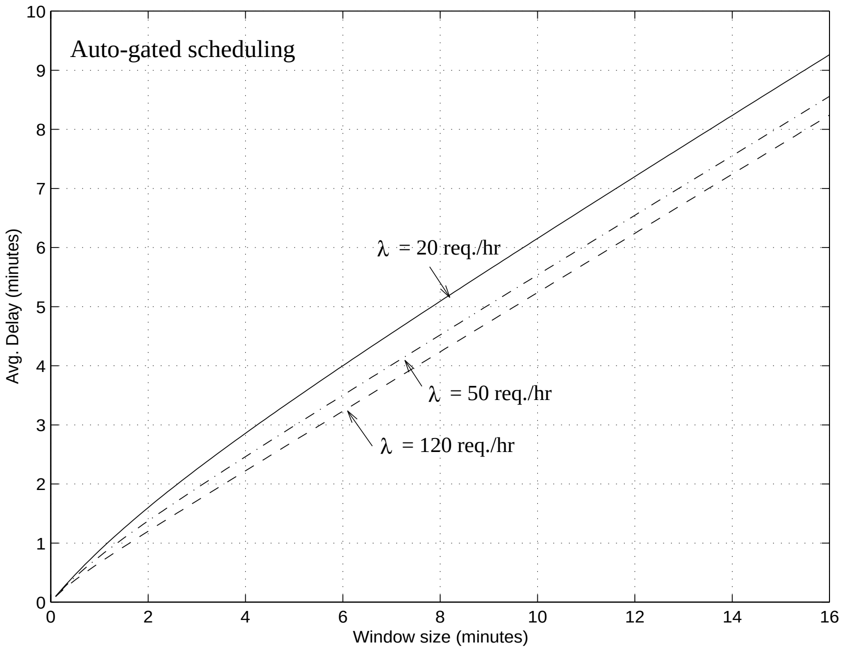


Figure 3.9: Average delay as a function of W given λ

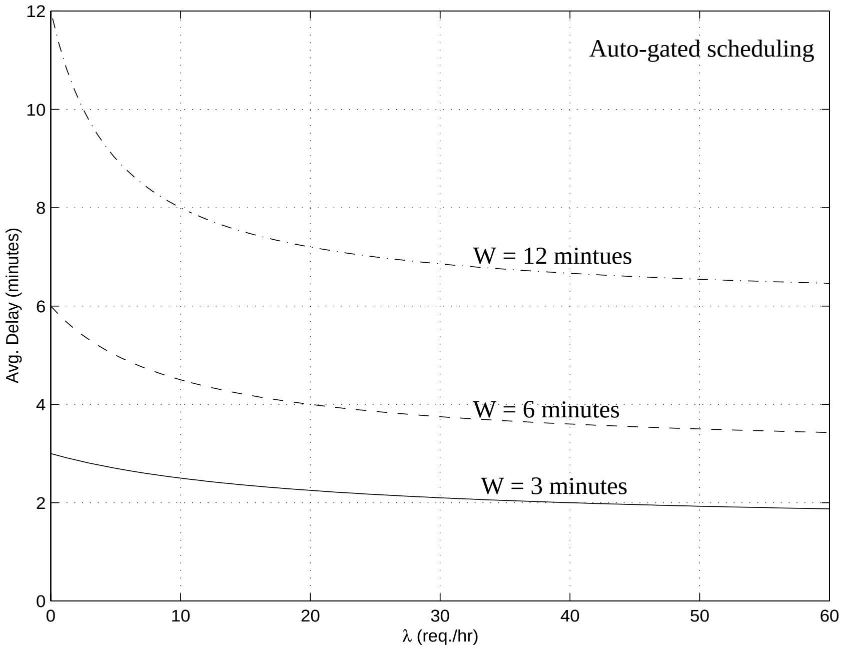


Figure 3.10: $\overline{D}$ as a function of λ for the auto-gated scheme

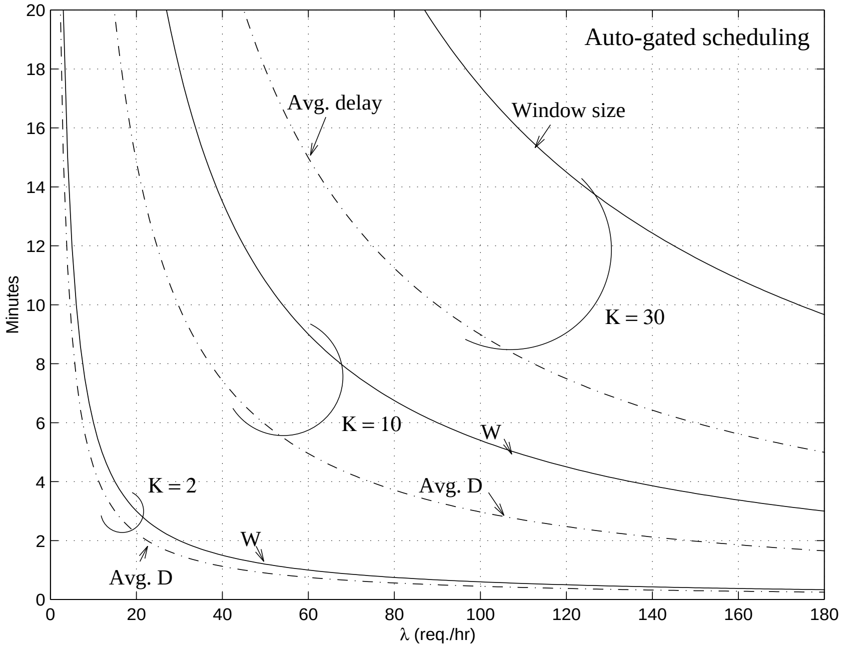


Figure 3.11: W and the corresponding $\overline{D}$ versus λ given K

show in Fig. 3.12 the case of $D_{max}^{(0)} = 12$ minutes and $K = 10$ (and hence $\lambda_0 = 45$ req./hr). We see from the figure that as λ increases but remains less than 45 req./hr, W is decreased just to maintain profitability. When λ increases beyond 45 req./hr, W does not have to be decreased further. From there on, the window size is kept at 12 minutes, thereby achieving higher profit. The result of such a choice is that the user delay when $\lambda < \lambda_0$ is larger than $D_{max}^{(0)}$.

One may achieve both profitability and a maximum delay guarantee if one were to charge users an additional fee over the delivery cost per movie ($P = P_V - \alpha$) chosen appropriately as a function of λ. P should be such that $K \equiv C/P = 1 + \lambda W$, and thus the fee normalized to C is given by $P/C = 1/(1 + \lambda W)$. We plot in Fig. 3.13 P/C versus λ given W. For very low arrival rates, when there is most likely only 1 user in a batch, the fee to be charged is equal to the stream cost. As λ increases, the fee to be charged drops quickly; for example, for $W = 6$ minutes when $\lambda = 40$ req./hr, the charge needs to be $0.2C$.

We now compare pure-VOD, fixed scheduling, fixed gating and auto-gated scheduling in terms of $\overline{S}$ and $\overline{N}$ for given values of λ and W. Since in all cases $\overline{S}$ is expressed in

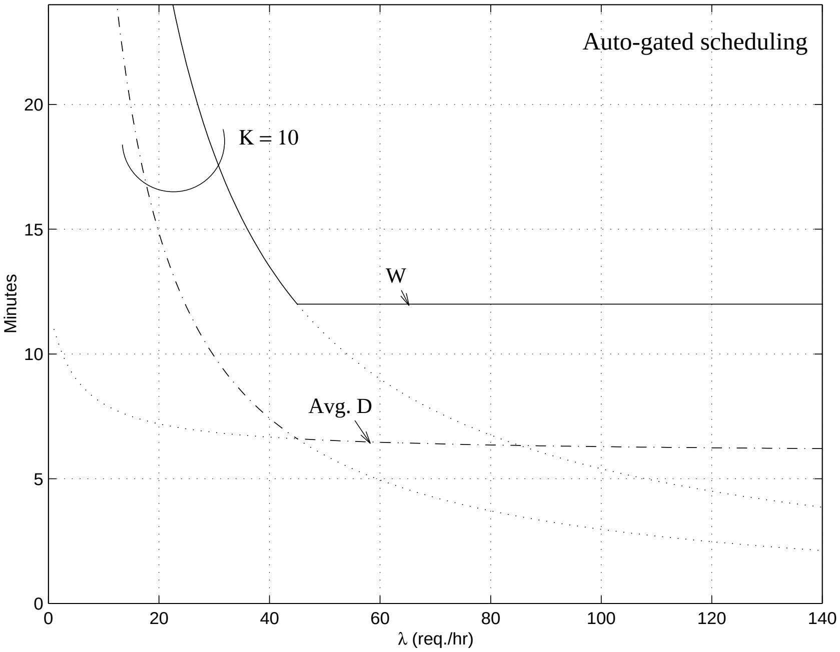


Figure 3.12: W and the corresponding $\overline{D}$ versus λ

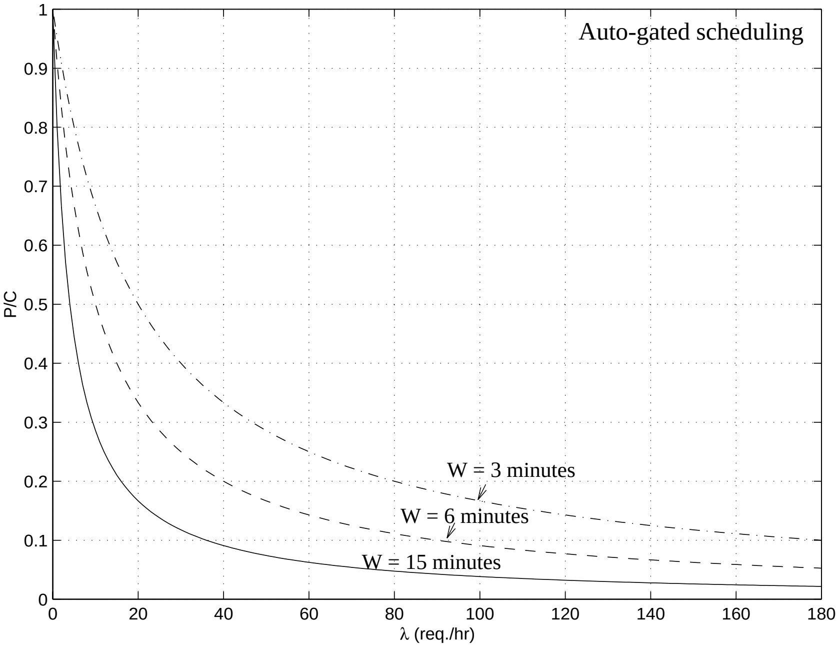


Figure 3.13: P/C versus λ

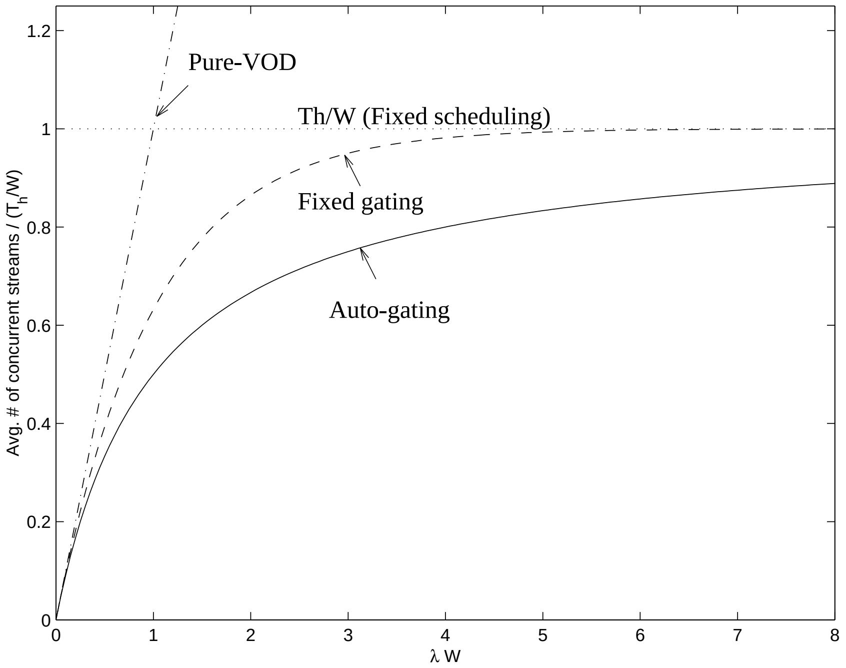


Figure 3.14: $\overline{S}$ (normalized to T_h/W) versus λW

terms of the product λW, we plot in Fig. 3.14 $\overline{S}$ normalized to T_h/W as a function of λW. Obviously, the auto-gated scheme consumes less bandwidth than fixed gating. In fact, given the expressions derived for $\overline{S}$ (Eqs. (3.4) and (3.5)), it is easy to show that fixed gating consumes up to 30% more streams than auto-gating (attained at $\lambda W = 1.7934$), while the maximum difference in $\overline{S}$ between the two is $0.2036 T_h/W$ (attained at $\lambda W = 2.51276$). Fixed scheduling always consumes on average T_h/W streams no matter what the arrival rate is, and both the fixed gating and the auto-gated schemes consume less bandwidth than fixed scheduling. However, as λW gets large, fixed scheduling is as good as fixed gating, and ultimately, the auto-gated scheme will asymptotically reach the same value. Note also that for $\lambda \leq 1/W$, even pure-VOD performs better than fixed scheduling. For completeness, we plot in Fig. 3.15 $\overline{S}$ as a function of W given λ.

We plot in Fig. 3.16 $\overline{N}$ for the various window-based schemes as a function of λ. We note that the batch sizes for all window-based schemes differ by at most one.

With respect to delay, for the same W, all window-based schemes guarantee the same maximum delay W. The average delay however is fixed at $W/2$ for both fixed scheduling

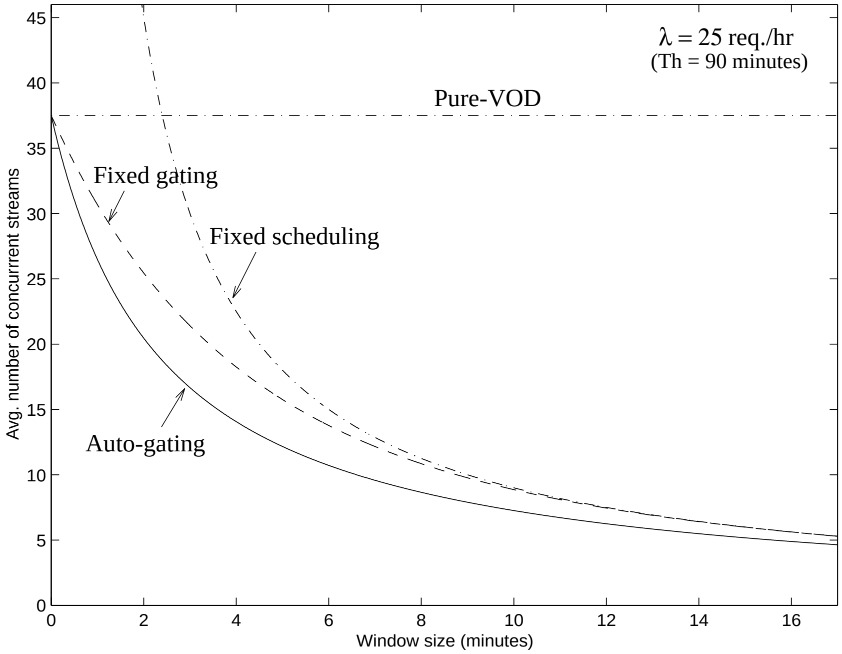


Figure 3.15: $\overline{S}$ versus W for the window-based schemes

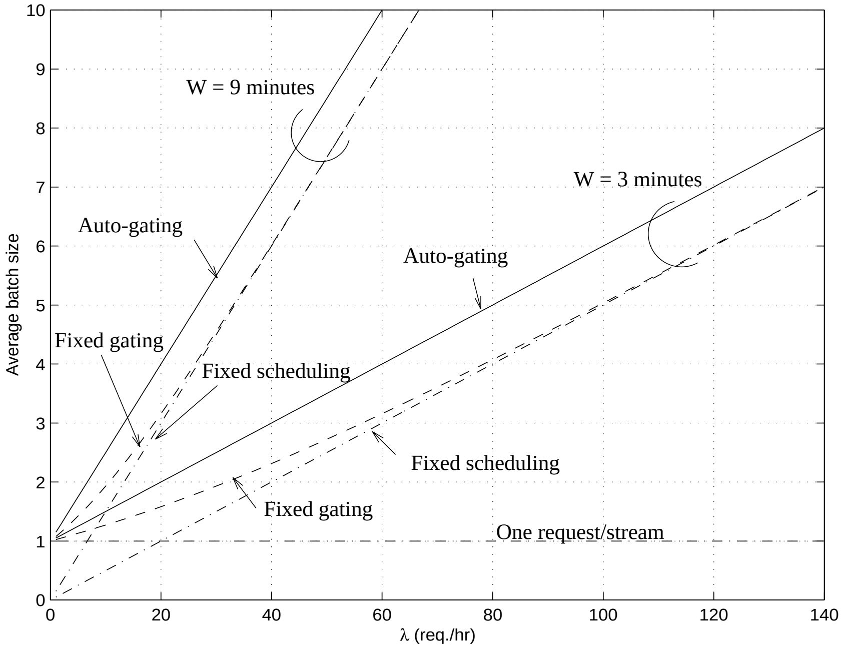


Figure 3.16: $\overline{N}$ versus λ given W

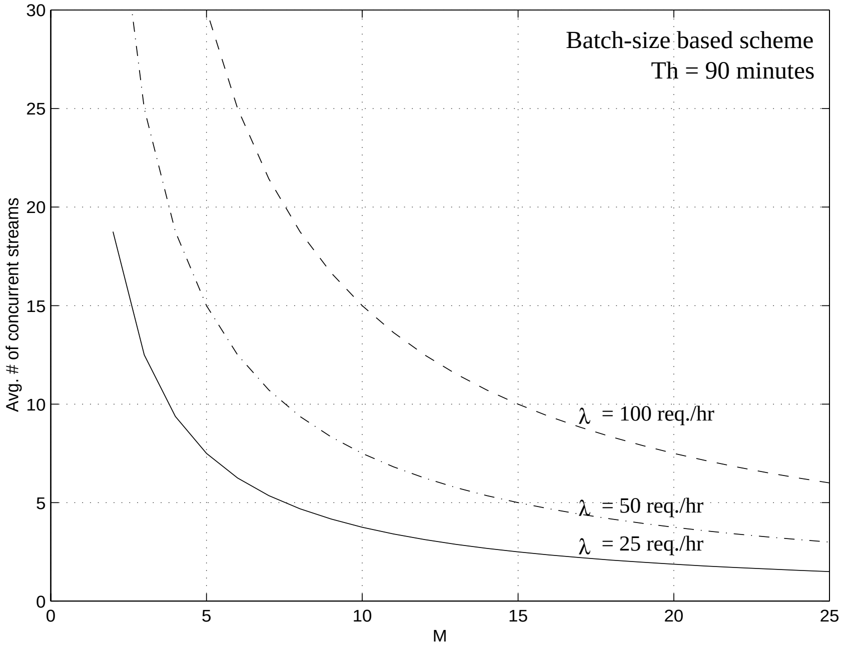


Figure 3.17: $\overline{S}$ as a function of M given λ

and fixed gating, while that of the auto-gated scheme is between $W/2$ and W depending on the intensity of λ (see Fig. 3.10).

Batch-size based scheme

We plot in Fig. 3.17 $\overline{S} = \lambda T_h/M$ versus M for various values of λ. Clearly, for a given λ, the larger M is, the smaller $\overline{S}$ is; $\overline{S}$ drops very rapidly as M is increased from zero, but then flattens out as it reaches asymptotically the value zero. $\overline{S}$ versus λ given M is plotted in Fig. 3.18. Clearly, $\overline{S}$ is linear in λ, and in contrast with what we have seen for window-based schemes in Sect. 3.3.3, $\overline{S}$ keeps increasing as λ increases, and does not settle to a limiting value. Therefore, if M is to remain fixed, when λ gets large, the batch-size based scheme in fact consumes more bandwidth than window-based schemes.

Since each batching period is extended until exactly M users have been collected, the delay in the batch-size based scheme is not bounded from above as is the case with the window-size based schemes. The distribution of delay is shown in Fig. 3.19 for given combinations of λ and M. There is an impulse at the origin of magnitude $1/M$ since

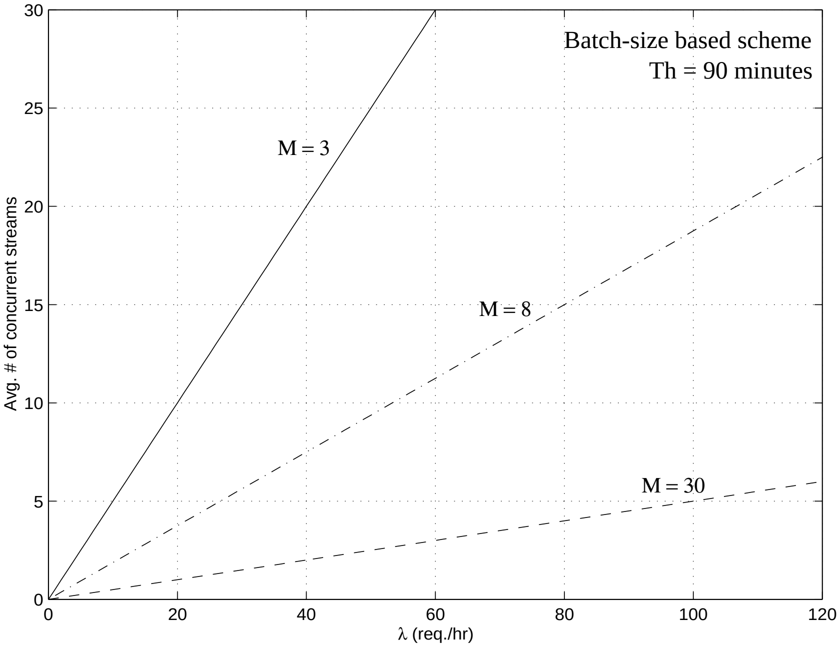


Figure 3.18: $\overline{S}$ versus λ given M

one user out of the M in a batch (the last one in a batch) enjoys zero delay. Figure 3.19(a) corresponds to $(\lambda, M) =$ (25 req./hr, 5). We see that the distribution has a tail indicating that some users have long delay. Figure 3.19(b) shows the distribution when λ is increased (to 50 req./hr). The impulse at the origin does not change, but the distribution is "squeezed" towards the origin, indicating that the user delay decreases when λ increases. Comparison between Figs 3.19(c) and 3.19(a) shows how an increase in M causes the impulse in the origin to shrink and the tail to spread out. The average user delay is hence increased.

From Sect. 3.3.2, the average delay is $\overline{D} = (M-1)/(2\lambda)$. We plot $\overline{D}$ versus M for given λ in Fig. 3.20. $\overline{D}$ increases linearly with M for a given λ; given λ the choice of a certain value M (greater than K) to achieve profitability comes at the expense of higher user delay. In Fig. 3.21, we show $\overline{D}$ versus λ, given M. $\overline{D}$ decreases with increasing values of λ. Note the sharp knee behavior of the curves in Fig. 3.21 which indicates that in the low range of M, a significant decrease in the average delay is achieved for a relatively small increase in λ, and the decrease in $\overline{D}$ beyond the knee is rather insignificant.

The maximum delay for auto-gated scheduling is W, while the distribution of delay for the batch-size based scheme has infinite tail. To compare them, we design both systems so as to satisfy a certain average delay $\hat{D}$ (in the auto-gated scheme, we select W while in the batch-size scheme, we select M so as to achieve $\hat{D}$), and then find the probability that the user delay D in the batch-size based scheme is higher than W. Given an average delay requirement $\hat{D}$, W is given by (from Eq. (3.7)),

$$W = \frac{1}{\lambda}\left(\hat{D}\lambda - 1 + \sqrt{1 + \lambda^2 \hat{D}^2}\right). \tag{3.14}$$

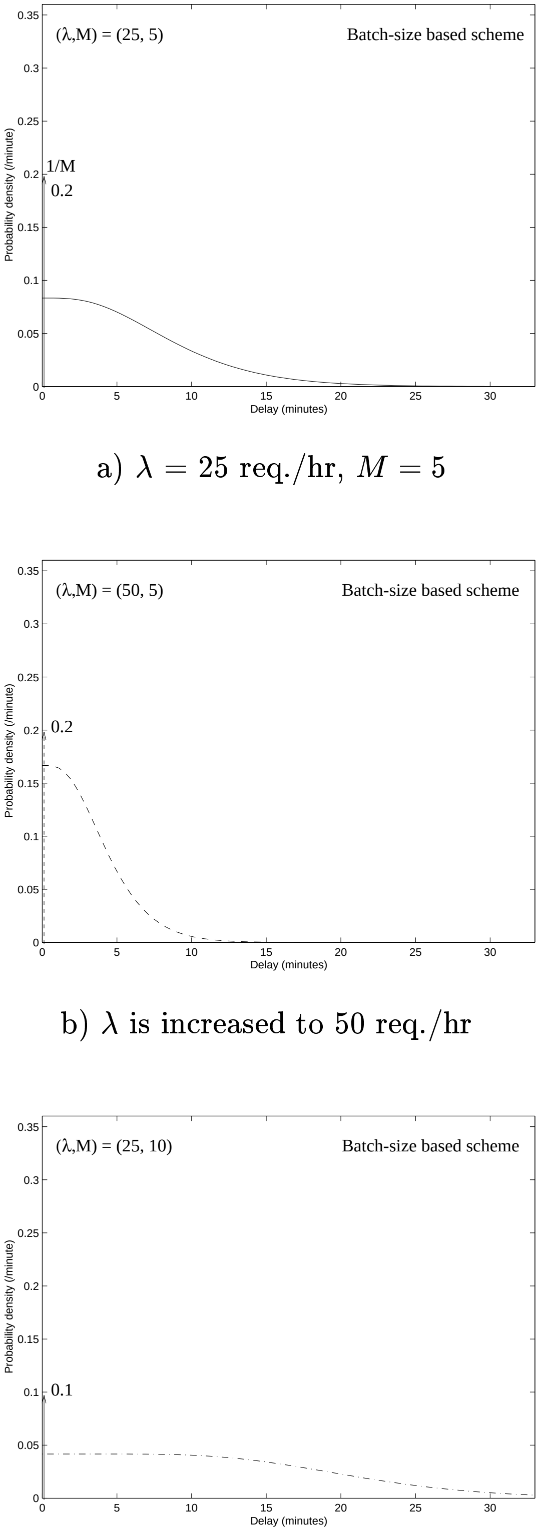
Note that as λ increases from zero, W increases from $\hat{D}$ to $2\hat{D}$. For the batch-size based scheme, in order to achieve the required average delay $\hat{D}$, we need

$$M = 1 + 2\hat{D}\lambda. \tag{3.15}$$

In Fig. 3.22, we show the probability that the user delay in the batch-size based scheme exceeds W as a function of λ. We see that when the arrival rate is low, this probability is high. However, as the arrival rate increases, this probability decreases.

We now consider the design of both systems so as to satisfy a given profitability K and compare their average delay. We plot in Fig. 3.23 the average delay as a function of λ for given K. We see that both schemes have average delays that are fairly close (with the batch-size based scheme having slightly lower delay).

Finally, it is interesting to note that for the same average batch-size, the average batching period for the batch-size based scheme and auto-gated scheduling are the same. How-



a) $\lambda = 25$ req./hr, $M = 5$

b) λ is increased to 50 req./hr

c) M is increased to 10

Figure 3.19: Delay distribution for the batch-size based scheme

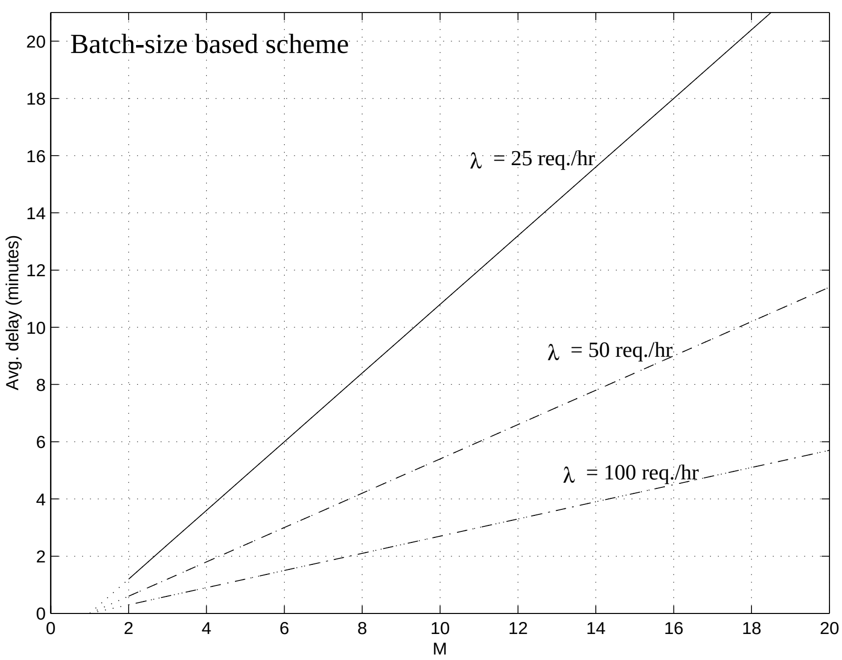


Figure 3.20: $\overline{D}$ as a function of M given λ

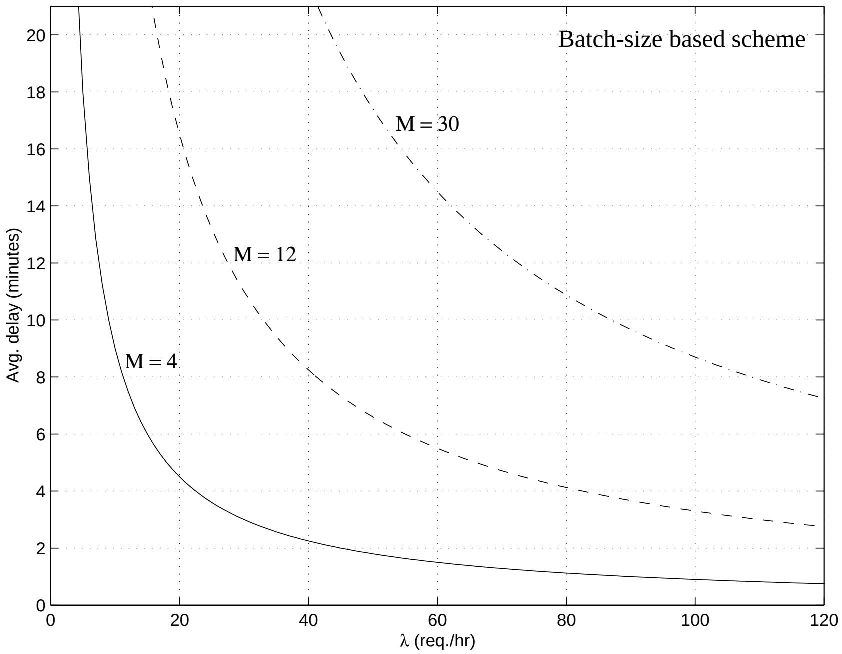


Figure 3.21: $\overline{D}$ versus λ given M

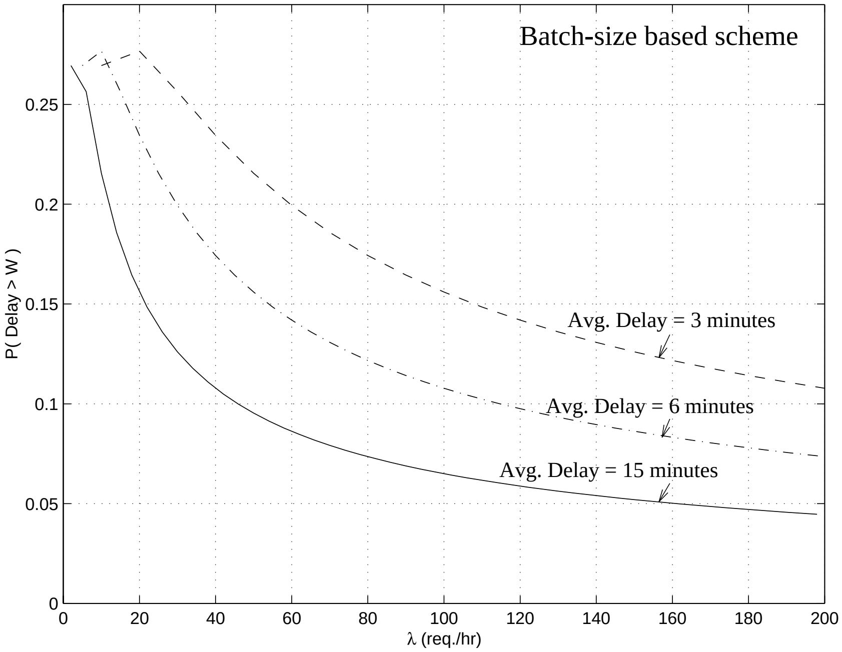


Figure 3.22: $P(D > W)$ for batch-size based scheme, given $\hat{D}$

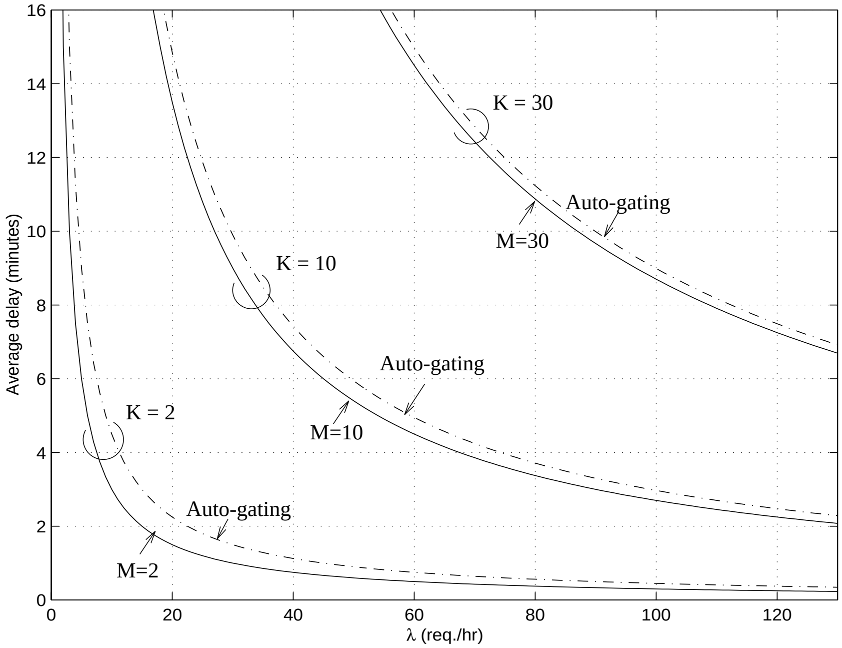


Figure 3.23: $\overline{D}$ versus λ given K

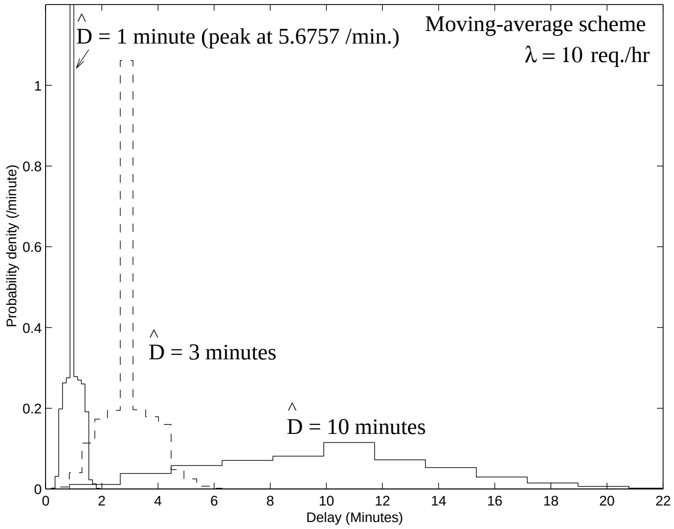
ever, the auto-gated scheduling offers delay bound to the users and eliminates user's uncertain wait. Furthermore, for a given λ, the maximum number of concurrent streams is T_h/W, while the maximum number of concurrent streams for the batch-size based scheme can be very high. This makes auto-gated scheduling the most attractive among all the schemes considered.

Moving-average scheme

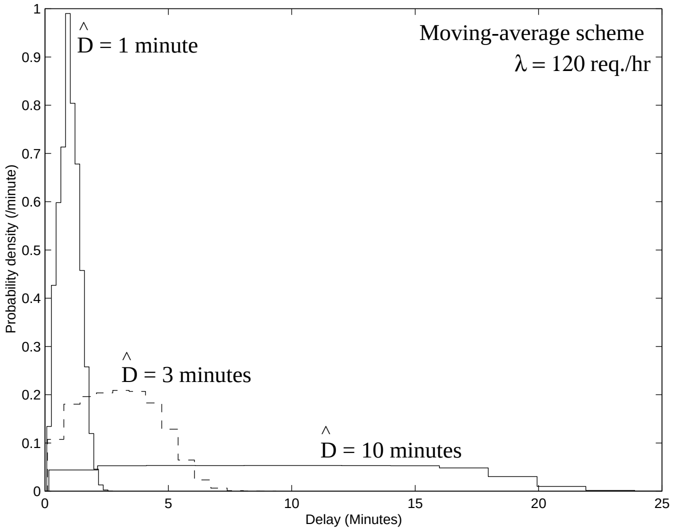
In the moving-average scheme, the movie show-time is pushed forward in time each time there is a new arrival in the batch. Therefore we are interested in the user delay distribution given that a certain average delay requirement $\hat{D}$ is to be satisfied. In Figs. 3.24(a) and 3.24(b), we show the user delay distribution with low arrival rate ($\lambda = 10$ req./hr) and high arrival rate ($\lambda = 120$ req./hr), respectively, for $\hat{D} = 1$ minute, 3 minutes, and 10 minutes. We see that most users actually enjoy a delay equal to the averages, and almost no user suffers a delay higher than $2\hat{D}$. It is encouraging to see that even though the batching window keeps lengthening with each user arrival, users are not delayed indefinitely. In fact, as λ increases, the average arrival time approaches the mid-point between the first and the last arrival of a batch (the batching period). Therefore, with the use of Eq. (3.3), we have $\lim_{\lambda \to \infty} W = 2\hat{D}$, and the delay distribution approaches $\sim U[0, 2\hat{D}]$. This is actually apparent when $\lambda = 120$ req./hr. Hence, as the arrival rate increases, $\overline{S}$ settles to a limiting value of $T_h/(2\hat{D})$.

In Fig. 3.25, we show the batch-size distribution for the moving-average scheme, with low arrival rate (Fig. 3.25(a)) and high arrival rate (Fig. 3.25(a)). For each arrival rate, we see that the distribution spreads as the targeted average delay increases. As arrival rate increases, the distribution becomes more and more "bell-like." Indeed, since the window size approaches $2\hat{D}$ as λ gets larger, the batch size distribution is expected to approach a Poisson distribution with mean batch size $2\lambda\hat{D} + 1$ (the one is due to the first arrival in the batch).

We show in Fig. 3.26 $\overline{S}$ versus $\hat{D}$ given λ. Also indicated is the case when $\lambda = \infty$ (i.e.,

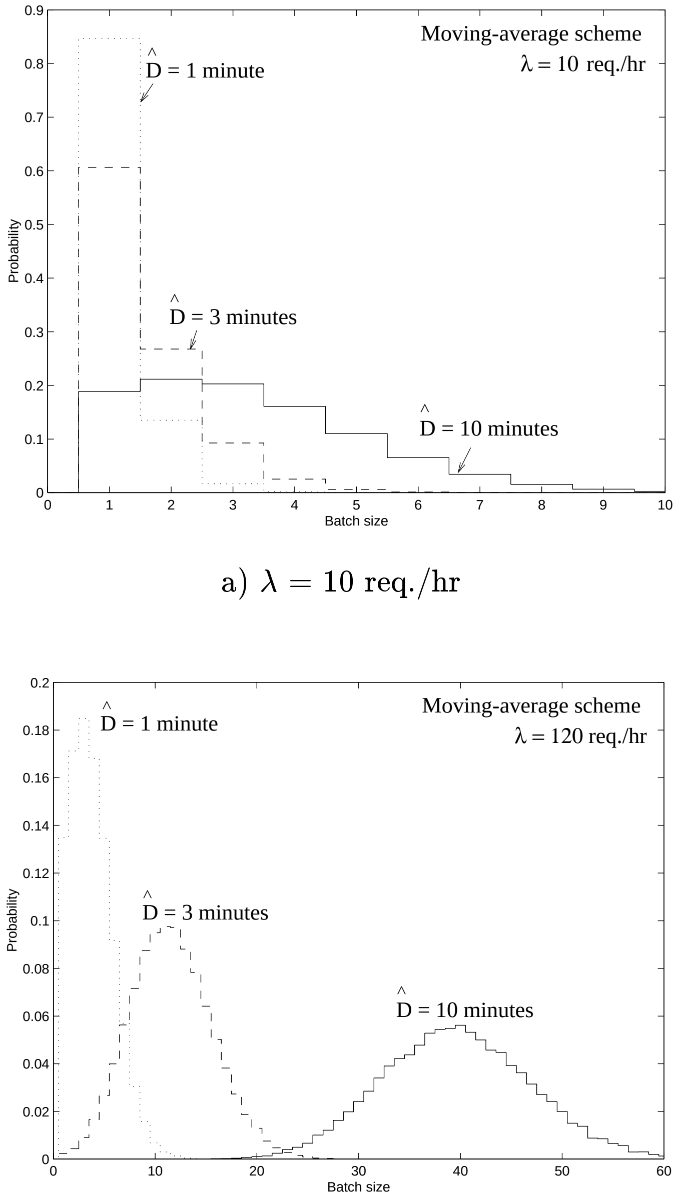


a) $\lambda = 10$ req./hr



b) $\lambda= 120$ req./hr

Figure 3.24: Delay distribution for the moving-average scheme



a) $\lambda = 10$ req./hr

b) $\lambda = 120$ req./hr

Figure 3.25: Batch-size distribution for the moving-average scheme

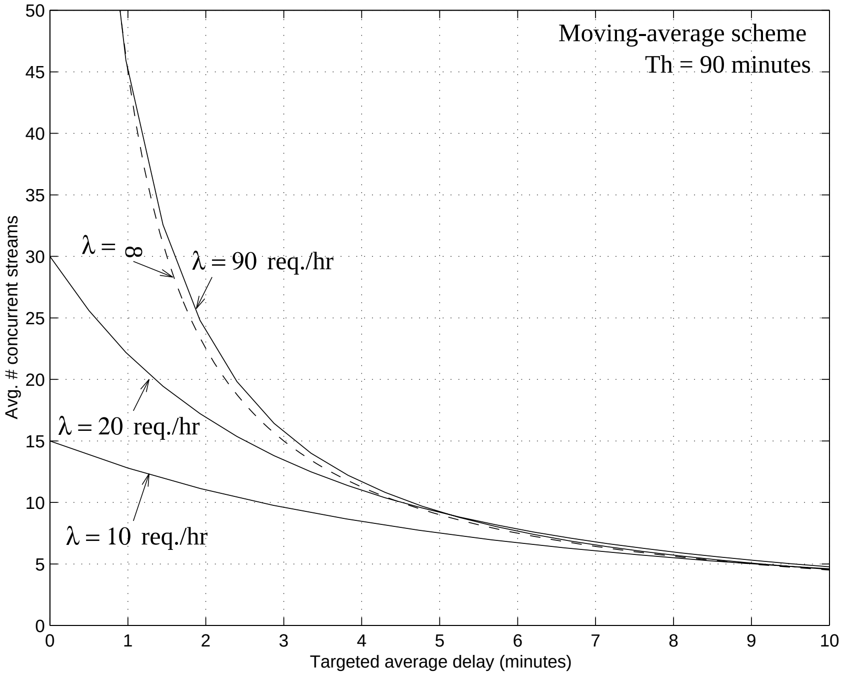


Figure 3.26: $\overline{S}$ versus $\hat{D}$ given λ for the moving-average scheme

$\overline{S} = T_h/(2\hat{D})$). For high arrival rate ($\lambda \geq 30$ req./hr), $\overline{S}$ decreases, first sharply and then more slowly, with $\hat{D}$. We see that most of the bandwidth saving is achieved when $\hat{D}$ is low ($\hat{D} \geq 3$ minutes).

In Fig. 3.27, we show $\overline{S}$ versus λ given $\hat{D}$. As expected, as λ increases, $\overline{S}$ increases and approaches the limiting values $T_h/(2\hat{D})$. For an average delay of as low as 5 minutes, only 9 streams is sufficient to serve hundreds of requests per hour.

In Fig. 3.28, we compare $\overline{S}$ as a function of λ, among the moving-average scheme, auto-gated and batch-size based schemes, given a certain average delay requirement ($\hat{D} = 3$ minutes). We see that the batch-size based scheme consumes the least bandwidth among all the schemes. In fact (from Eqs. (3.5), (3.14) and (3.15)), it is not hard to show that auto-gated scheduling consumes up to 25% (at $\lambda\hat{D} = 0.75$) more bandwidth than the batch-size based scheme, while the maximum difference in $\overline{S}$ between the batch-size based scheme and the auto-gated scheme is $0.082T_h/\hat{D}$ (at $\lambda = 1.15/\hat{D}$). Note that in all schemes, the limiting value of $\overline{S}$ as λ increases is $T_h/(2\hat{D})$. We see that only 15 streams is needed to provide $\hat{D} = 3$ minutes.

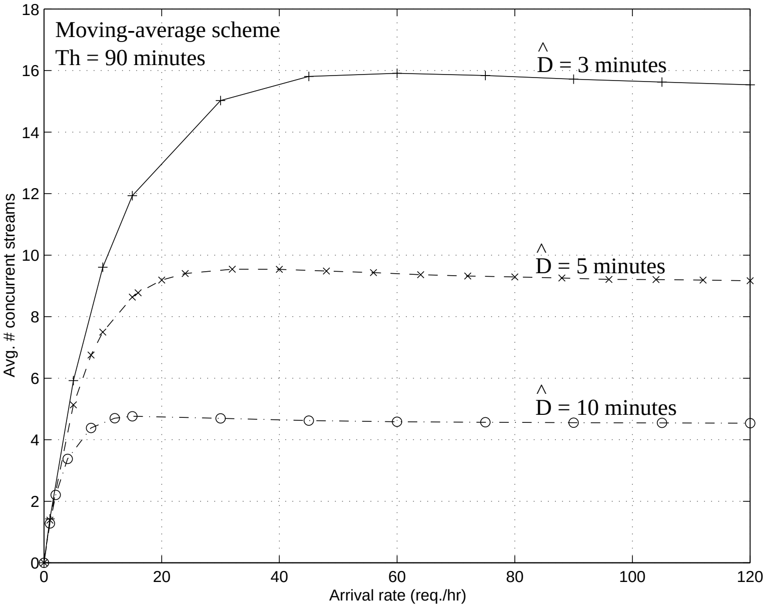


Figure 3.27: $\overline{S}$ versus λ given $\hat{D}$ for the moving-average scheme

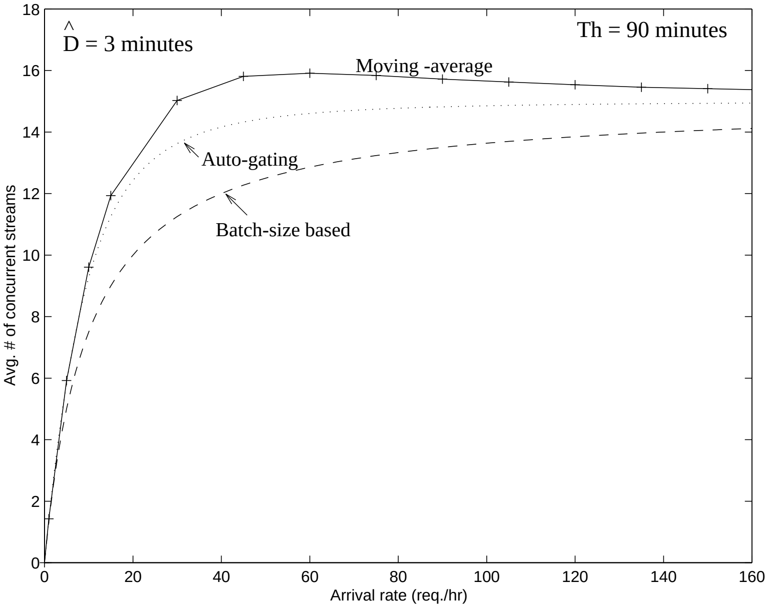


Figure 3.28: $\overline{S}$ versus λ

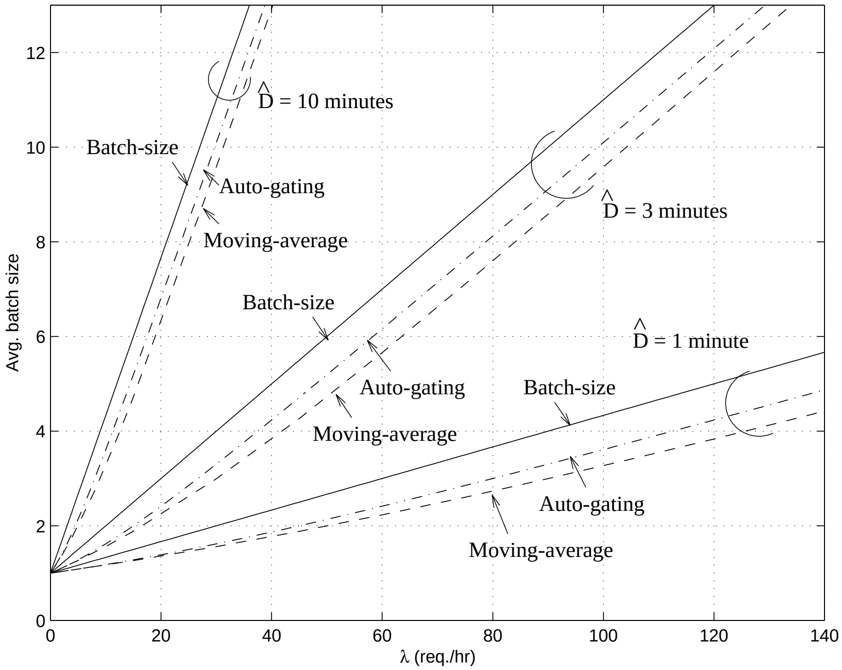


Figure 3.29: $\overline{N}$ versus λ given $\hat{D}$

In Fig. 3.29, we compare $\overline{N}$ as a function of λ given a certain $\hat{D}$. We see that there is not much difference in the batch sizes among all the schemes.

3.3.4 Adaptive scheme

Description

We now introduce a new adaptive scheme which combines the key advantage of the window-size based schemes (namely guaranteed delay) and the advantage of the batch-size based scheme (namely guaranteed per stream revenue) by ensuring that when the arrival rate is sufficiently high (and hence profit can be easily achieved), the system guarantees fairly low delay to the users; but when arrival rate is not so high, the system guarantees a certain level of profit as long as user's delay does not exceed a certain bound. The scheme therefore balances service quality (in terms of the delay user experienced) and system profit adaptively.

We consider that user satisfaction is high if the delay experienced is below a certain value

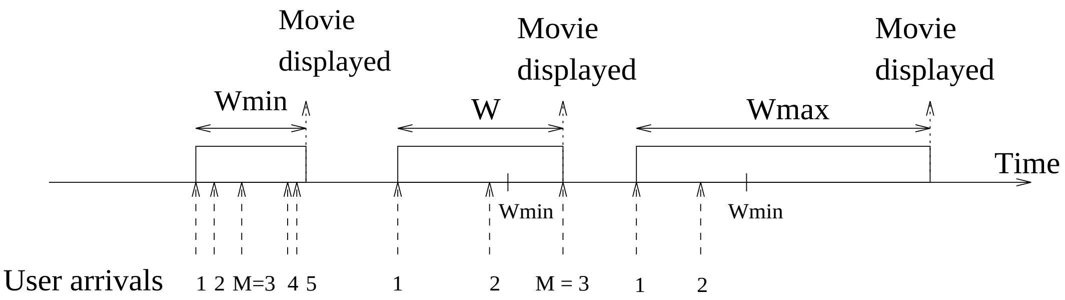


Figure 3.30: Adaptive batching scheme ($M = 3$)

D_{min}. We also consider that there is a delay $D_{max} > D_{min}$ beyond which user satisfaction is very low and, for all practical purposes, the probability of user reneging is high. Delays between D_{min} and D_{max} are tolerated and within this range no reneging is likely, but the user satisfaction is not high. As an example, D_{min} may be in the range 3–7 minutes while D_{max} may be in the range 15–40 minutes.

According to this user satisfaction model, the adaptive scheme is shown in Fig. 3.30. The scheme has three parameters: $M \geq K$, $W_{min} = D_{min}$, and $W_{max} = D_{max}$, and operates as follows: If M users arrive within W_{min}, the system keeps batching until W_{min} is reached, thereby increasing the profitability beyond the minimum M; if W_{min} is reached before M users are batched, the batching window is extended until either M or W_{max} is reached, whichever occurs first. Thus, when the arrival rate drops, the system tries to maintain profitability by using batch-size based scheme with $M \geq K$; but since users may renege if they wait higher than D_{max}, a maximum batching window of W_{max} is imposed (even if there are fewer than M users arriving within the window of size W_{max}, the movie is shown anyway). Users, in this system, may not know the exact video show time; however the show time is guaranteed to be in the range (W_{min}, W_{max}) following the first arrival in the batch.

Analysis of the scheme

Let W denote the batching period. It is a random variable with range $[W_{min}, W_{max}]$. Let α be the probability that $W = W_{min}$; it is equal to the probability that more than $M - 1$

users (excluding the first one in the batch) arrive within W_{min}, and is then expressed by:

$$\alpha = P(W = W_{min}) = 1 - \sum_{m=1}^{M-1} \frac{(\lambda W_{min})^{m-1}}{(m-1)!} e^{-\lambda W_{min}}. \tag{3.16}$$

Let β be the probability that $W = W_{max}$. It is equal to the probability that fewer than $(M-1)$ requests arrive (excluding the first request in the batch) within W_{max}, and is then expressed by,

$$\beta = P(W = W_{max}) = \sum_{m=1}^{M-1} \frac{(\lambda W_{max})^{m-1}}{(m-1)!} e^{-\lambda W_{max}}. \tag{3.17}$$

Let γ be the probability that $W_{min} < W < W_{max}$. It is equal to the probability that the batch size of M is reached at $W = w$, $W_{min} < w < W_{max}$, and is then expressed by,

$$\gamma = \int_{W_{min}}^{W_{max}} g_W(w) dw, \tag{3.18}$$

where $g_W(w)$ is given in Eq. (3.10). Clearly, γ is also given by $\gamma = 1 - \alpha - \beta$.

The distribution of N is given by,

$$P(N = i) = \begin{cases} \frac{(\lambda W_{max})^{i-1}}{(i-1)!} e^{-\lambda W_{max}} & \text{for } 1 \le i \le M-1, \\ \frac{(\lambda W_{min})^{M-1}}{(M-1)!} e^{-\lambda W_{min}} + \gamma & \text{for } i = M, \\ \frac{(\lambda W_{min})^{i-1}}{(i-1)!} e^{-\lambda W_{min}} & \text{for } i \ge M+1, \end{cases} \tag{3.19}$$

which can be used to find $\overline{N}$.

$\overline{N}$ may also be calculated by conditioning on W. Let $\overline{N}_\alpha = E[N|W = W_{min}]$, $\overline{N}_\beta = E[N|W = W_{max}]$, and $\overline{N}_\gamma = E[N|W_{min} < W < W_{max}]$. Then $\overline{N} = \alpha\overline{N}_\alpha + \beta\overline{N}_\beta + \gamma\overline{N}_\gamma$, where

$$\begin{aligned} \overline{N}_\alpha &= \sum_{m=M}^{\infty} m \frac{(\lambda W_{min})^{m-1}}{\alpha(m-1)!} e^{-\lambda W_{min}}, & (3.20) \\ \overline{N}_\beta &= \sum_{m=1}^{M-1} m \frac{(\lambda W_{max})^{m-1}}{\beta(m-1)!} e^{-\lambda W_{max}}, & (3.21) \\ \overline{N}_\gamma &= M. & (3.22) \end{aligned}$$

We get the user delay distribution by again conditioning on W. Given that $W = W_{min}$ and there are m $(m \ge M)$ requests in the batch, one (the first one in the batch) would have delay W_{min}, and the remaining $(m-1)$ of them would have a delay uniformly distributed $\sim U[0, W_{min}]$. Therefore, for $x \leq W_{min}$, the user delay distribution is given by,

$$\begin{aligned} f_D(x|W=W_{min}) &= \frac{1}{\overline{N}_\alpha} \sum_{m=M}^{\infty} \frac{(\lambda W_{min})^{m-1}}{\alpha(m-1)!} e^{-\lambda W_{min}} \cdot \\ & \quad \left((m-1)\tfrac{1}{W_{min}} + \delta(x - W_{min})\right). \end{aligned} \tag{3.23}$$

Similarly, given that a batching window is of size W_{max}, user delay distribution is ($x \leq W_{max}$),

$$\begin{aligned} f_D(x|W=W_{max}) &= \frac{1}{\overline{N}_\beta} \sum_{m=1}^{M-1} \frac{(\lambda W_{max})^{m-1}}{\beta(m-1)!} e^{-\lambda W_{max}} \cdot \\ & \quad \left((m-1)\tfrac{1}{W_{max}} + \delta(x - W_{max})\right). \end{aligned} \tag{3.24}$$

Given that $W = w$, $W_{min} < w < W_{max}$ (and hence the batch size is M), a user would have delay w, $(M-2)$ users would have delay $\sim U[0, w]$, and a user will have zero delay. Therefore, for $0 \leq x \leq w$,

$$f_D(x|W_{min} < w < W_{max}) = \frac{1}{M}\delta(x-w) + \frac{M-2}{M}\frac{1}{w} + \frac{1}{M}\delta(x).$$

Removing the condition on w, for $x \leq W_{min}$, the delay distribution is given by

$$\begin{aligned} f_D(x) &= \frac{1}{\overline{N}} \Big\{ \beta \overline{N}_\beta f_D(x|W = W_{max}) + \\ & \quad \alpha \overline{N}_\alpha f_D(x|W = W_{min}) + \\ & \quad \gamma M \int_{W_{min}}^{W_{max}} f_D(x|W_{min} < w < W_{max}) \frac{g_W(w)}{\gamma} dw \Big\} \\ &= \frac{1}{\overline{N}} \left\{ \sum_{m=1}^{M-1} \frac{(\lambda W_{max})^{m-1}}{(m-1)!} e^{-\lambda W_{max}} (m-1) \frac{1}{W_{max}} + \right. \\ & \quad \sum_{m=M}^{\infty} \frac{(\lambda W_{min})^{m-1}}{(m-1)!} e^{-\lambda W_{min}} \left(\frac{m-1}{W_{min}} + \delta(x - W_{min}) \right) \\ & \quad \left. + (M-2) \int_{W_{min}}^{W_{max}} \frac{g_W(w)}{w} dw + \gamma \delta(x) \right\}. \end{aligned} \tag{3.25}$$

For $W_{min} < x \leq W_{max}$, the distribution is,

$$f_D(x) = \frac{1}{\overline{N}} \left\{ \sum_{m=M}^{\infty} \frac{(\lambda W_{min})^{m-1}}{(m-1)!} e^{-\lambda W_{min}} (m-1) \frac{1}{W_{min}} + \right.$$

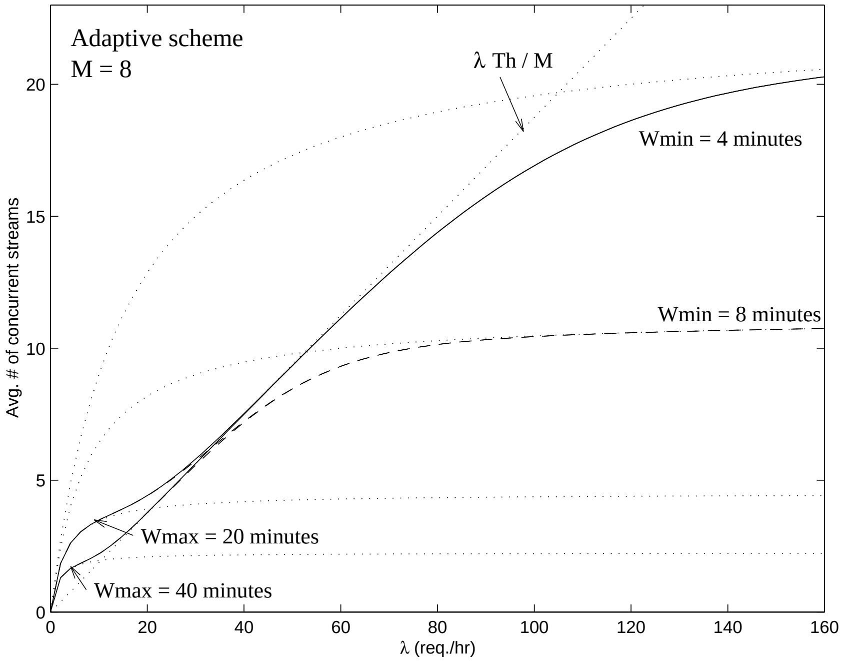


Figure 3.31: $\overline{S}$ versus λ for the adaptive scheme

$$\sum_{m=1}^{M-1} \frac{(\lambda W_{max})^{m-1}}{(m-1)!} e^{-\lambda W_{max}} \left(\frac{m-1}{W_{max}} + \delta(x - W_{max}) \right)$$
$$+ g_W(x) + (M-2) \int_x^{W_{max}} \frac{g_W(w)}{w} dw \Bigg\}. \qquad (3.26)$$

$\overline{D}$ may be obtained by $\int_0^{W_{max}} x f_D(x) dx$. $\overline{D}$ may also be calculated by conditioning on W. Let $\overline{D}_\alpha = E[D|W = W_{min}]$, $\overline{D}_\beta = E[D|W = W_{max}]$, and $\overline{D}_\gamma = E[D|W_{min} < W < W_{max}]$. We have $\overline{D} = (\alpha \overline{N}_\alpha \overline{D}_\alpha + \beta \overline{N}_\beta \overline{D}_\beta + \gamma M \overline{D}_\gamma)/\overline{N}$, where

$$\overline{D}_\alpha = \frac{1}{\overline{N}_\alpha} \sum_{m=M}^{\infty} \left((m-1)\frac{W_{min}}{2} + W_{min} \right) \frac{(\lambda W_{min})^{m-1}}{\alpha(m-1)!} e^{-\lambda W_{min}},$$

$$\overline{D}_\beta = \frac{1}{\overline{N}_\alpha} \sum_{m=1}^{M-1} \left((m-1)\frac{W_{max}}{2} + W_{max} \right) \frac{(\lambda W_{max})^{m-1}}{\beta(m-1)!} e^{-\lambda W_{max}},$$

$$\overline{D}_\gamma = \frac{1}{M} \int_{W_{min}}^{W_{max}} \frac{g_W(w)}{\gamma} M \frac{w}{2} dw.$$

Numerical results

We plot in Fig. 3.31 $\overline{S}$ versus λ. We also show (in light dotted lines) the corresponding $\overline{S}$ for the batch-size based scheme (with M set to the value used in the adaptive scheme) and

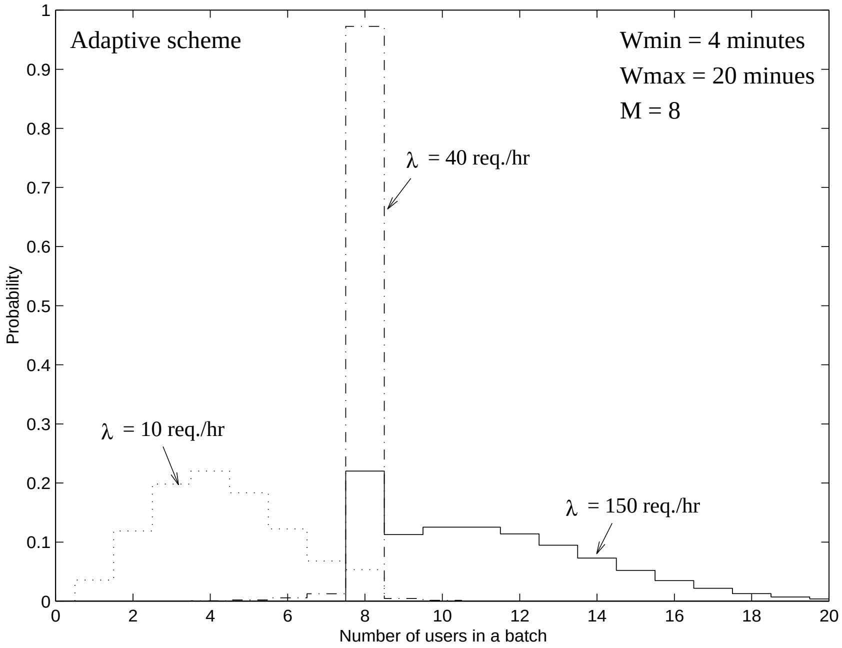


Figure 3.32: Batch-size distribution for the adaptive scheme ($M = 8$, $W_{min} = 4$ minutes, $W_{max} = 20$ minutes)

for the auto-gated scheme (with W set to the various values of W_{min} and W_{max} used in the adaptive scheme). We see that as λ increases, the average number of concurrent streams first behaves as in the auto-gated scheme with window size W_{max}, then it switches over to the batch-size based scheme (shown as the straight line), and then finally, as the arrival rate increases further, it behaves as in the auto-gated scheme with window size W_{min}.

In Fig. 3.32, we show the batch size distribution for three values of λ's: low value ($\lambda = 10$ req./hr), intermediate value ($\lambda = 40$ req./hr) and large value ($\lambda = 150$ req./hr). We see that when λ is small, the distribution is similar to the Poisson distribution (as in the auto-gated scheme), and when λ =40 req./hr, most of the batches are of size M, and when λ is further increased to 150 req./hr, the probability of a batch size equal to M decreases, and the distribution spreads to values larger than M.

In Fig. 3.33, we show the average batch size $\overline{N}$ as a function of λ; we also show in light dotted lines the same curves for the auto-gated scheme with the corresponding parameters. Similar to $\overline{S}$, we see that as λ increases, $\overline{N}$ follows the auto-gated scheme curve with

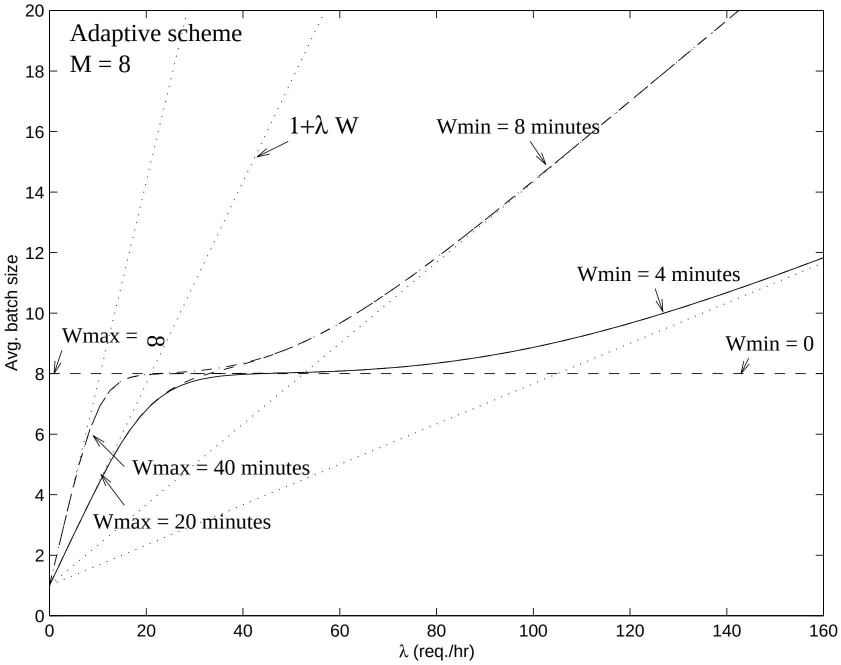
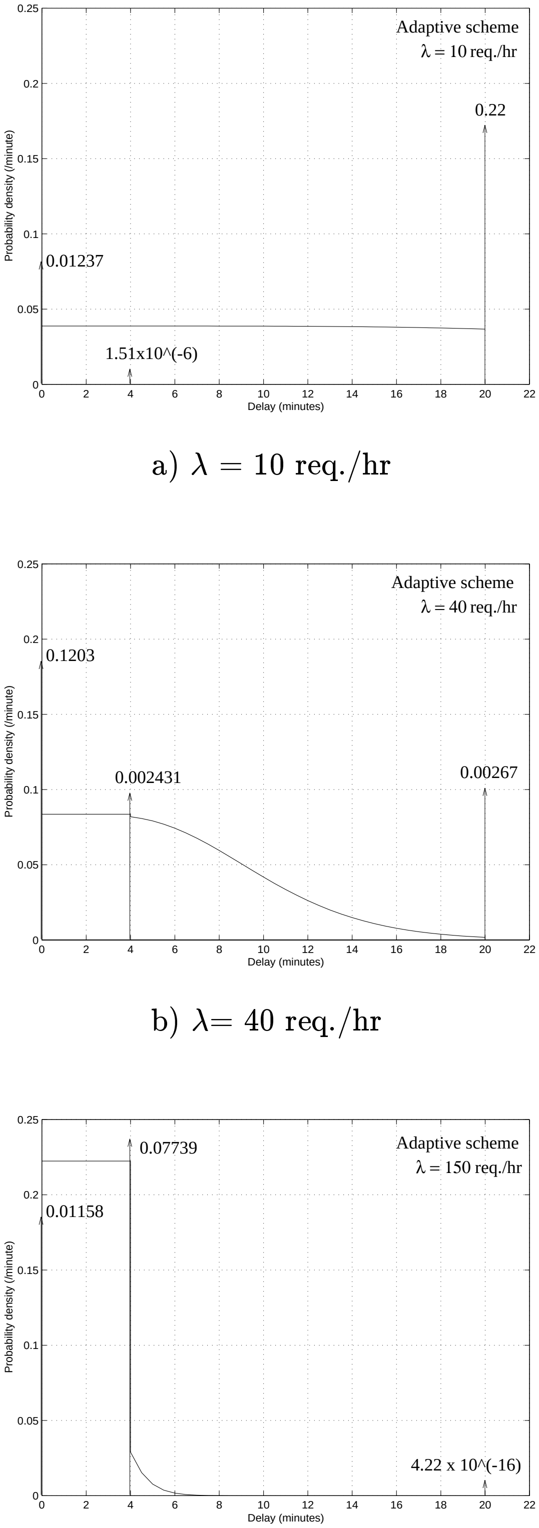


Figure 3.33: $\overline{N}$ versus λ for the adaptive scheme

window size W_{max}, and then the batch-size plateau with the corresponding parameter of M, and finally the auto-gated scheme again with window size W_{min}. Note that the portion of the graph lying above $N = M$ corresponds to the profitable region. For a time varying arrival rate, the total system profitability depends on how often the process visits the upper portion (with $\overline{N} \geq M$) and the lower portion.

We now examine the distribution of the user delay. In Fig. 3.34, we show the distribution for various values of the arrival rate. We see that when the arrival rate is low (Fig. 3.34(a)), the distribution bears great resemblance with that of the auto-gated scheme with window size W_{max}. Figure 3.34(b) shows the distribution when the arrival rate is at an intermediate value. We see that there are more users with zero delay (those ending the batching process by having accumulated M users) and W_{min}; on the other hand, the number of users with delay W_{max} decreases. In Fig. 3.34(c), we show the distribution for relatively high λ. We see that more users have delay W_{min}, but the number of users having zero delay (the batch-size based case) and W_{max} decreases greatly.

In Fig. 3.35, we show as a function of λ the probability that the batching window is of



a) λ = 10 req./hr

b) λ= 40 req./hr

c) λ = 150 req./hr

Figure 3.34: Delay distribution with (W_{min}, M, W_{max}) = (4 minutes, 8, 20 minutes)

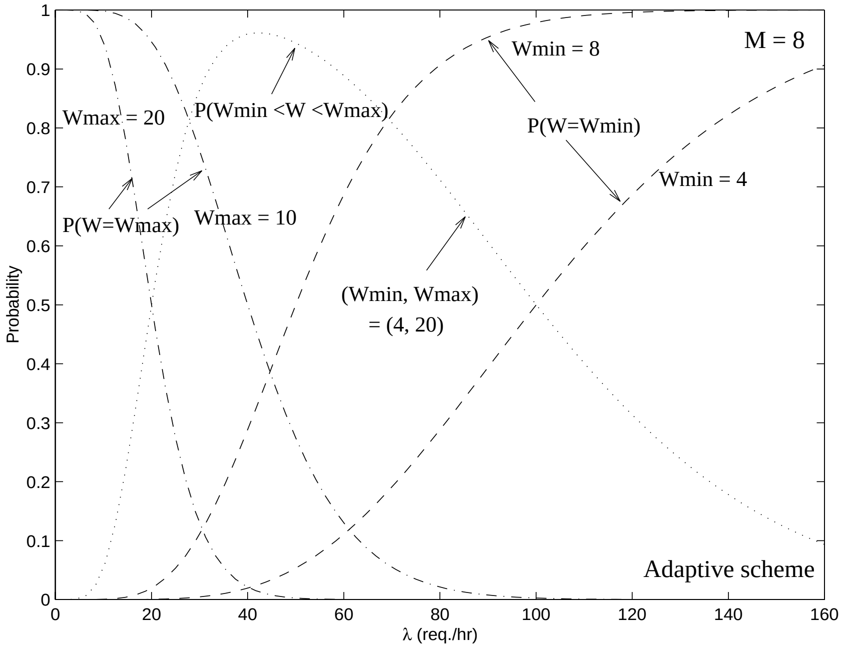


Figure 3.35: $P(W = W_{min})$ and $P(W = W_{max})$ versus λ. Also shown is $P(W_{min} < W < W_{max})$ for (W_{min}, W_{max}) = (4 minutes, 20 minutes)

size W_{max} (i.e., a batch size $< M$), of size W_{min} (i.e., batch size $\geq M$) and the probability that it is in between (i.e., batch size equal to M). We see that as the arrival rate increases, the window size changes from being mainly of size W_{max}, to being between W_{min} and W_{min}, and finally to being equal to W_{min}.

In Fig. 3.36, we plot the average delay as a function of λ. We see that when the arrival rate is low, the average delay is that of auto-gated scheme with window size W_{max}, and as the arrival rate increases, it switches to that of auto-gated scheme with window size W_{min}.

3.3.5 Minimum bandwidth to meet a certain user's delay requirement

We note from the above discussion that the window-based scheme has the distinct feature of guaranteeing a maximum user delay (and thus fits the step function model of user reneging behaviour). Thus if the users expect a delay bounded by a maximum value $\hat{d}$, then a

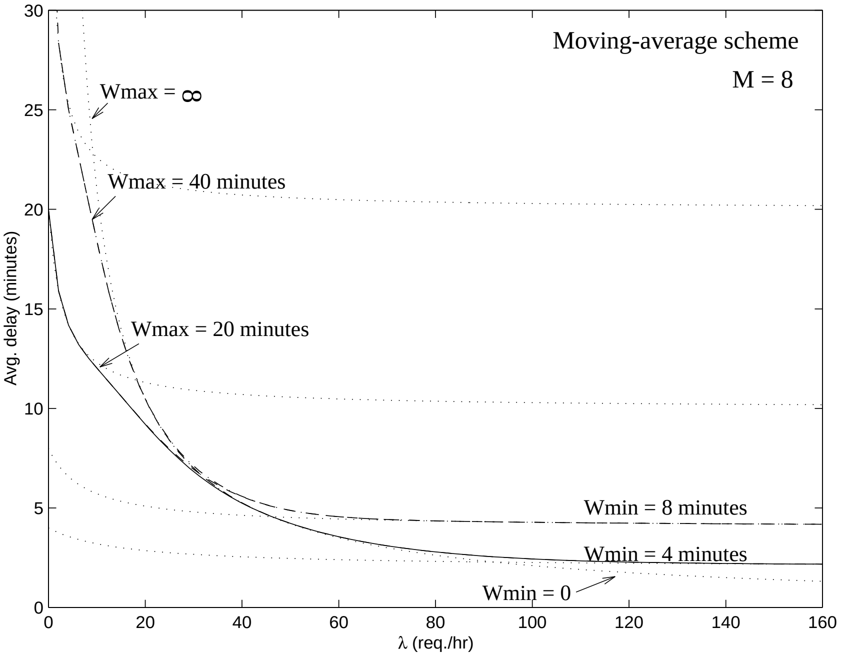


Figure 3.36: Average delay versus λ for the adaptive scheme

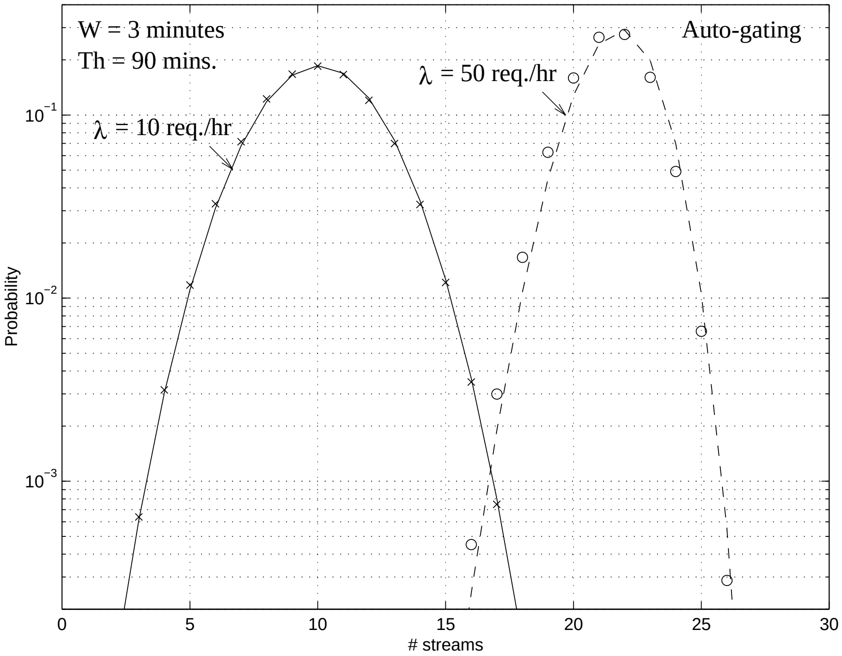


Figure 3.37: Stream distribution for $W = 3$ minutes

window-based scheme with $W = \hat{d}$ meets the delay expectation. The number of streams required S, however, varies over time in the range between 0 and $\lceil T_h/\hat{d} \rceil$, and can reach the maximum $\hat{N} = T_h/\hat{d}$, albeit with very small probability. This is illustrated in Figs. 3.37 and 3.38, in which we show the stream distribution for the auto-gated scheduling $\lambda = 10$ req./hr and 50 req./hr, with $W = 3$ minutes and 6 minutes respectively. (The discrete points are from the simulation data and continuous lines are from analysis.)

Since in some cases, the total number of available channels may be limited (e.g., leased channels or server's streaming capacity), the question arises as to what should the minimum number of streams N_s^* allocated to a movie be in order to meet a certain user delay requirement? More specifically, we seek to design the system such that the probability of a user's delay D exceeding a certain value $\hat{d}$ to be less than a small values ε, i.e., $P(D > \hat{d}) < \varepsilon$. Since we are designing a system meeting user's delay expectation, we limit ourselves to window-based schemes and consider the reneging probability to be negligible (or equivalently, if the reneging behaviour is modelled by a step function with a delay limit equal to

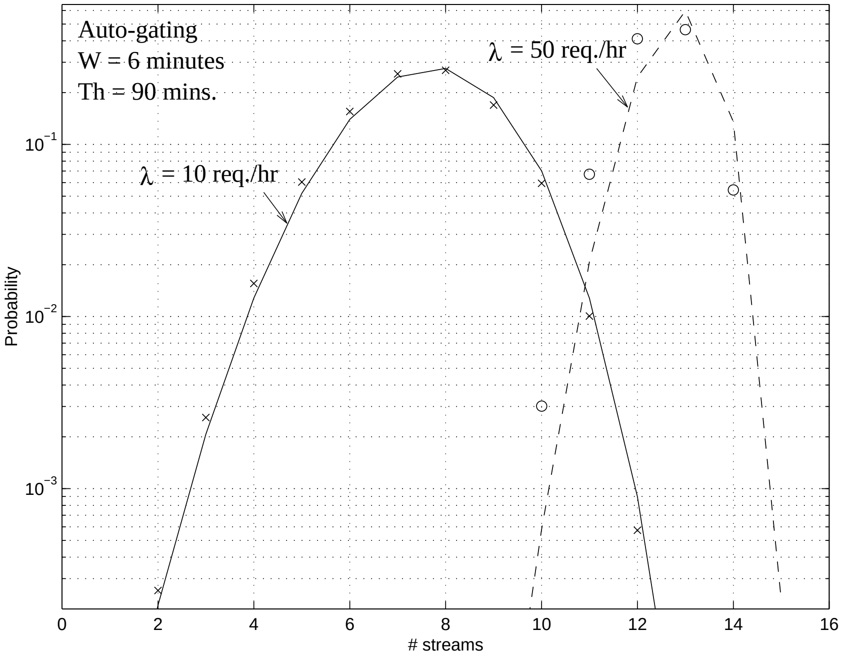


Figure 3.38: Stream distribution for $W = 6$ minutes

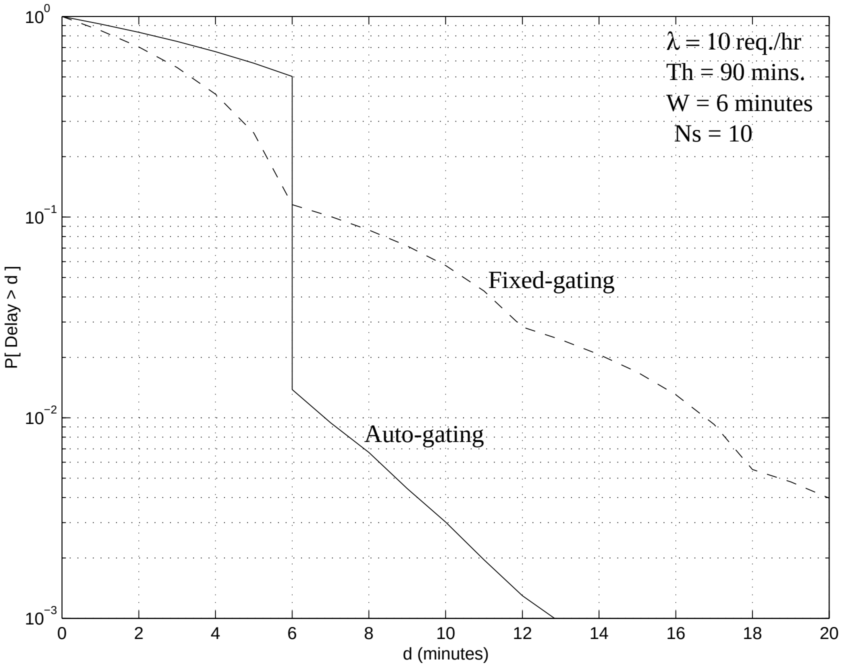


Figure 3.39: $P(D > d)$ vs. d for auto-gated scheduling and fixed-gating

$\hat{d}$, then ε would represents the probability that a user reneges).

We again consider a movie of duration T_h minutes, to which N_s streams are allocated. The arrival process of the requests is Poisson with rate λ. In this study, we use simulation.

Bandwidth planning

As an example, we consider $T_h = 90$ minutes, $\hat{d} = 6$ minutes and $\varepsilon \approx$1%. In this case, $\hat{N} = 15$ streams. Clearly, if N_s is set to 15, then $P(D > \hat{d}) = 0$. However, when $N_s < 15$, it is probable that at the end of a batching window, all the channels are already occupied and users have to wait longer than $\hat{d}$.

We begin by comparing two window-based schemes: fixed-gating and auto-gating. In Fig. 3.39, we show the complimentary delay distribution $P(D > d)$ for the auto-gating and fixed-gating, with $N_s = 10$ and $W = 6$ minutes. As expected, we see that the auto-gating has a much lower tail than the fixed gating, clearly establishing that it offers better bandwidth utilization. Hence, we consider only the auto-gating in the following.

We first look at how W affects $P(D > d)$ given $N_s < \hat{N}$. Thus, we show in Fig. 3.40

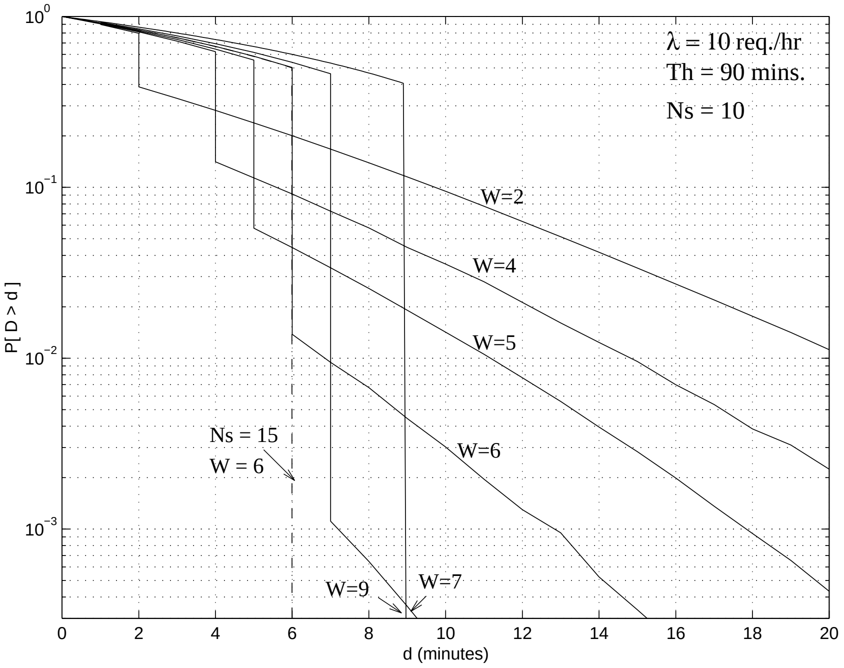


Figure 3.40: $P(D > d)$ with $N_s = 10$ and different W's

$P(D > d)$ versus d, with N_s set to a fixed value (namely $N_s = 10$) and various values of W when $\lambda = 10$ req./hr; the dotted line represents $P(D > d)$ for $N_s = 15$ and $W = 6$ minutes (for which again $P(D > \hat{d}) = 0$). We see that for each value of W, there is an impulse at $d = W$ (as the first user in a batch incurs a delay equal to W), followed by a tail (indicating that users have to wait for an available stream). We observe from Fig. 3.40 that W has a strong effect on the user delay distribution, and that when N_s is set to a certain value below $\hat{N}$, $P(D > \hat{d})$ is minimized when W is set equal to $\hat{d}$; also in this case ($N_s = 10$), $P(D > \hat{d}) \approx 10^{-2}$, meaning that for a small value of ε ($\varepsilon \approx 10^{-2}$), the number of streams required can be reduced from the maximum of 15 to 10.

Note that the area under the graph is the average delay $\overline{D}$ of the users. User delay D is the sum of the gating delay D_G (the time waiting for the batching window to end) and the time waiting for an available stream D_s. As W decreases, D_G decreases while D_s increases (since streams are more likely to run out). In Fig. 3.41, we show $\overline{D}$ versus W, given N_s. Also shown is the average delay with infinite bandwidth according to Eq. (3.7). We see that indeed, as W decreases from T_h/N_s, $\overline{D}$ first decreases (due to the decrease in gating

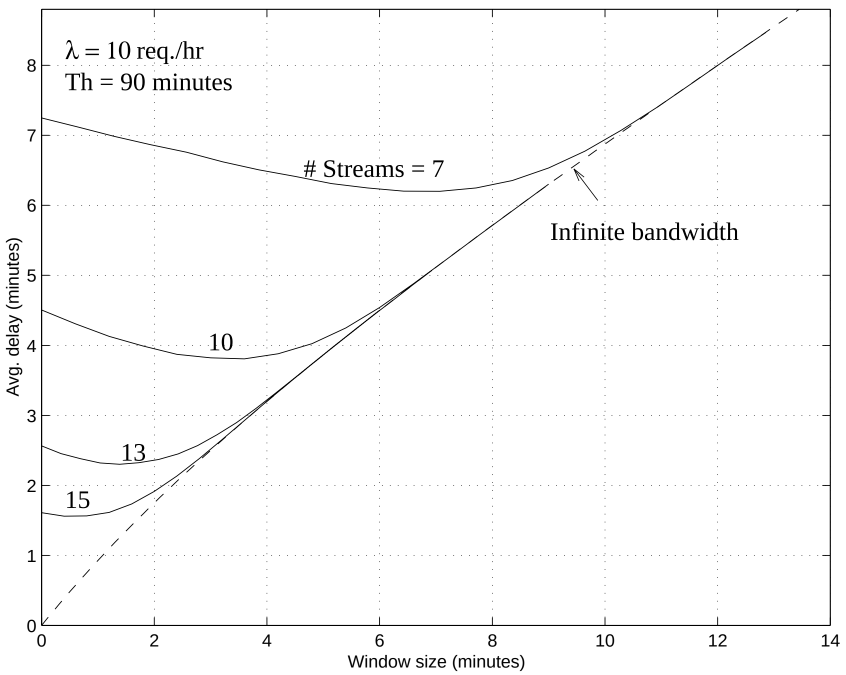


Figure 3.41: $\overline{D}$ versus W given N_s

delay), and then increases (due to the waiting time for an available stream). For $N_s = 10$, we see that at minimal $\overline{D}$ (corresponding to $W = 4$ minutes), the delay distribution has an undesirable high tail (ref. Fig. 3.40).

We show in Fig. 3.42 how $P(D > \hat{d})$ varies with $\hat{d}$ given N_s. We see that as $\hat{d}$ decreases, $P(D > \hat{d})$ increases first sharply then gradually. We can also read from the graph the minimal expectation $\hat{d}$ one should target given certain N_s and ε.

To see the effect of N_s on $P(D > d)$, we show in Fig. 3.43 $P(D > d)$ versus d for different values of N_s, considering two cases for $\hat{d}$ ($\hat{d} = 6$ minutes and 15 minutes) and setting $W = \hat{d}$ in each case. Clearly, we see that to meet smaller value of ε, the system requires larger value of N_s. We also see that when $\hat{d}$ is equal to 15 minutes, the bandwidth saving is quite low, since for $\hat{d} = 15$ minutes, $\hat{N} = 6$ and the bandwidth is already well-utilized. We show in Fig. 3.44 $P(D > \hat{d})$ versus N_s, from which given ε and $\hat{d}$, N_s^* can be directly read.

We show in Fig. 3.45 N_s^* versus $\hat{d}$ given ε, still for $\lambda = 10$ req./hr. Also shown is the maximum number of streams required. We see that for $\hat{d} \leq 6$ minutes, N_s^* can be markedly lower than $\hat{N}$. However, as $\hat{d}$ increases, such saving decreases quickly. When

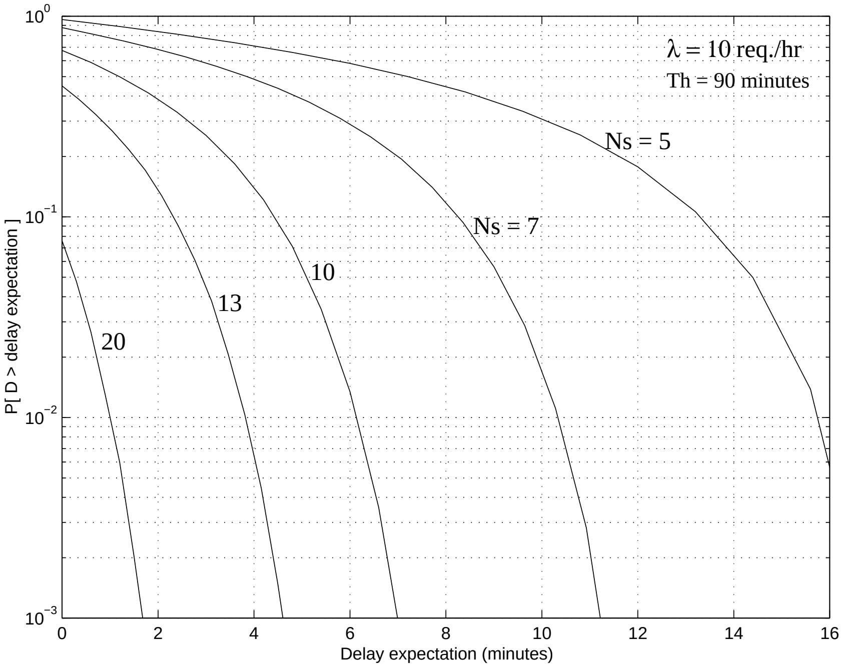


Figure 3.42: $P(D > \hat{d})$ versus $\hat{d}$ given N_s

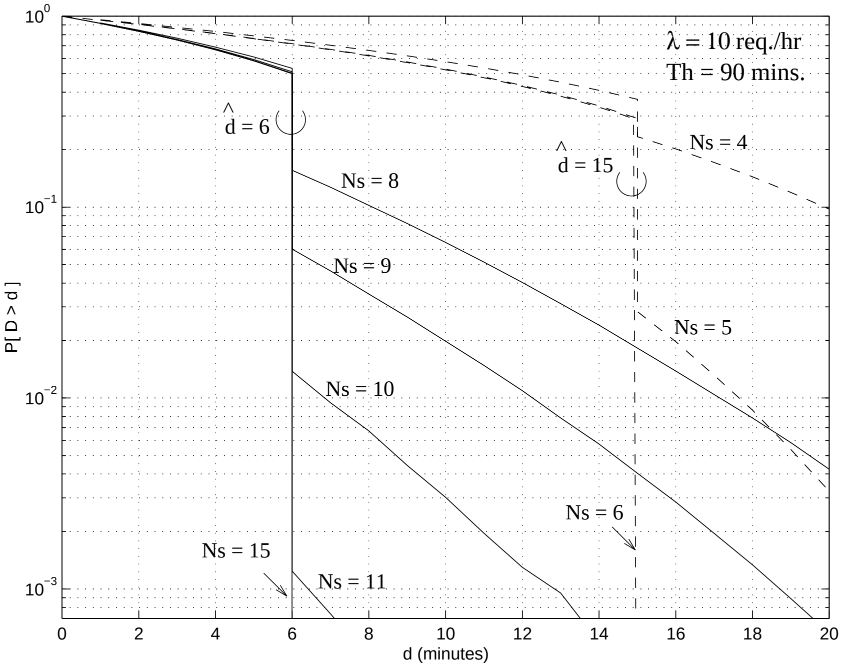


Figure 3.43: $P(D > d)$ versus d for different values of N_s

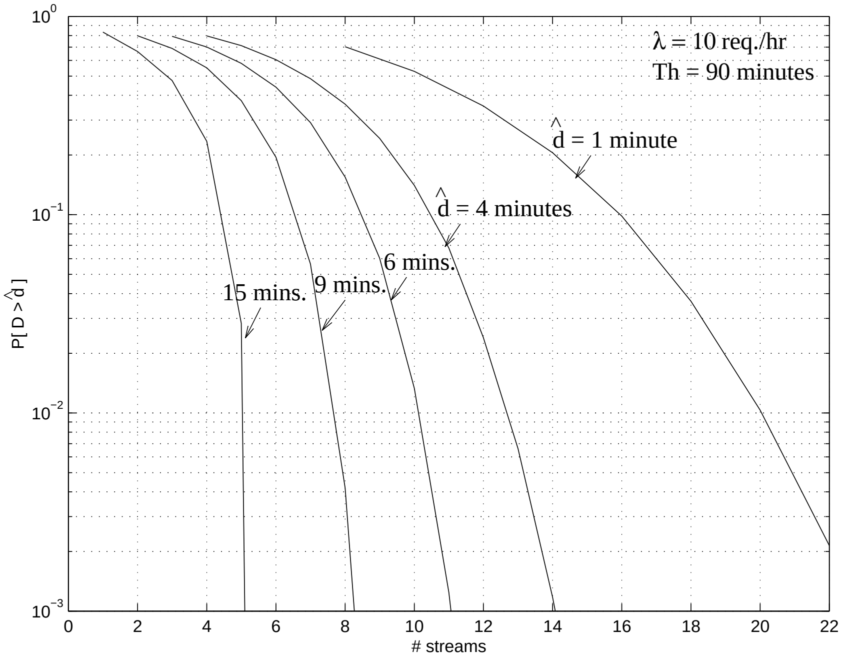


Figure 3.44: $P(D > \hat{d})$ versus N_s given $\hat{d}$

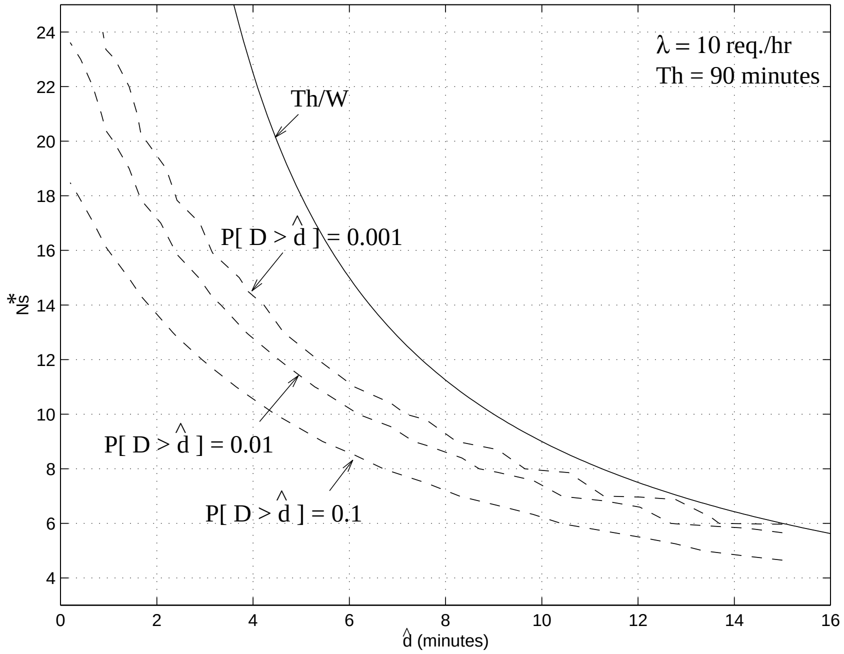


Figure 3.45: N_s^* versus $\hat{d}$ given ε

$\hat{d} \geq 12$ minutes, the saving becomes insignificant.

We show in Figs. 3.46 and 3.47 the influence of an increase in λ on $P(D > d)$ and $P(D > \hat{d})$, respectively by considering $\lambda = 30$ req./hr. Obviously, as λ increases, the delay tail increases and hence we require more streams to satisfy the same delay expectation. With the same $\hat{d}$, there is no longer as much saving as before.

The above results pertain to a relatively low value of λ, namely 10 req./hr. As λ gets larger, the saving decreases since the average number of streams $\overline{S}$ required to meet a maximum delay requirement $\hat{d}$ approaches the maximum $(T_h/\hat{d})$. To illustrate this fact, we show in Fig. 3.48 how N_s^* increases with λ when $P(D > \hat{d})$ is to be kept constant, for the case $\hat{d} = 6$ minutes. We see that as λ increases, N_s^* also increases to a value close to $\hat{N} = 15$.

In summary, the saving is only significant when both λ and $\hat{d}$ are relatively low, but as one or the other gets larger, then the saving diminishes. Note that when considering movies seperately, a low value of λ for a movie means that the movie is not popular and the saving in question becomes of interest. By allowing the sharing of channels among multiple

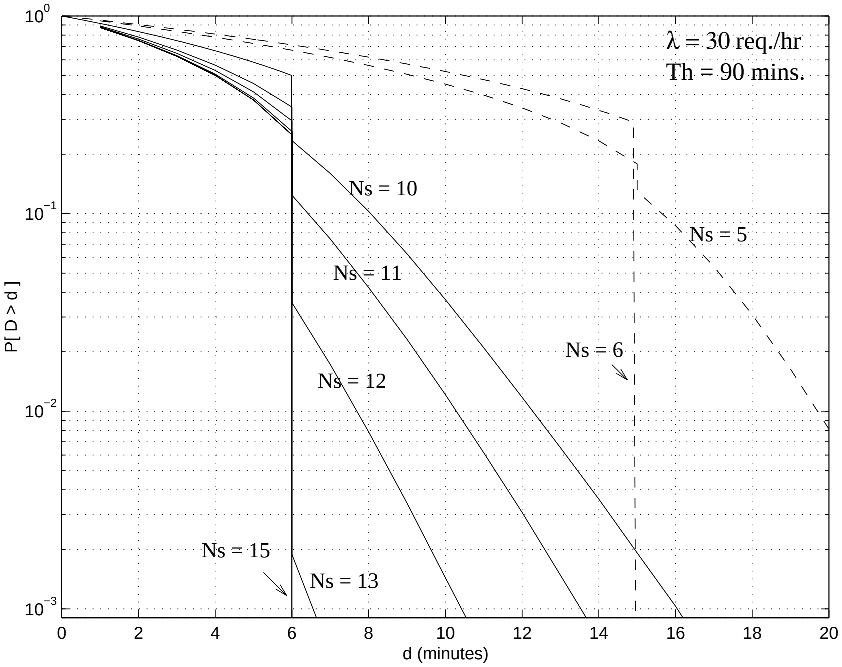


Figure 3.46: $P(D > d)$ versus d given N_s's for $\lambda = 30$ req./hr

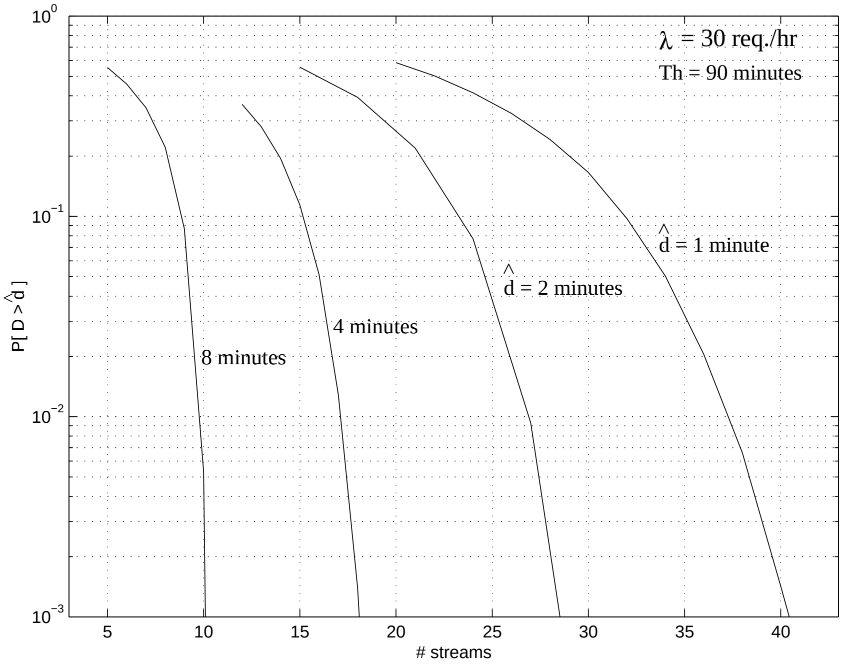


Figure 3.47: $P(D > \hat{d})$ versus N_s given $\hat{d}$ (λ =30 req./hr)

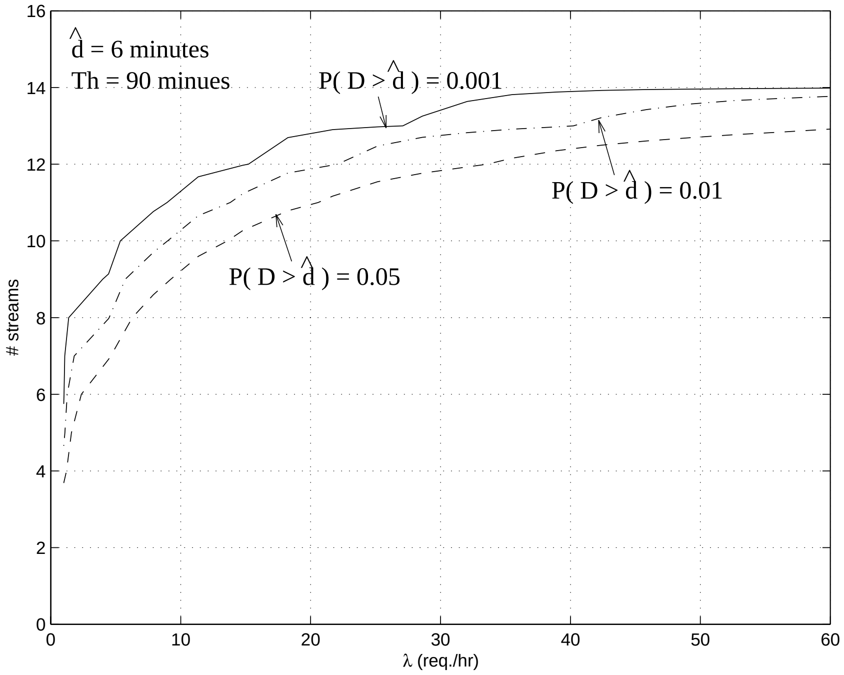


Figure 3.48: N_s^* versus λ given ε, with $\hat{d} = 6$ minutes

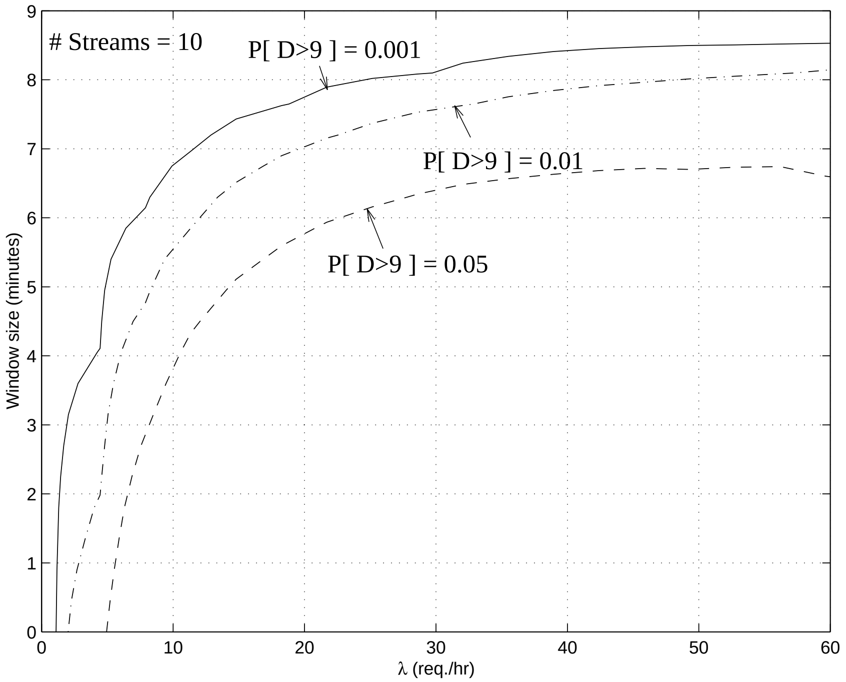


Figure 3.49: Minimum window size versus λ to guarantee a certain $P(D > \hat{d})$

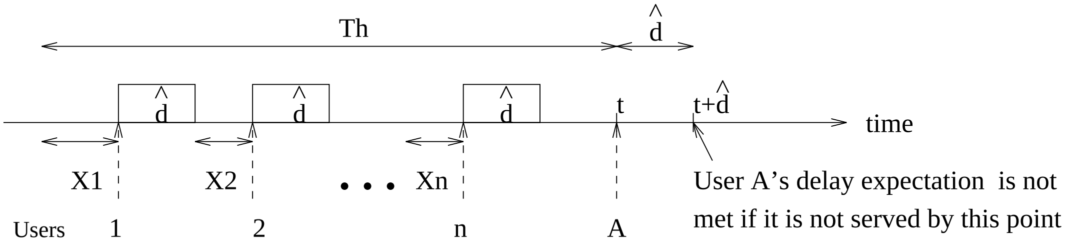

Th
^
d
^
d
^
d
^
d
t
t+^d
time
X1
X2
• • •
Xn
Users
1
2
n
A
User A's delay expectation is not met if it is not served by this point

Figure 3.50: Time diagram for analyzing $P(D > \hat{d})$

movies, additional saving may be achieved.

In Fig. 3.49, we show W versus λ so that $P(D > \hat{d})$ ($\hat{d} = 9$ minutes) is kept constant, given a certain N_s. We see that as λ increases, W first increases rapidly and then more slowly. Note that W would not go beyond T_h/N_s. Figure 3.49 can be used in the multiple movies case to plan the window size for a movie given its request rate and its assigned bandwidth.

Approximate expression for $P(D > \hat{d})$

In this section, we derive the approximate expression for for $P(D > \hat{d})$, with $W = \hat{d}$. Refer to Fig. 3.50. Consider a random arrival at time t. Its delay expectation is not met if it is not served by time $t + \hat{d}$. Such event occurs when all N_s streams are used at time t and no stream is freed between time $[t, t + \hat{d}]$, which in turns means that all the N_s streams have to be allocated between $[t - T_h, t]$. Let S be the number of concurrent streams used at time t (S is assumed stationary and hence the subscript t is dropped). To derive $P(D > \hat{d})$, we first assume infinite bandwidth and then approximate the probability by $P(S \geq N_s)$ (This approximation is obviously good when $P(S \geq N_s)$ is low, which is our case of general interest). The delay expectation is not met when $\sum_{i=1}^{N_s} X_i \leq T_h - N_s\hat{d}$. Therefore,

$$\begin{aligned} P(D > \hat{d}) &\approx P(S > N_s) \qquad (3.27) \\ &= \int_0^{T_h - N_s\hat{d}} g_{N_s}(x)dx \\ &\geq \frac{[\lambda(T_h - N_s\hat{d})]^{N_s}}{N_s!} e^{-\lambda(T_h - N_s\hat{d})}. \qquad (3.28) \end{aligned}$$

In Figs. 3.51 and 3.52, we show $P(D > \hat{d})$ versus N_s given $\hat{d}$, with $\lambda = 10$ req./hr and 30 req./hr respectively. We see that our analysis follows very well with the simulation points, especially for low $P(D > \hat{d})$ ($P(D > \hat{d}) < 0.1$), the case of practical interest). We see from the figures that the lower bound (Eq. (3.28)) may be used to approximate $P(D > \hat{d})$. Note that $P(D > \hat{d})$ decreases very fast with N_s, especially when $N_s\hat{d}$ is close to T_h.

3.4 Achieving High Profit in Providing Near Video-on-Demand Services

In this section, we consider how high profit can be achieved with the batching schemes mentioned above. We are interested in the following closely-related measures:

- The per-batch profit: The average per-batch profit is given by $(\overline{N} - K)P$, where $\overline{N}$ denotes again the average batch size;

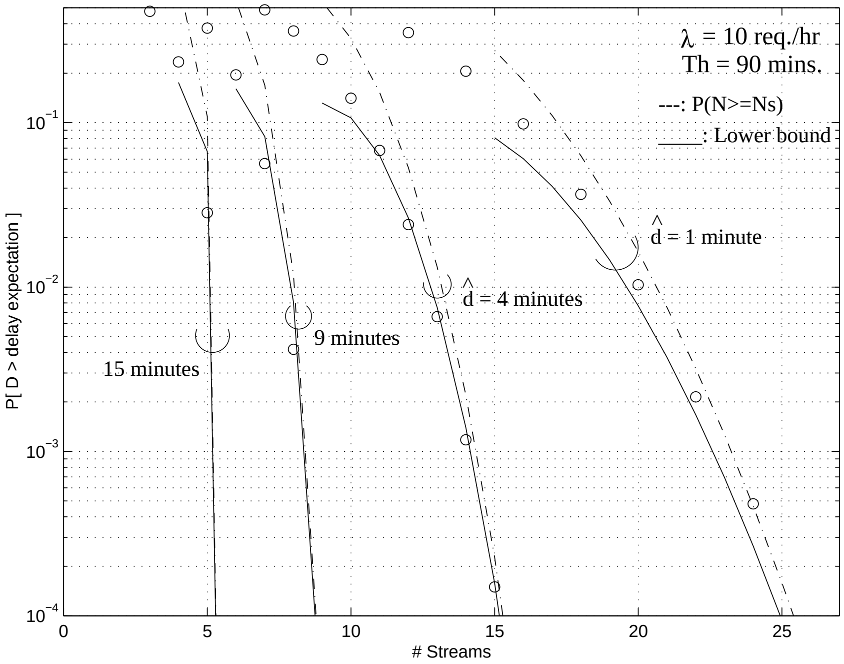


Figure 3.51: $P(D > \hat{d})$ versus N_s for $\lambda = 10$ req./hr

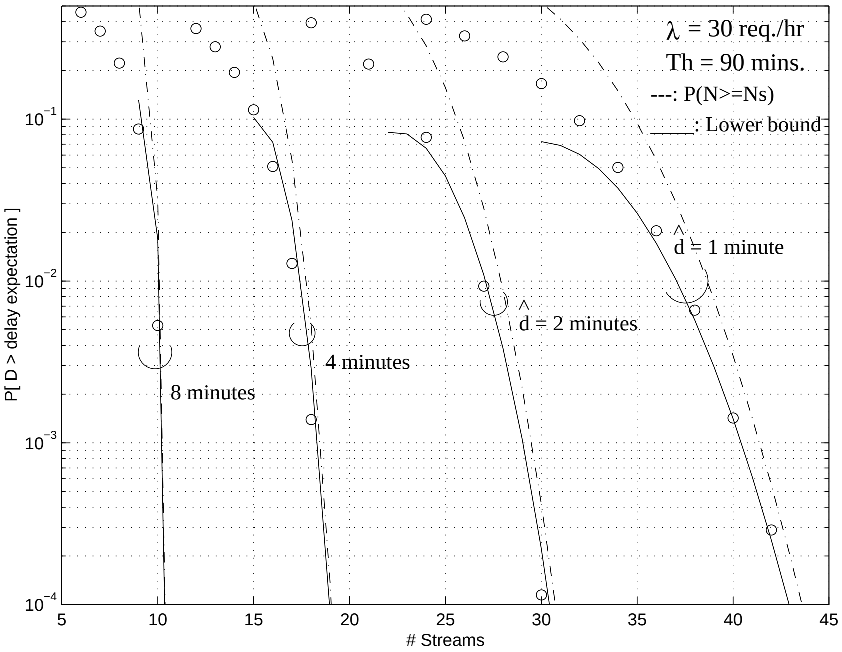


Figure 3.52: $P(D > \hat{d})$ versus N_s for $\lambda = 30$ req./hr

- The profit rate θ: Profit rate refers to the profit per unit time (dollars/minute). Denote $\overline{T}$ the average time between channel assignment to display a movie. Then θ is given by,

$$\begin{aligned}\theta &= \frac{\overline{N}P_V - (C + \alpha\overline{N})}{\overline{T}} && (3.29)\\ &= \frac{\overline{N}P - C}{\overline{T}}, && (3.30)\end{aligned}$$

  Define $\hat{\theta}$ as the normalized profit rate with respect to P. Then Eq. (3.30) becomes,

$$\hat{\theta} \triangleq \frac{\theta}{P} = \frac{\overline{N} - K}{\overline{T}}. \quad (3.31)$$

- The system throughput λ': The system throughput represents the number of accepted requests served per unit time, which is given by,

$$\lambda' = \frac{\overline{N}}{\overline{T}}, \quad (3.32)$$

  and the associated request loss rate p_L is given by,

$$\begin{aligned}p_L &= 1 - \frac{\lambda'}{\lambda} && (3.33)\\ &= 1 - \frac{\overline{N}}{\lambda\overline{T}}. && (3.34)\end{aligned}$$

- The average number of streams used $\overline{S}$: $\overline{S}$ is given by,

$$\begin{aligned}\overline{S} &= \frac{\lambda' T_h}{\overline{N}} && (3.35)\\ &= \frac{T_h}{\overline{T}}. && (3.36)\end{aligned}$$

Note that once we derive $\overline{N}$ and $\overline{T}$, λ', p_L and $\overline{S}$ are also known.

We first discuss the reneging model we used in our study.

3.4.1 User's reneging behavior

We assume that requests arrive according to a Poisson process with rate λ. We consider that each user is independently willing to wait for a period of time U such that if its requested movie is not displayed by then, it reneges. U is a random variable with a cumulative distribution $R(u) = P(U < u)$ with mean $\overline{U}$. $R(u)$ is referred to as the reneging function. For concreteness and illustrative purposes, we have considered the following three reneging functions in this paper; note that our study is not necessarily limited to these funtions:

- Exponential function — Users are always willing to wait for a minimum time $U_{min} \geq 0$; the additional waiting time above U_{min} is exponentially distributed with mean τ (the time waiting tolerance), i.e.,

  $$R(u) = \begin{cases} 0, & \text{if } 0 \leq u \leq U_{min}, \\ 1 - e^{-(u-U_{min})/\tau}, & \text{otherwise.} \end{cases} \tag{3.37}$$

  Therefore, the average time the users are willing to wait is $\overline{U} = U_{min} + \tau$. The larger τ is, the more willing are users to wait.

- Linear function — Users are willing to wait for a time U_{min} minutes, after which their waiting time is uniformly distributed between U_{min} and U_{max}. Letting $\Delta \triangleq U_{max} - U_{min}$, the reneging function is hence,

  $$R(u) = \begin{cases} 0, & \text{for } u \leq U_{min}, \\ (x - U_{min})/\Delta, & \text{for } U_{min} < u \leq U_{max}, \\ 1, & \text{otherwise.} \end{cases} \tag{3.38}$$

  In this case, $\overline{U} = U_{min} + \Delta/2$.

- Step function — Here we consider a step reneging function in which users are willing to wait for a time U_0, beyond which they renege with probability 1:

  $$R(u) = \begin{cases} 0, & \text{for } u \leq U_0, \\ 1, & \text{for } u > U_0. \end{cases} \tag{3.39}$$

  In this case, $\overline{U} = U_0$. Note that this function is the limiting case of exponential function with $\tau \to \infty$ with $U_{min} = U_0$. Owing to its simplicity, it can also be considered as an alternative approximation for any arbitrary reneging function $R(u)$ by setting U_0 to some appropriate value (such as $U_0 = \overline{U}$).

3.4.2 Auto-gated scheduling

In this scheme, the first user of a batch is forced to wait W (the window size) before it is served, and hence the scheme offering a delay bound W to the users. Note that since the first user in a batch may renege (and hence the batching window may be advanced to the next request), the show-time of the movie can no longer be advertised at the user's request. Obviously, W should be less than the maximum tolerable waiting time of the users; otherwise, a batch could not be successfully formed.

The probability p_1 that the first user in a batch reneges before the batching window W finishes is obviously given by $p_1 = R(W)$. Hence, the average time between consecutive service times $\overline{T}$ is given by,

$$\begin{aligned} \overline{T} &= \sum_{i=1}^{\infty} p_1^{i-1}(1-p_1)\frac{i}{\lambda} + W && (3.40) \\ &= \frac{1}{(1-R(W))\lambda} + W. && (3.41) \end{aligned}$$

Let $\tilde{p}$ be the probability that a request arriving within a batch stays till the end of the batching window. By conditioning on the amount of time left at its arrival until the batching window ends, $\tilde{p}$ is given by,

$$\tilde{p} = \int_0^W (1 - R(x))\frac{dx}{W}. \tag{3.42}$$

Including the first user in the batch, the average batch size is hence given by,

$$\overline{N} = 1 + \lambda W \tilde{p}. \tag{3.43}$$

Note that

$$\frac{d\overline{N}}{dW} = \lambda(1 - R(W)) \geq 0. \tag{3.44}$$

Therefore, $\overline{N}$ is a non-decreasing function in W, and attains its maximum when $R(W) = 1$. Hence, to maximize the per-batch profit $(\overline{N} - K)P$, W should be chosen equal to the maximum delay tolerance of the users, and the corresponding maximum per-batch profit is $(1 + \lambda\overline{U})P - C$. We thus note that maximizing per-batch profit makes no sense when the reneging function is exponential, since then $W = \overline{T} = \infty$, and thus $p_L = 1$. It makes no sense when the reneging function is linear since $\overline{T} = \infty$ and thus p_L is also 1. It only makes sense for the step function for which $W = U_0$ and the result is quite clear. A more interesting measure of profit is the profit rate, which examines the system profit over the infinite time horizon.

We now derive $\hat{\theta}$ with the specific users' reneging functions.

- Exponential function — The loss probability for the first user is given by,

$$p_1 = \begin{cases} 0, & \text{for } W < U_{min}, \\ 1 - e^{-(W-U_{min})/\tau}, & \text{otherwise.} \end{cases} \tag{3.45}$$

Hence, the average time between stream allocation $\overline{T}$ is given by (Eq. (3.41)),

$$\overline{T} = \begin{cases} \frac{1}{\lambda} + W, & \text{for } W < U_{min}, \\ \frac{1}{\lambda} e^{(W-U_{min})/\tau} + W, & \text{otherwise.} \end{cases} \tag{3.46}$$

The average batch size $\overline{N}$ is given by (Eq. (3.43)),

$$\overline{N} = \begin{cases} 1 + \lambda W, & \text{for } W < U_{min}, \\ 1 + \lambda U_{min} + \lambda\tau(1 - e^{-(W-U_{min})/\tau}), & \text{otherwise.} \end{cases} \tag{3.47}$$

Hence, $\overline{N}$ approaches asymptotically to $1 + \lambda U_{min} + \lambda\tau$.

Applying the above expressions for $\overline{N}$ and $\overline{T}$, we obtain the following expression for the normalized profit rate:

$$\hat{\theta} = \begin{cases} \lambda \left(1 - \frac{K}{1+\lambda W}\right), & \text{for } W < U_{min} \\ \frac{1+\lambda U_{min}+\lambda\tau(1-e^{-W/\tau})-K}{e^{(W-U_{min})/\tau}/\lambda+W}, & \text{otherwise.} \end{cases} \tag{3.48}$$

Clearly, for a given W, $\hat{\theta} \geq 0$ if K satisfies,

$$K \leq \begin{cases} 1 + \lambda W, & \text{for } W < U_{min} \\ 1 + \lambda U_{min} + \lambda\tau(1 - e^{-(W-U_{min})/\tau}), & \text{otherwise.} \end{cases} \tag{3.49}$$

Note that the service is not profitable if $K > 1 + \lambda(U_{min} + \tau)$; if $K < 1 + \lambda U_{min}$, the system is profitable whenever $W \geq W_0 = (K-1)/\lambda$; and if $1 + \lambda U_{min} \leq K \leq 1 + \lambda U_{min} + \lambda\tau$, the system is profitable when

$$W \geq W_0 = U_{min} - \tau \ln\left(1 - \frac{K - \lambda U_{min} - 1}{\lambda\tau}\right). \tag{3.50}$$

Note also that $\hat{\theta}$ approaches 0 as $W \to \infty$. Therefore, for any $K < 1 + \lambda(U_{min} + \tau)$, there is an optimal window size $W^* \in [W_0, \infty)$ which achieves the maximum profit rate (or profit over the infinite time horizon). The optimal window size is obtained by setting $d\hat{\theta}/dW^* = 0$, i.e.,

$$\begin{aligned} 1 + \lambda W^* e^{-(W^*-U_{min})/\tau} - \left(1 + \frac{e^{(W^*-U_{min})/\tau}}{\lambda\tau}\right) \times & \\ \left[1 + \lambda U_{min} - K + \lambda\tau(1 - e^{-(W^*-U_{min})/\tau})\right] &= 0. \end{aligned} \tag{3.51}$$

Clearly, $W^* \geq U_{min}$.

We now address user delay and derive its distribution. For $W < U_{min}$, $f_D(x)$ and $\overline{D}$ have already been derived in Sect. 3.3.2. We now consider the case $W \geq U_{min}$. Note that the first user in a batch always has delay W, and the requests arriving in the last U_{min} minutes of a batch have delay uniformly distributed between $[0, U_{min}]$, and the remaining ones in the batch would have delay $U_{min} < x < W$ with probability $(1 - R(x))$. Hence, the delay distribution $f_D(x)$ is,

$$f_D(x) = \frac{1}{\overline{N}} \times \begin{cases} (\lambda U_{min})\frac{1}{U_{min}}, & \text{for } 0 \leq x \leq U_{min}, \\ \lambda e^{-(x-U_{min})/\tau}, & \text{for } U_{min} < x < W, \\ \delta(x - W), & \text{otherwise.} \end{cases} \tag{3.52}$$

where $\delta(x)$ is the usual delta function with $\delta(x) = 0$ for $x \neq 0$ and $\int_0^\infty \delta(x) = 1$. The average delay $\overline{D}$ is hence given by $\int_0^W x f_D(x)dx$:

$$\begin{aligned} \overline{D} \;=\; & \frac{1}{1 + \lambda U_{min} + \lambda\tau(1 - e^{-(W-U_{min}/\tau)}} \times \\ & \left[\lambda\frac{U_{min}^2}{2} + \lambda\tau\left(U_{min} + \tau(1 - e^{-(W-U_{min})/\tau}) - We^{-(W-U_{min})/\tau}\right) + W\right]. \end{aligned} \tag{3.53}$$

- Linear function — In this case, if $W \leq U_{min}$, p_L is obviously 0 and $\hat{\theta}$ is $\lambda(1 - K/(1 + \lambda W))$. If $W > U_{max}$, since all users renege before a batch can be successfully formed, $p_L = 1$ and $\hat{\theta} = 0$.

  For $U_{min} < W \leq U_{max}$, the first user in a batch would renege before the batching window finishes with probability p_1 given by $p_1 = (W - U_{min})/\Delta$. Hence, the mean interval between consecutive channel allocations is,

$$\overline{T} = \frac{1}{1 - (W - U_{min})/\Delta} \cdot \frac{1}{\lambda} + W. \tag{3.54}$$

  The average batch size is given by,

$$\overline{N} \;\; = \;\; 1 + \lambda U_{min} + \frac{\lambda}{2\Delta}\left(\Delta^2 - (U_{max} - W)^2\right). \tag{3.55}$$

- Step function — Obviously, when $W > U_0$, since none of the users can stay until the end of a batching window, $p_L = 1$ and $\hat{\theta} = 0$. When $W \leq U_0$, no users would renege ($p_L = 0$) and hence $\overline{T}$ is given by,

$$\overline{T} = \frac{1}{\lambda} + W. \tag{3.56}$$

  Therefore, $\hat{\theta} = \lambda\left(1 - \frac{K}{1+\lambda W}\right)$.

3.4.3 Fixed-gating

In this section, we consider optimizing the batching window for fixed-gating with user reneging. As mentioned in Sect. 3.3, in this scheme, the movie is scheduled at regular intervals every W minutes (the batching window). If there are outstanding requests at the end of a batching window, the movie is shown; otherwise the show-time is dropped. In this scheme, movie show-time can be advertised to the users beforehand.

Let $\tilde{p}$ be the probability that a request would stay till the end of a batching window. By conditioning on the amount of time left at its arrival until the end of the window, we have,

$$\tilde{p} = \int_0^W (1 - R(u))\frac{du}{W}. \tag{3.57}$$

Hence, the probability of no request at the end of a batching window is given by $e^{-\lambda\tilde{p}W}$, and the average interval between consecutive streams is hence $\overline{T} = W/(1 - e^{-\lambda\tilde{p}W})$. The average batch size is given by $\overline{N} = \lambda\tilde{p}W/(1 - e^{-\lambda\tilde{p}W})$, from which we see that $\overline{N}$ increases (asymptotically to $\lambda\overline{U}/(1 - e^{-\lambda\overline{U}})$) as W increases.

For the exponential reneging funtion, we hence have,

$$\tilde{p} = \begin{cases} 1, & \text{for } W \leq U_{min}, \\ \frac{U_{min}}{W} + \frac{\tau}{W}\left(1 - e^{-(W - U_{min})/\tau}\right), & \text{otherwise.} \end{cases} \tag{3.58}$$

For the linear reneging function, we have,

$$\tilde{p} = \begin{cases} 1, & \text{for } W \leq U_{min}, \\ 1 - \frac{1}{2W\Delta}(W - U_{min})^2, & \text{for } U_{min} < W < U_{max}, \\ \frac{U_{min} + U_{max}}{2W}, & \text{for } W > U_{max}. \end{cases} \tag{3.59}$$

For the step reneging function, we obviously have,

$$\tilde{p} = \begin{cases} 1, & \text{for } W \leq U_0, \\ \frac{U_0}{W} & \text{otherwise.} \end{cases} \tag{3.60}$$

3.4.4 Batch-size based scheme

Here the system constinues to batch request until a certain number M of requests are collected. The system is aware of users reneging and does not count users that have reneged when counting requests. If M is low, then the profit is low due to too small a number of users served by a channel and too high the frequency of channel allocation; on the other hand, if M is high, then the profit is low due to users reneging. Therefore, we expect that there is an optimal value of M which maximizes the profit rate.

The analysis of such a scheme if users do not renege at all has been studied above. The analysis of the scheme when the reneging function is exponential with $U_{min} = 0$ is tractable; by following [75], the average time between consecutive channel assignment is,

$$\overline{T} = \frac{1}{\lambda}\sum_{i=0}^{M-1}\left[\frac{i!}{(\lambda\tau)^i}\sum_{n=0}^{i}\frac{(\lambda\tau)^n}{n!}\right], \tag{3.61}$$

and therefore, the normalized profit rate is $\hat{\theta} = (M - K)/\overline{T}$. The mathematical analysis for other reneging functions discussed in this chapter is more difficult; we resort to simulation for these cases.

3.4.5 Numerical examples and comparisons

Auto-gated scheduling

A. Exponential reneging function

We first study the influence of W on the profit rate $\hat{\theta}$, request loss rate p_L, batch size $\overline{N}$ and user delay, with $U_{min} = 0$. Then we study the influence of λ, K, τ and U_{min}.

We show in Fig. 3.53 $R(u)$ versus u given τ. The reneging probability first increases rather linearly with u, then in a more gentler manner, indicating that most users renege at low delay (indeed expected for the exponential distribution). Users are more willing to wait when τ increases. With $\tau = 15$ minutes users are not very patient (almost 30% of the users cannot wait beyond 5 minutes and more than 60% of the users cannot wait beyond 15 minutes), while with $\tau = 120$ minutes users are quite patient (almost 80% of the users can wait more than 30 minutes).

We consider in the following $\tau = 45$ minutes (moderately patient users) and $\lambda = 100$ req./hr. We show in Fig. 3.54 $\hat{\theta}$ as a function of W. For a given $K(= C/P)$, as W increases, $\hat{\theta}$ first increases sharply to reach a maximum, and then decreases slowly. There is an optimal window size W^* which maximizes $\hat{\theta}$. The sharp increase in $\hat{\theta}$ (especially when K is low) indicates that high profit can be achieved with low values of W. As K increases, more users have to be batched in order to amortize the channel cost and hence W^* also increases. Clearly, for $K \leq 1$, the system is always profitable no matter what W is, and for $K > 1$, there is a minimal window size beyond which the system is profitable. We note that the interesting case is typically $K > 1$, perhaps even greater than 10, indicating that the cost of a channel is only recovered if 10 paying users are served by it.

The user loss rate increases with W as shown in Fig. 3.55, in which we plot p_L versus

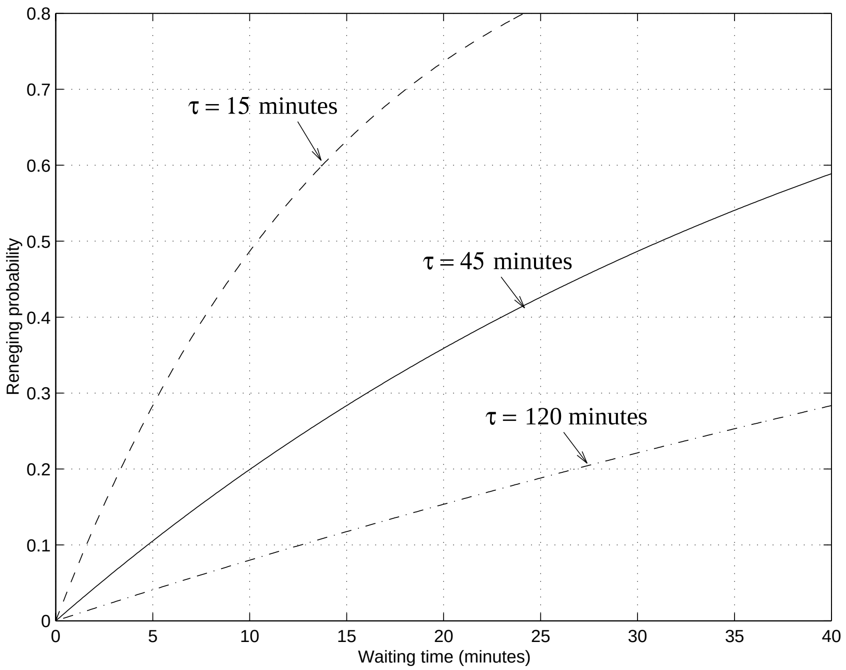


Figure 3.53: $R(u)$ vs. u for the exponential reneging function

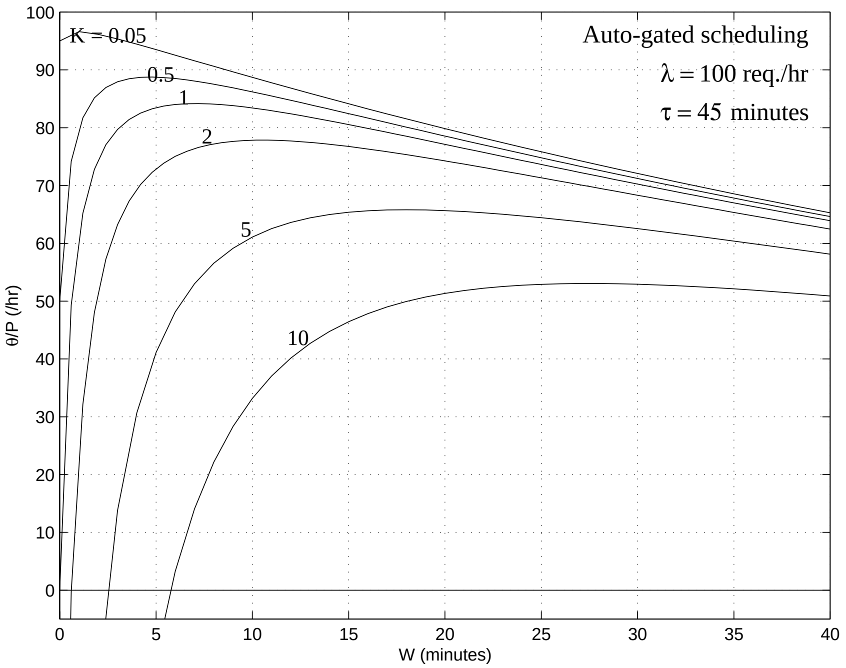


Figure 3.54: $\hat{\theta}$ vs. W, given K

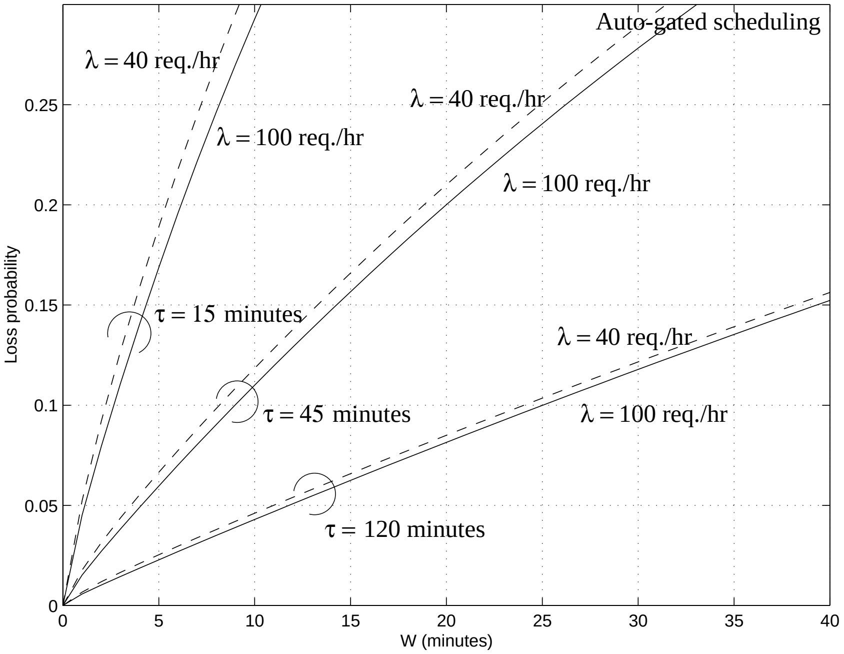


Figure 3.55: p_L versus W, given λ and τ

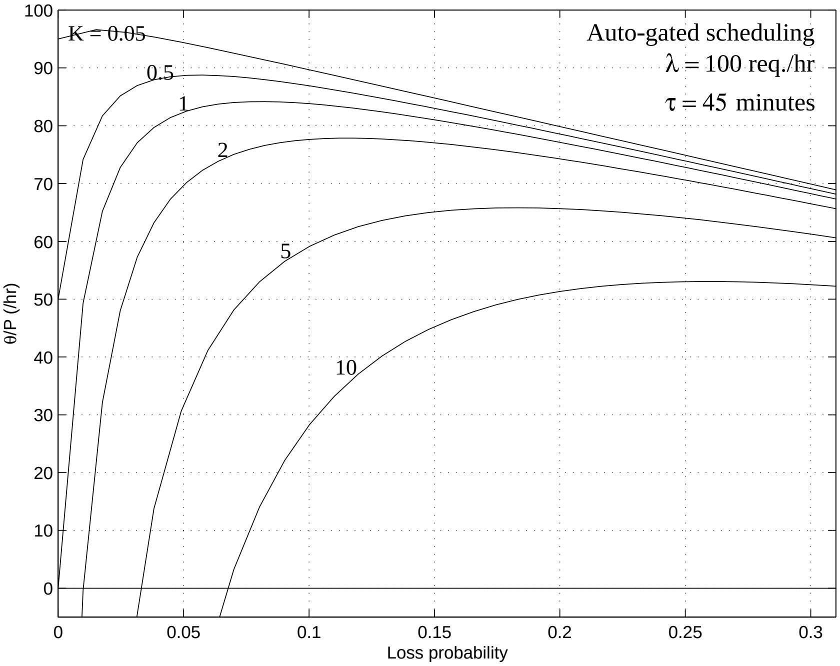


Figure 3.56: $\hat{\theta}$ vs. p_L given K

W for various value of λ and τ. We see that p_L is rather linear with W, and shows little dependence on λ. For a given p_L, the necessary window size does not depend much on λ. Thus, there is no incentive to use values for W above W^*. In fact, it may be preferable to choose values for W below W^* and thus achieve a lower profit rate than the maximum in order to keep p_L low. To illustrate this point, we plot in Fig. 3.56 $\hat{\theta}$ versus p_L (that is the trade-off between $\hat{\theta}$ and p_L) for $\tau = 45$ minutes and $\lambda = 100$ req./hr. Note that the shape of the curves is very similar to those of Fig. 3.54. Consider, for example, $K = 2$; we may choose $W = 4$ minutes ($\hat{\theta} = 70$/hr) instead of the optimum $W^* = 10$ minutes ($\hat{\theta}^* = 78$/hr), achieving a loss rate of 5% instead of 12%, respectively; with larger $K = 10$, we may choose $W = 12$ minutes ($\hat{\theta} = 40$/hr) instead of $W^* = 27$ minutes ($\hat{\theta}^* = 51$/hr), achieving a loss rate of about 12% instead of 26%, respectively.

In Figs. 3.57 and 3.58, we plot $\hat{\theta}$ versus W and $\hat{\theta}$ versus p_L, respectively, for a lower value of λ. We see that both curves follows the same trends as that of $\lambda = 100$ req./hr. However, since arrival rate is decreased, $\hat{\theta}$ also decreases. The "break-even" window size W_0 and the optimal window size W^* increases so that more users can be batched to amortize

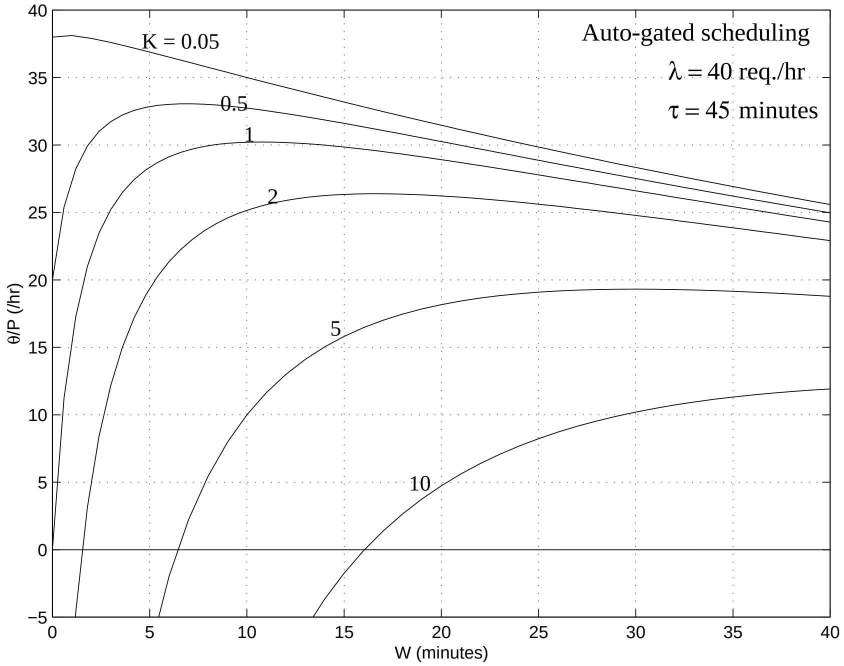


Figure 3.57: $\hat{\theta}$ versus W, given K for $\lambda = 40$ req./hr

the channel cost. As λ decreases, the loss rate for a given window size increases. At low λ, if we want to keep the loss rate below a certain value, the system may no longer be profitable. For example, for $K = 10$ and $p_L \leq 15\%$, the system is profitable when $\lambda = 100$ req./hr, but otherwise if $\lambda = 40$ req./hr. To remain profitable, a lower K should be used, by either using a cheaper channel or charging a higher pay-per-view.

In Fig. 3.59, we show the system throughput as a function of W, given λ and τ. The case with $W = 0$ is the pure-VOD case. As W increases, more users are lost and hence λ' monotonically decreases. As τ decreases, λ' decreases very rapidly. For a given τ, the decrease in throughput is more marked with high λ.

We show in Fig. 3.60 $\overline{N}$ versus W, given λ and τ. We see that $\overline{N}$ monotonically increases from 1, finally settling to a limiting value ($1 + \lambda\tau$, from Eq. (3.47)). The limiting value is achieved much faster with lower τ.

In Fig. 3.61, we plot $\overline{S}$ versus W. Also shown is the case in which users never renege ($\tau = \infty$). $\overline{S}$ does not depend much on τ given λ. Furthermore, there are only slight differences in $\overline{S}$ when λ is high enough (i.e., $\lambda > 40$ req./hr).

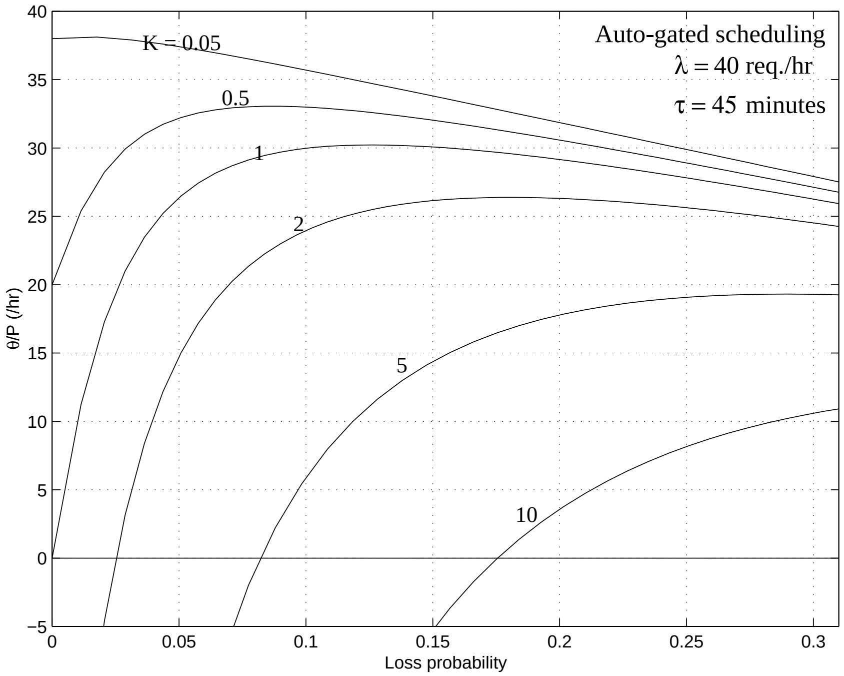


Figure 3.58: $\hat{\theta}$ versus p_L, given K for $\lambda = 40$ req./hr

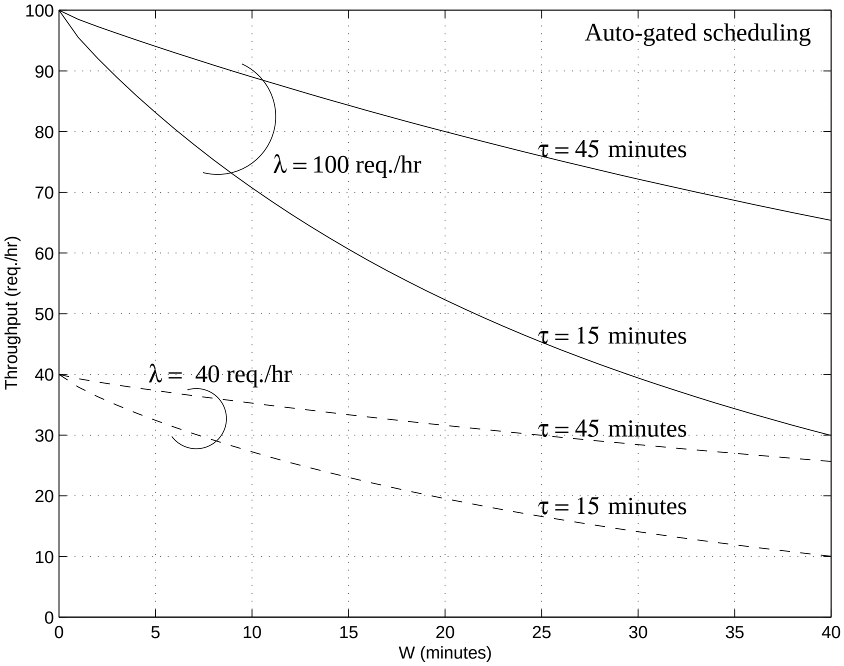


Figure 3.59: λ' vs. W, given λ and τ

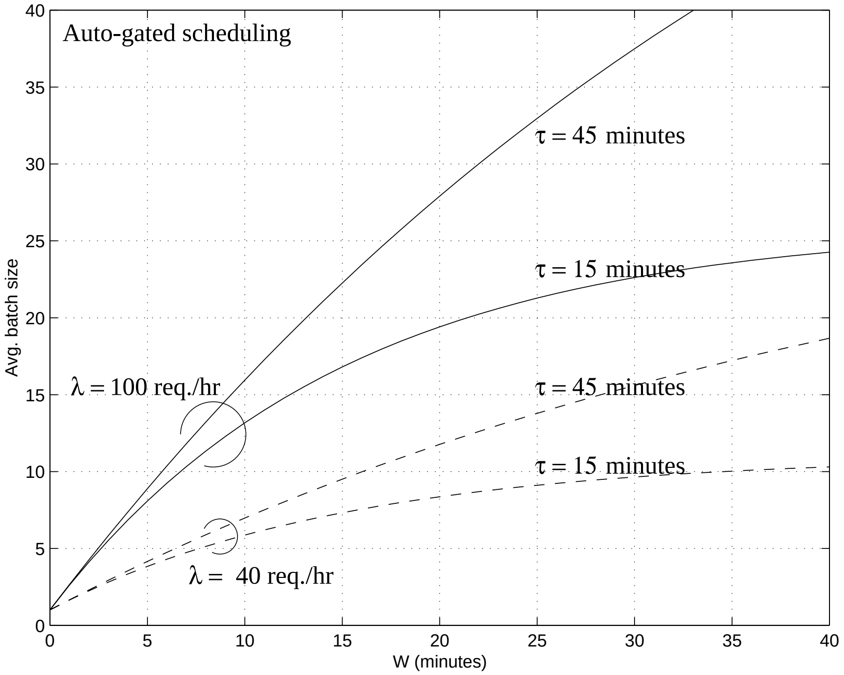


Figure 3.60: $\overline{N}$ vs. W, given λ and τ

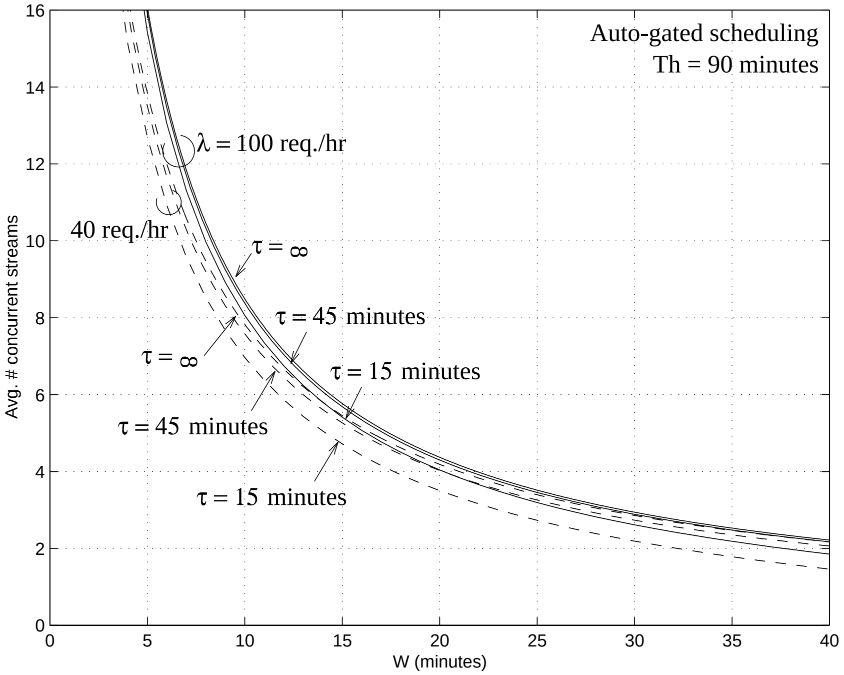


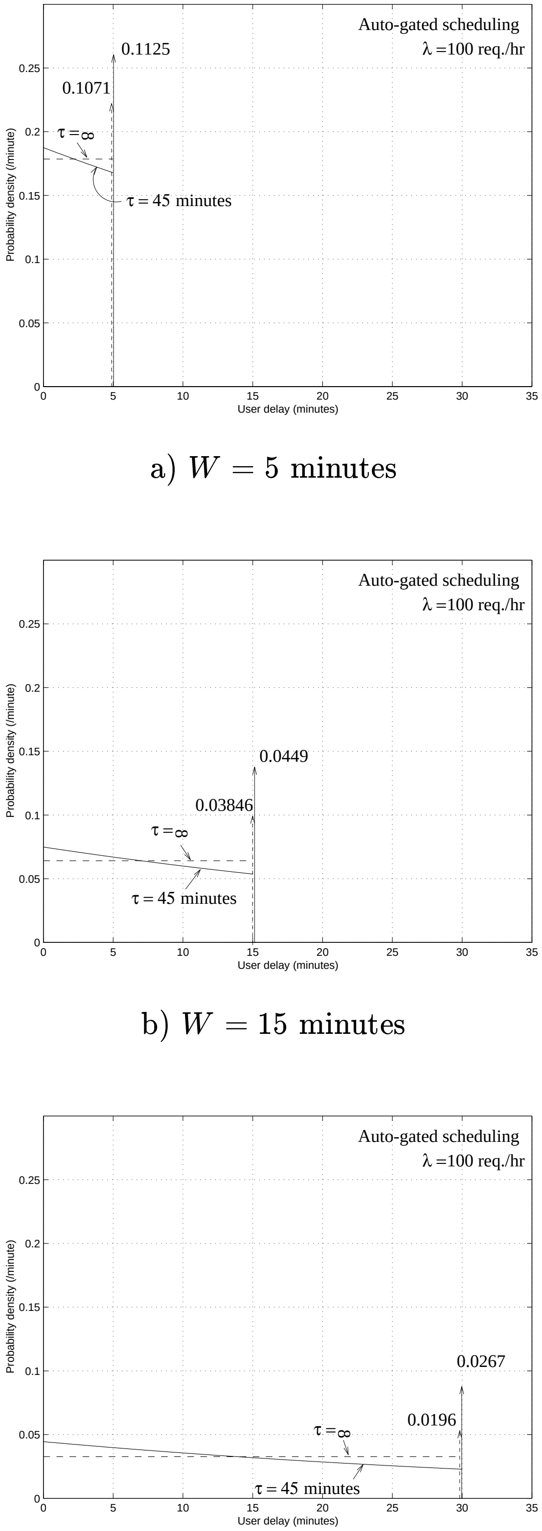
Figure 3.61: $\overline{S}$ versus W, given λ and τ

In Fig. 3.62, we show the delay distribution for $\lambda = 100$ req./hr with $W = 5$ minutes ($p_L \approx 5\%$), $W = 15$ minutes ($p_L \approx 15\%$) and $W = 30$ minutes ($p_L > 25\%$). Also shown is the case where users never renege ($\tau = \infty$). Due to user loss, the impulse at W minutes is higher for the case with $\tau = \infty$; the distribution is also skewed towards lower delays; however, overall, there is not much difference between the two distributions, even when p_L is high.

We show in Fig. 3.63 the average delay $\overline{D}$ versus W. There is not much difference even up to very high W (and hence high loss rate). Therefore for most of our interest, the user delay distribution can be approximated by the simpler case with $\tau = \infty$. Note that at low W, the average delay with finite τ is higher, since user reneging causes more users having delay W. At larger W, $\overline{D}$ is lower since the user distribution is biased towards lower delay. We show in Fig. 3.64 $\overline{D}$ with a lower λ ($\lambda = 40$ req./hr). Similar trend is observed.

We now study the influence of the load λ on W^*, W_0, p_L and $\hat{\theta}$. In Fig. 3.65, we show W^* and W_0 as a function of λ, given K. (The region below $\theta = 0$ ($W < W_0$) is unprofitable while the region above $\hat{\theta}^*$ ($W > W^*$) leads to high p_L with lower profit.) To achieve profitability, we must have $W_0 \leq W \leq W^*$. As λ increases, both W^* and W_0 decrease, first quite sharply and then more gradually. In Fig. 3.66, p_L is plotted against λ, with $W = W_0$ and $W = W^*$. We see that p_L decreases as λ increases, quite rapidly at first and then more gently. We also see that for high K, p_L resulting from maximizing the profit can be fairly high and hence undesirable. However, there is a wide range of W for which p_L can be kept low and the system is profitable. In Fig. 3.67, we show $\hat{\theta}^*$ as a function of λ given K. The maximum attainable profit increases somewhat linearly with λ. Furthermore, for a given K, there is a minimal value of λ for which the system is profitable (given by $1 + \lambda\tau = K$. See Eq. (3.49)).

We now examine the influence of K on W^* and W_0, on p_L and on $\hat{\theta}^*$ in Figs. 3.68, 3.69 and 3.70, given λ. As K increases, W^*, W_0 and the corresponding p_L increase almost linearly with K. Once again, we see that it is not always desirable to have $W = W^*$ since it leads to high p_L. As K increases, $\hat{\theta}^*$ decreases quite rapidly due to larger W^* and hence



a) $W = 5$ minutes

b) $W = 15$ minutes

c) $W = 30$ minutes

Figure 3.62: Delay distribution $f_D(d)$ for the auto-gated scheduling with exponential reneging function

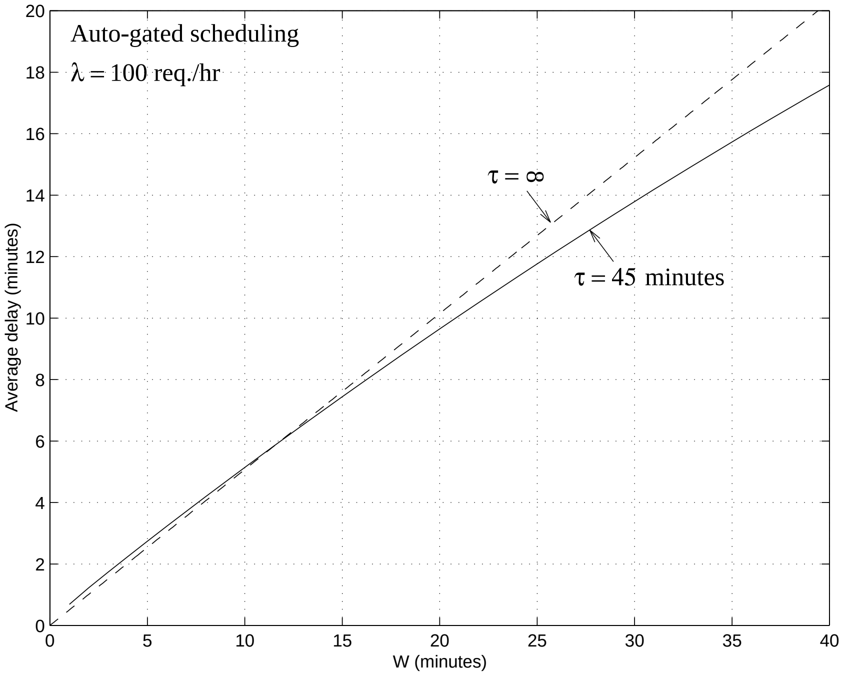


Figure 3.63: $\overline{D}$ versus W, given τ with $\lambda = 100$ req./hr

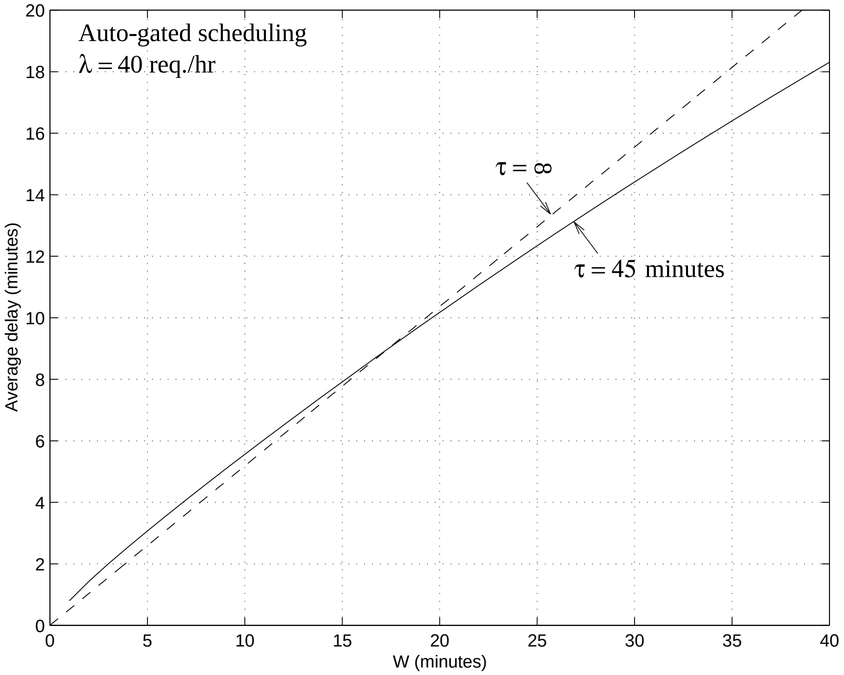


Figure 3.64: $\overline{D}$ versus W, given τ with $\lambda = 40$ req./hr

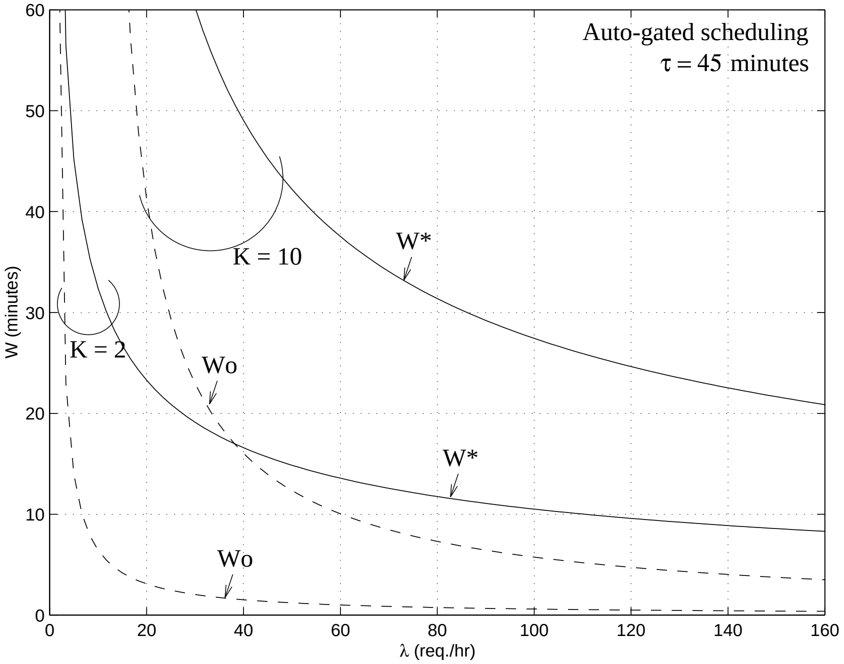


Figure 3.65: W (W^* and W_0) vs. λ, given K

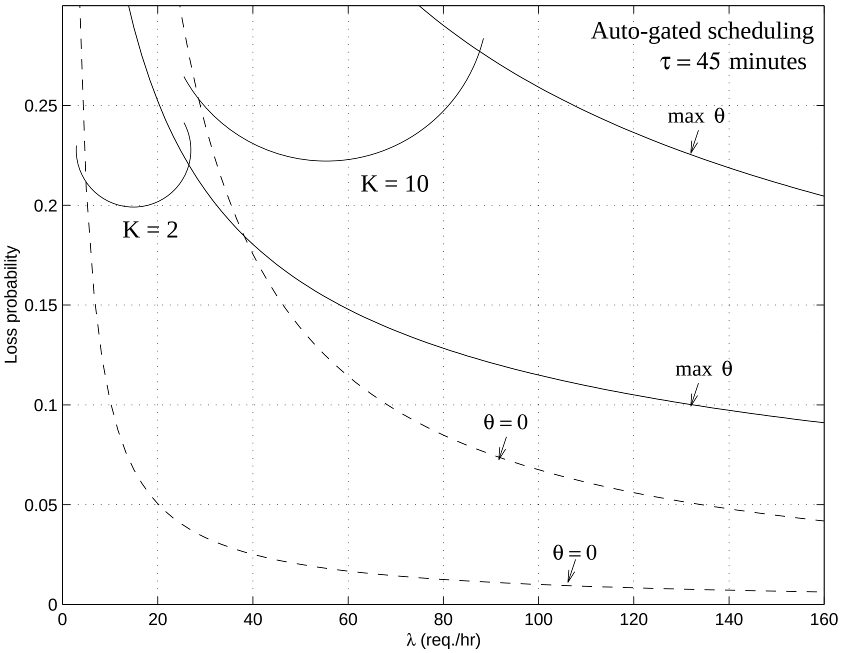


Figure 3.66: p_L vs. λ, given K

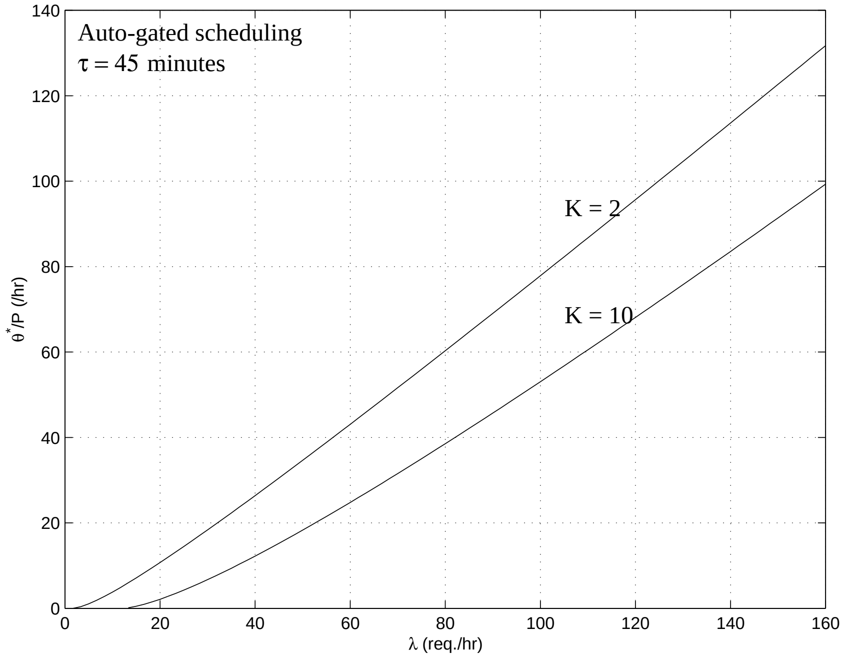


Figure 3.67: $\hat{\theta}^*$ versus λ, given K

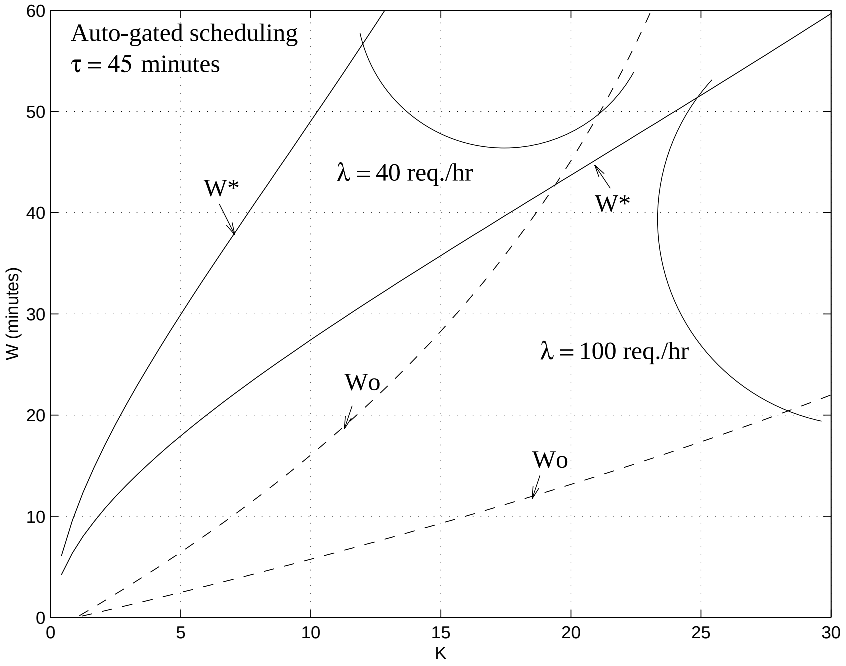


Figure 3.68: W^* and W_0 versus K, given λ

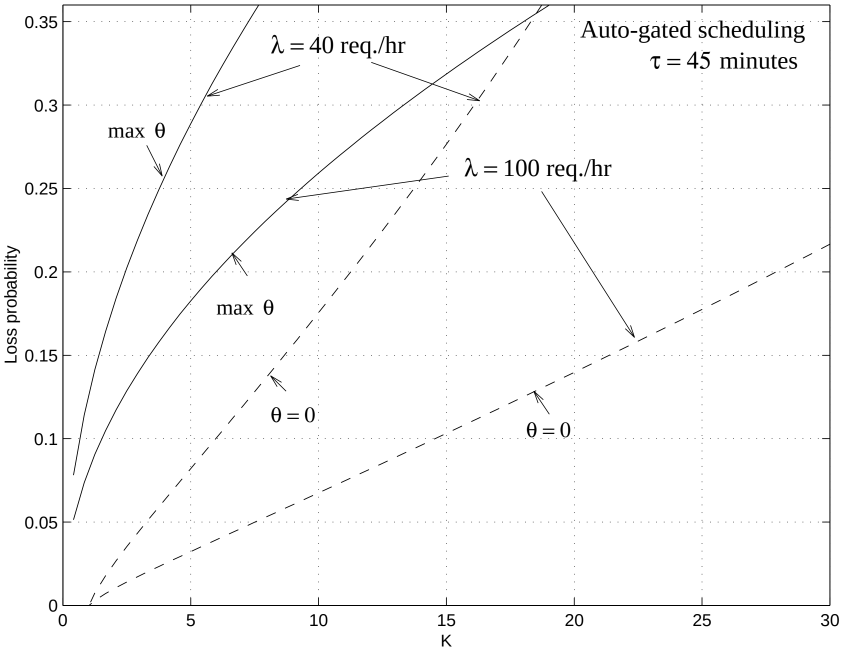


Figure 3.69: p_L versus K, given λ

higher user loss.

We now examine the effect of the user delay tolerance τ on the system profitability. We

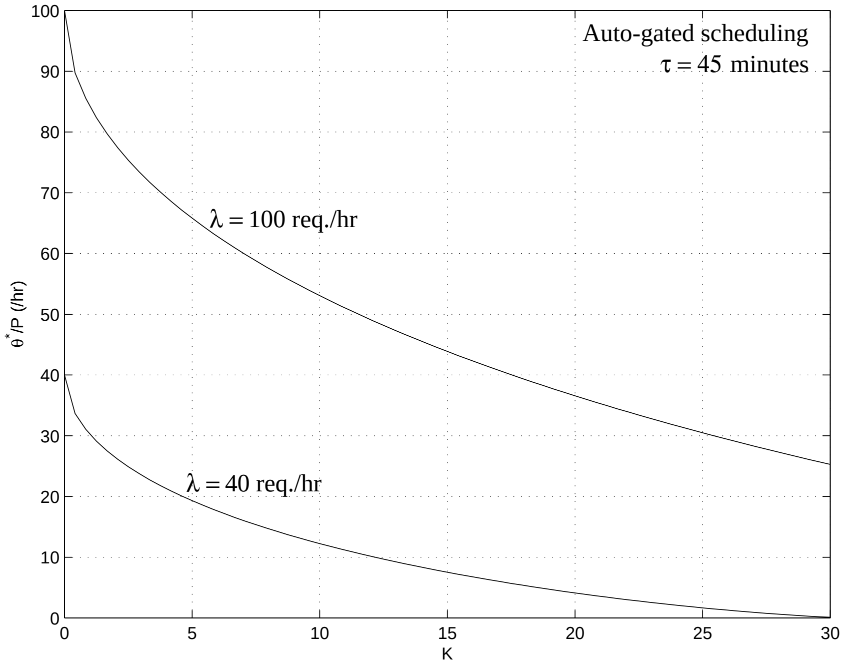


Figure 3.70: $\hat{\theta}^*$ versus K, given λ

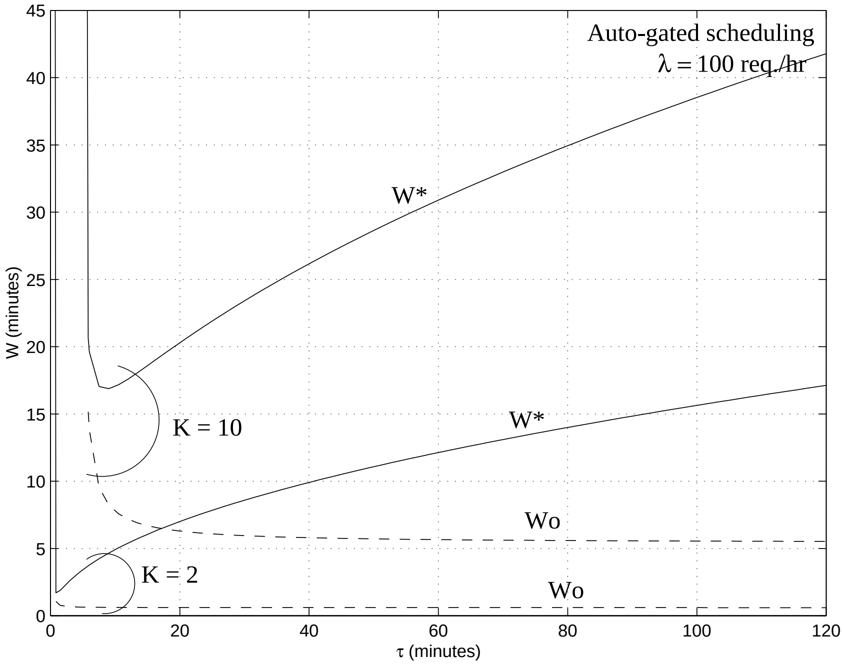


Figure 3.71: W (W^* and W_0) versus τ, given K

show in Fig. 3.71 W^* and W_0 versus τ. From Eq. (3.49), we note that given $K \geq 1+\lambda U_{min}$, τ has to be larger than a certain value ($= (K-1)/\lambda - U_{min}$) to ensure profit (otherwise users would be reneging too soon). This indeed is seen in the figure by the asymptotic behavior of W^* and W_0 as τ approaches the minimum tolerance from above. As τ increases from its lower limit, W^* first decreases very sharply from a high value to a low value, following which W^* increases with user delay tolerance. W_0, however, decreases with τ, first sharply, and then very slowly showing that W_0 is not sensitive to τ. In Fig. 3.72 we show p_L versus τ. As τ increases, p_L decreases, first sharply and then more gradually. To keep p_L low, τ should be at least larger than the "knee." We show in Fig. 3.73 that $\hat{\theta}^*$ increases with τ. Note the sharp knee when K is low, indicating that beyond the knee which there is no much increase in $\hat{\theta}^*$ as τ increases.

We now examine the effect of U_{min}. Figure 3.74 shows p_L versus W given U_{min}. We see that for a certain W, the higher U_{min} is, the lower p_L is. The family of curves are also remarkably parallel to each other.

We show in Fig. 3.75 how W^* and W_0 are affected by U_{min}, given K. We see that

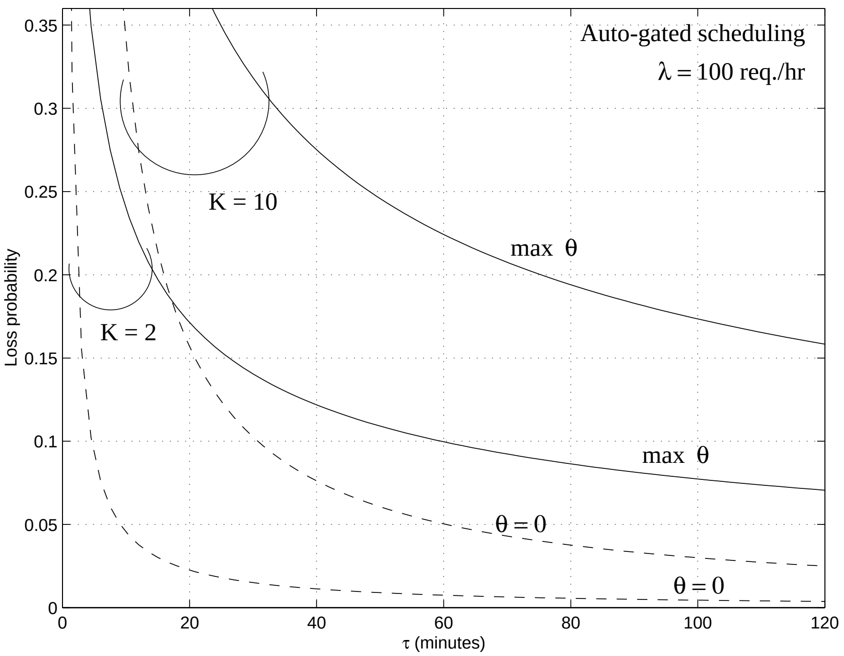


Figure 3.72: p_L versus τ, given K

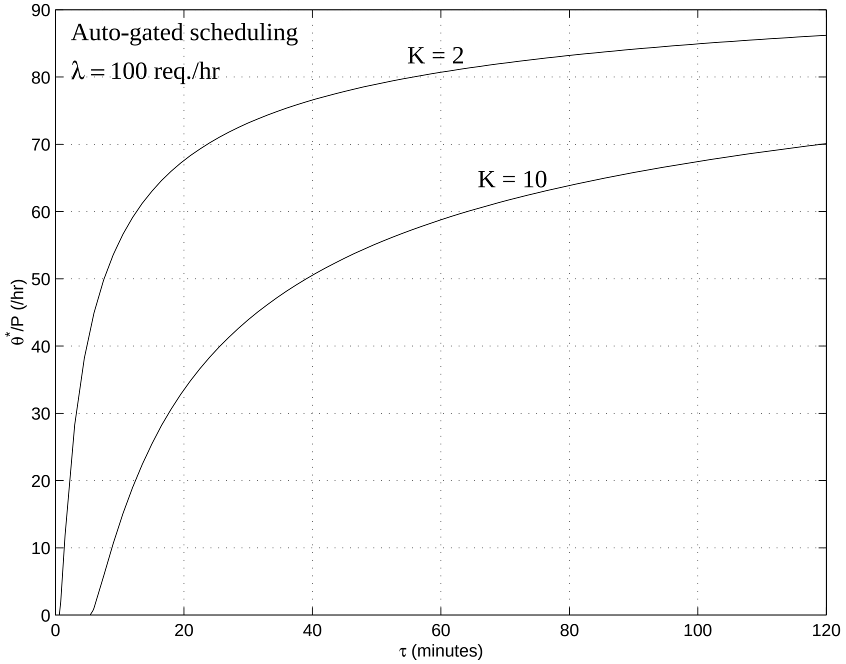


Figure 3.73: $\hat{\theta}^*$ versus τ, given K

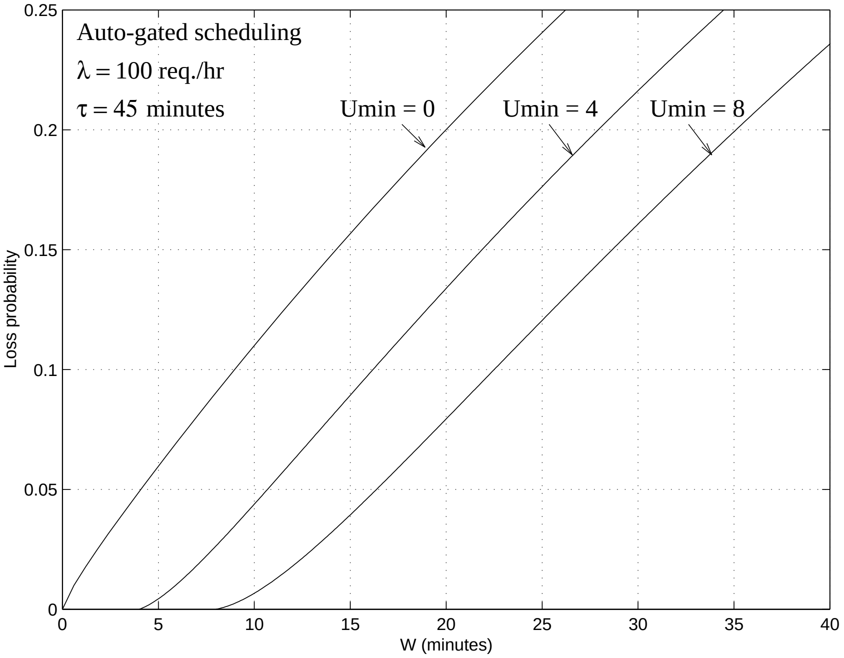


Figure 3.74: p_L versus W given U_{min}

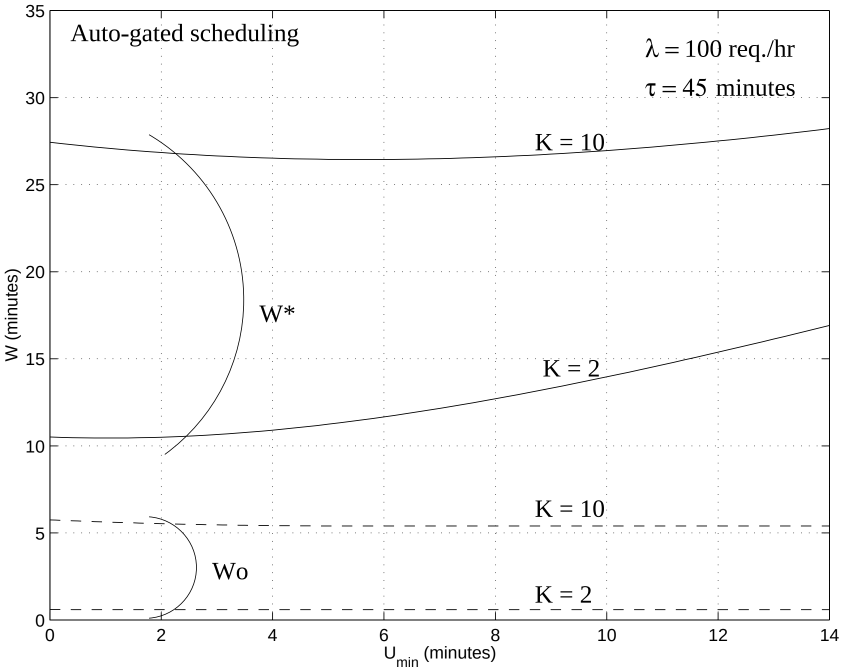


Figure 3.75: W^* and W_0 as a function of U_{min} given K

W^* does not change much especially for low U_{min}; and as U_{min} increases, W^* also slightly increase, keeping $W^* \geq U_{min}$. There is not much change in W_0 as U_{min} increases ($W_0 = (K-1)/\lambda$ as $U_{min} > (K-1)/\lambda$). In Fig. 3.76, we show p_L as a function of U_{min}. As U_{min} increases, p_L decreases quite significantly. We show in Fig. 3.77 $\hat{\theta}^*$ versus U_{min}. Clearly, the profit rate increases with U_{min} as p_L decreases.

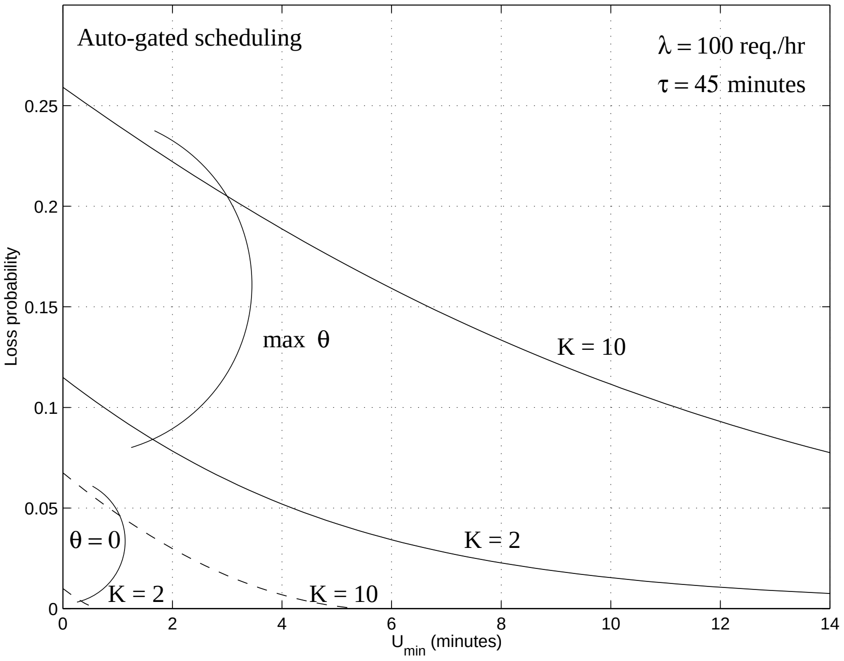


Figure 3.76: p_L as a function of U_{min} given K

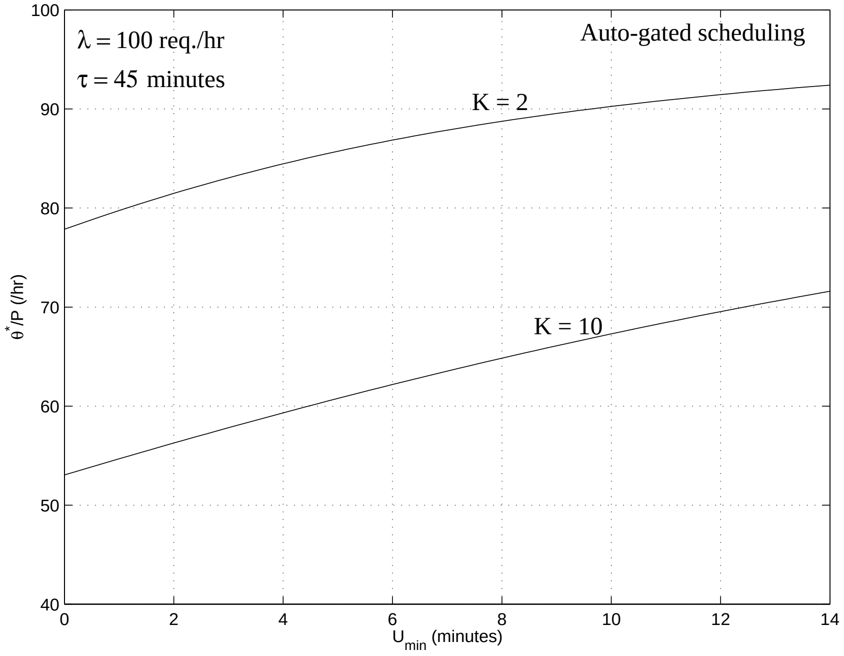


Figure 3.77: $\hat{\theta}^*$ as a function of U_{min} given K

In Figs. 3.78, 3.79 and 3.80, we show the corresponding graphs with a lower λ (= 40 req./hr). Though the profitability is lower, similar trends are observed.

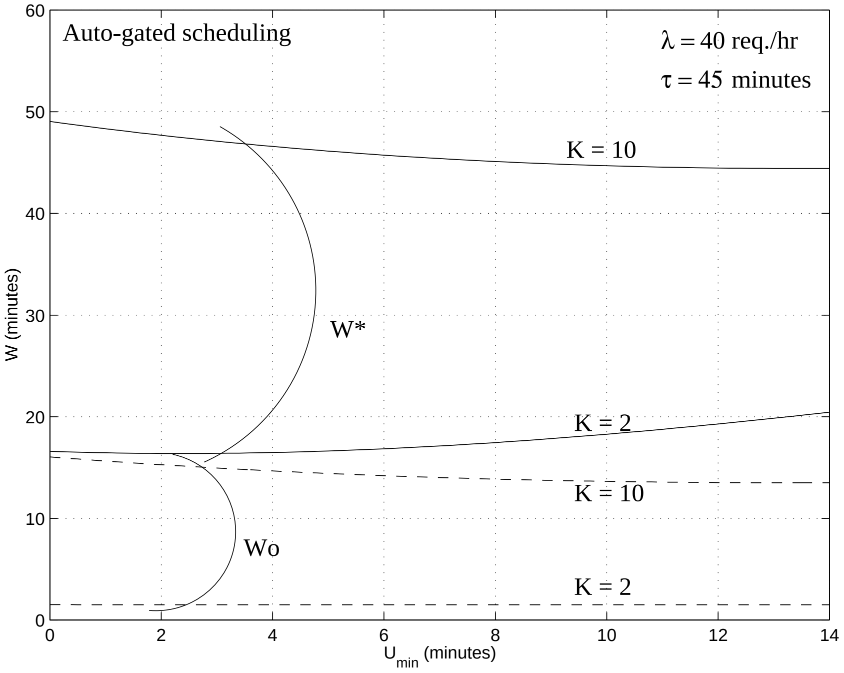


Figure 3.78: W^* and W_0 as a function of U_{min} given K

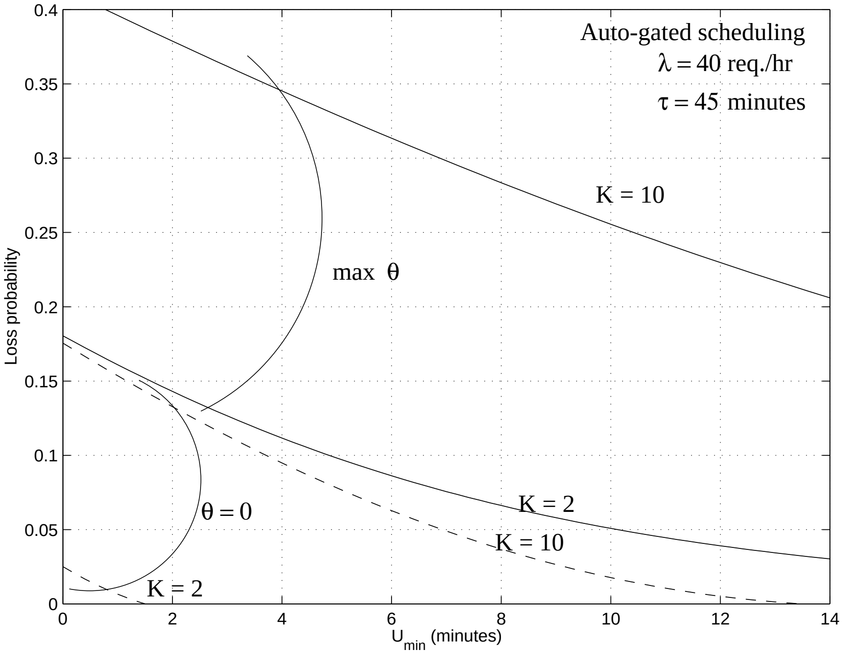


Figure 3.79: p_L as a function of U_{min} given K

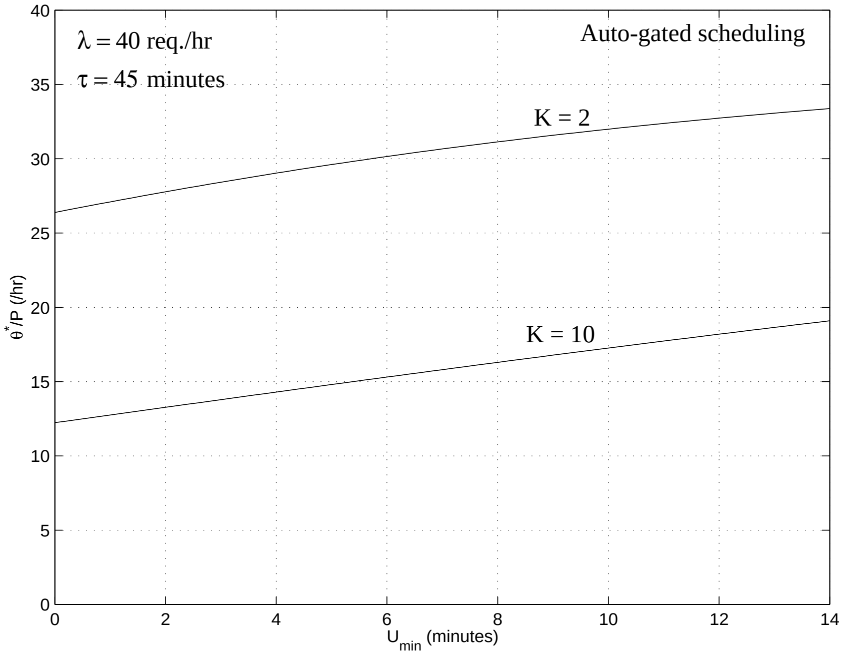


Figure 3.80: $\hat{\theta}^*$ as a function of U_{min} given K

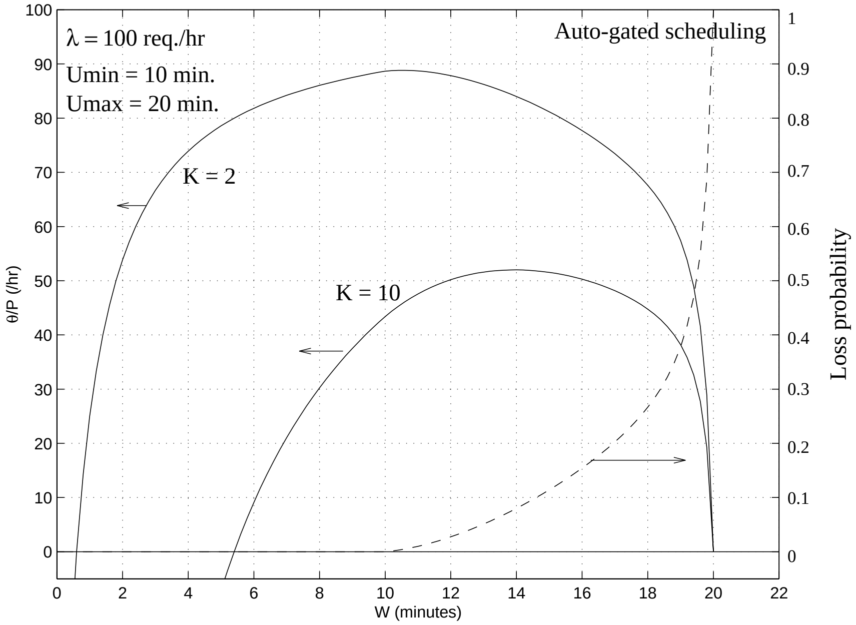


Figure 3.81: $\hat{\theta}$ and p_L versus W, given K

B. Linear reneging function

We now consider the linear reneging function. We show in Fig. 3.81 $\hat{\theta}$ and p_L as a function of W, with $U_{min} = 10$ minutes and $U_{max} = 20$ minutes. When $W \leq U_{min}$, $\hat{\theta}$ increases monotonically. When W is further increased, $\hat{\theta}$ first increases (as a larger W potentially batches more users) and then decreases rapidly (due to user reneging). For low K, W^* is extremely close to U_{min}, while for larger K, W^* is somewhat larger than U_{min}. In all cases, $\hat{\theta}^*$ is achieved with low p_L ($\leq 10\%$).

As a comparison with the exponential reneging function, we plot in Fig. 3.82 $\hat{\theta}$ versus W, with $\overline{U}$ chosen to be the same as in Fig. 3.54 ($U_{min} = 0$ minutes and $U_{max} = 90$ minutes). Also shown in dashed line is the corresponding p_L. From the remarkable resemblance between the two figures, we see that whether it is the linear or the exponential reneging function does not affect $\hat{\theta}^*$ and W^* much, so long as they have the same mean. Since the exponential distribution is skewed towards lower delay values than the uniform distribution, W^* and the corresponding $\hat{\theta}^*$ are slightly lower.

C. Step reneging function

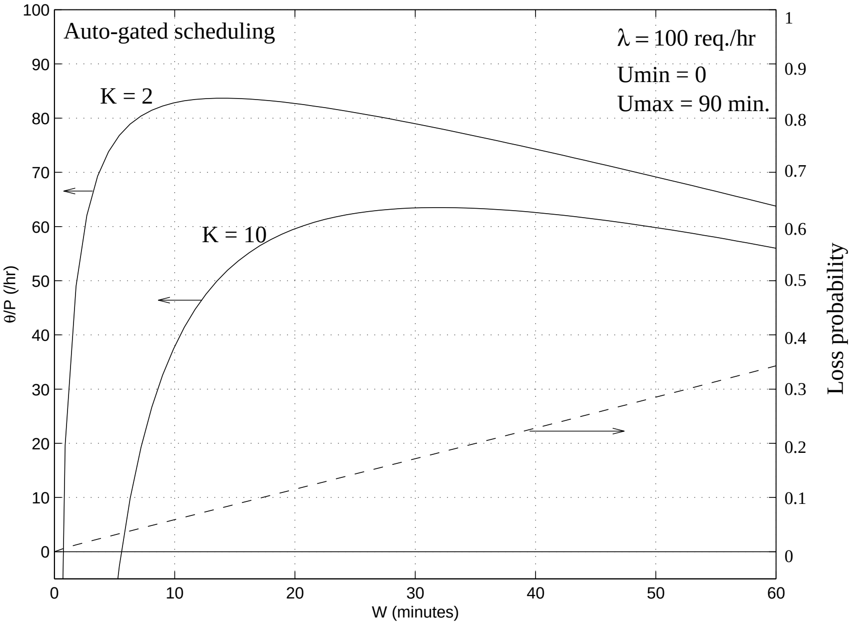


Figure 3.82: θ versus W with the linear user reneging function

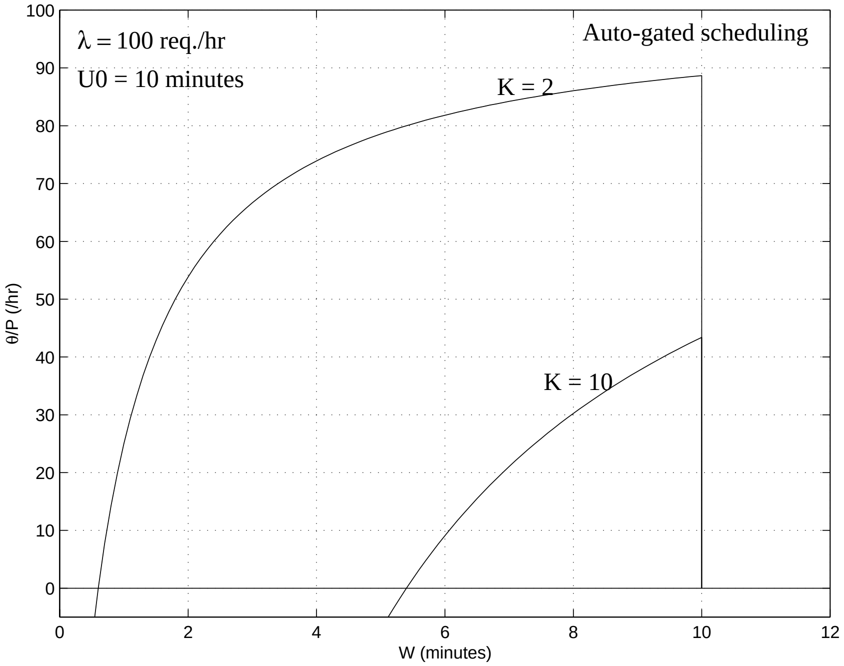


Figure 3.83: $\hat{\theta}$ versus W given K with the step reneging function

We show in the following the profitability with step reneging function. In Fig. 3.83, we plot $\hat{\theta}$ versus W given K, with $U_0 = 10$ minutes. Obviously, $\hat{\theta}$ increases monotonically when $W \leq U_0$ (with $p_L = 0$). As $W > U_0$, no batch can be successfully formed and hence $\hat{\theta} = 0$. For low K, the "knee" shows that most of the profit can be gained at a lower W than U_0.

We show in Fig. 3.85 how $\hat{\theta}^*$ varies with U_0, given K. For a given λ, there is a minimum U_0 for which the system is profitable. As U_0 increases, $\hat{\theta}^*$ gradually increases, approaching to the value λ. We also see that, for low K, most of the profit is attained at low U_0 (4–6 minutes).

We finally show in Fig. 3.85 $\hat{\theta}^*$ as a function of λ, given K. The profit is very close to being linear in λ, bounded from below by $\lambda - K/W$. Given U_0 and K, there is a minimum arrival rate ($= (K-1)/U_0$) above which profit is possible.

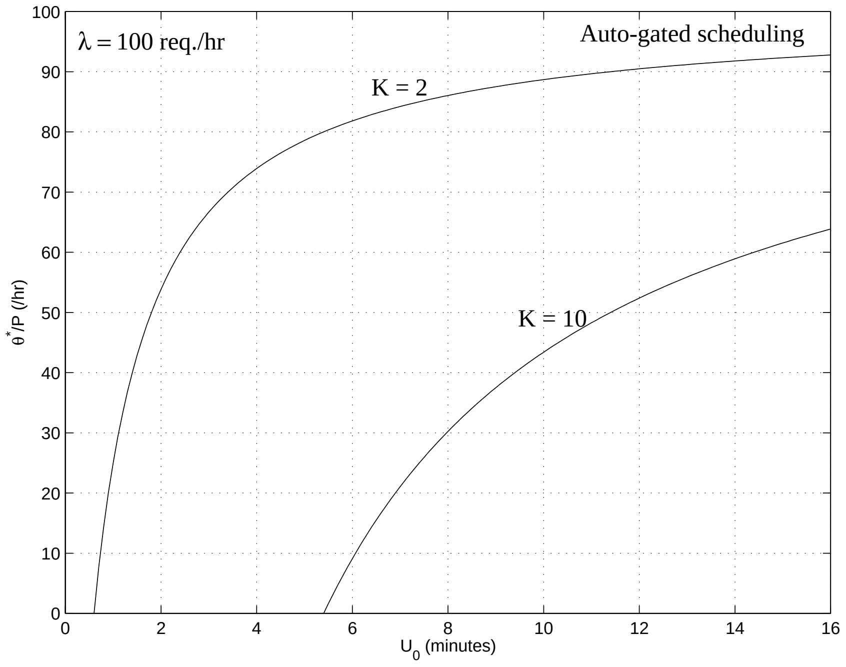


Figure 3.84: $\hat{\theta}^*$ versus U_0 given K

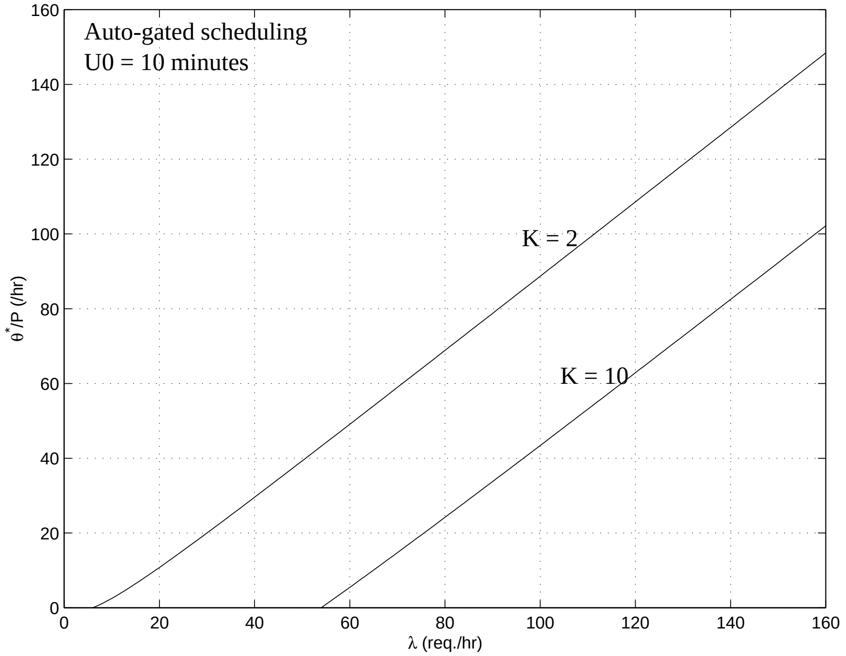


Figure 3.85: $\hat{\theta}^*$ versus λ given K

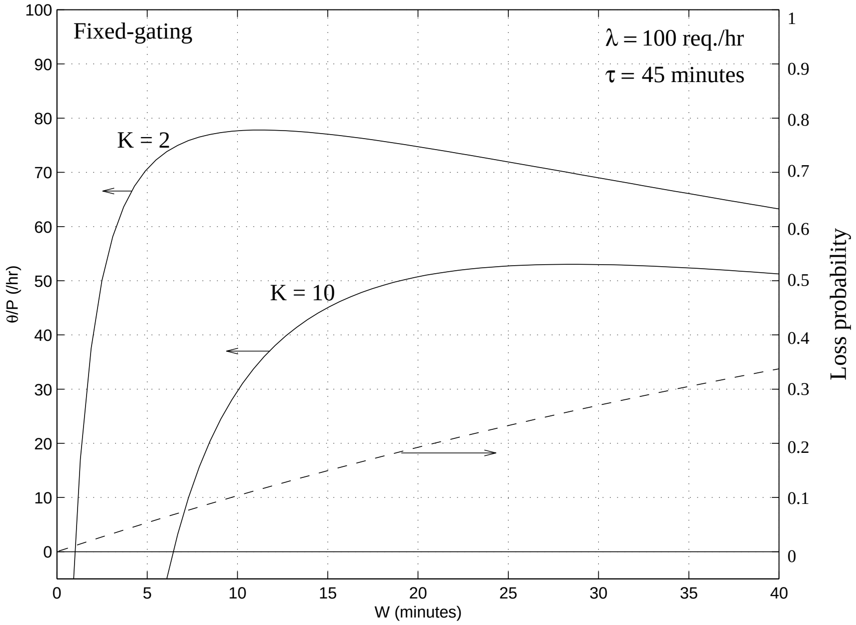


Figure 3.86: $\hat{\theta}$ vs. W for the fixed-gating, with the exponential reneging function

Fixed-gating

We show in Fig. 3.86 $\hat{\theta}$ versus W with the exponential reneging function ($U_{min} = 0$ and $\tau = 45$ minutes) and $\lambda = 100$ req./hr. Note the remarkable similarity in $\hat{\theta}$ between the scheme and that of the auto-gating (Fig. 3.54). Fixed-gating, however, comes with another advantage: it offers users certain waiting time.

We show in Fig. 3.87 the case with the linear reneging function. Comparing with the auto-gated case (Fig. 3.81), we again see the remarkable resemblance in $\hat{\theta}^*$ and W^*. However, fixed-gating is more forgiving when the window size is larger than the maximum delay tolerance of the users ($W > U_{max}$): while in the auto-gating case, no users can be served ($p_L = 1$) and $\hat{\theta}$ drops to zero, fixed-gating has a relatively gentler roll-off in performance

We show in Fig. 3.88 the case with the step user reneging function. By comparing with that of auto-gating (Fig. 3.83), we see again the similarity in W^* and $\hat{\theta}^*$ (though auto-gating, due to its higher average batch-size, has slightly higher $\hat{\theta}^*$, as apparent when K is high), and the slower decrease in $\hat{\theta}$ when $W > U_0$.

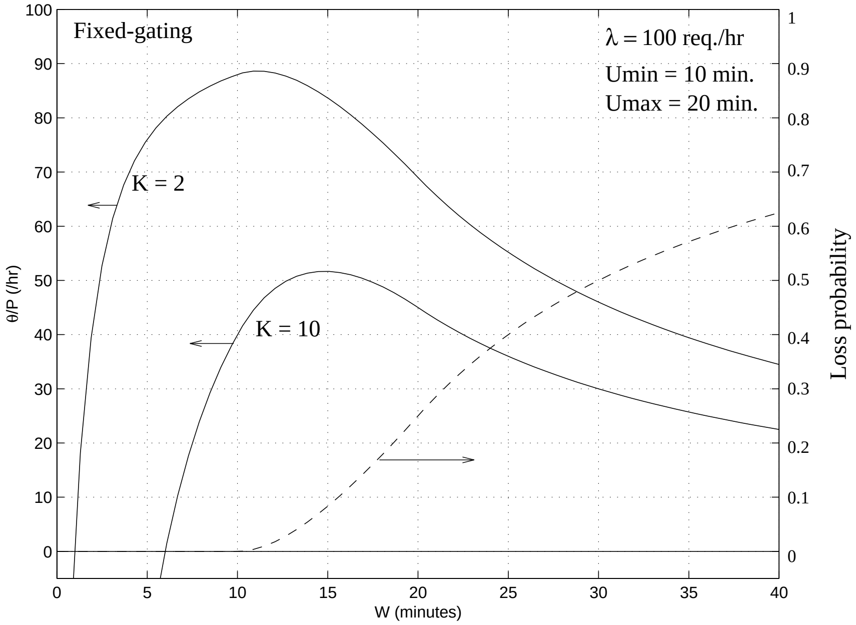


Figure 3.87: $\hat{\theta}$ vs. W for the fixed-gating, with the linear reneging function

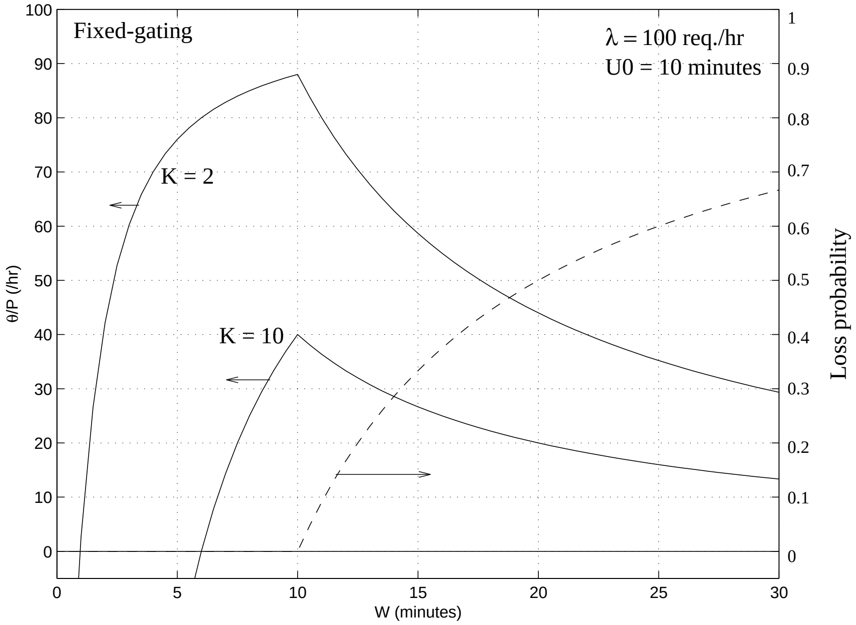


Figure 3.88: $\hat{\theta}$ vs. W for the fixed-gating, with the step reneging function

We see that there is not much diffierence between fixed-gating and auto-gated scheduling in terms of the maximum profit achievable. However, in auto-gated scheduling, the window size should not be larger than the maximum delay tolerance of the users since system performance drops off very quickly beyond that; while in the fixed-gating, the performance rolls off more gently.

Batch-size based scheme

In this section, we study the profitability of the batch-size based scheme.

First we consider the exponential reneging function, with $U_{min} = 0$. We show in Fig. 3.89 $\hat{\theta}$ as a function of batch-size M, given K ($\tau = 45$ minutes and $\lambda = 100$ req./hr). Also shown in the dashed line is the user loss rate p_L. We see that once $M > K$, the system becomes profitable, and there is an optimal batch-size M^* achieving the highest profit rate $\hat{\theta}^*$. Note the remarkable resemblance in $\hat{\theta}$ and p_L at the optimal operating point between this figure and Fig. 3.54 (the auto-gated case). We also find that M^* is very close to the $\overline{N}$ obtained with W^* in the auto-gated scheduling! We show in Fig. 3.90 the case with lower λ, in which the same resemblance is observed. We found that so long as both schemes are profitable, the optimal operating points are similar.

There is a slight difference between the schemes, however. While the auto-gated scheduling does not discriminate between profitable and unprofitable batches and serves the requests once the batching window finishes, the batch-size based scheme is able to selectively pick those arrival patterns which lead to profit (with $M \geq K$). Therefore, when λ is low (i.e., $\lambda < (K-1)/\tau$), auto-gated scheduling would not be profitable but the batch-size based scheme can be profitable, albeit with high user loss rate. We illustrate this point in Fig. 3.91, in which we show $\hat{\theta}$ and p_L as a function of M, with $\lambda = 10$ req./hr and $\tau = 45$ minutes. The auto-gated scheduling would not be profitable in this case (Fig. 3.67); the batch-size based scheme maintains profitability, but with a high loss probability ($> 50\%$!).

We plot in Fig. 3.92 λ' versus M, given λ. The case for $M = 1$ is the pure-VOD case. Due to user reneging, λ' monotonically decreases with M. The rather linear decrease in λ'

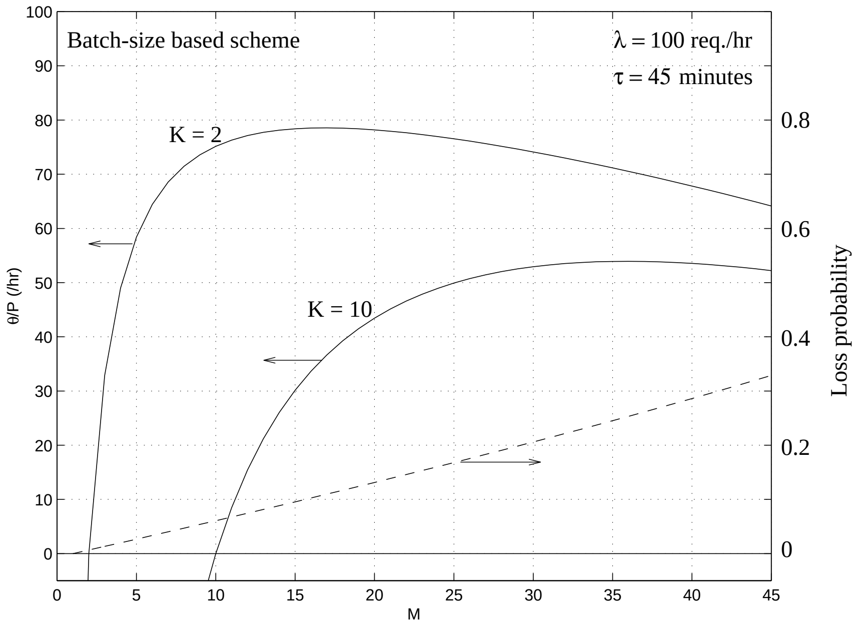


Figure 3.89: $\hat{\theta}$ vs. M given K, with $\lambda = 100$ req./hr

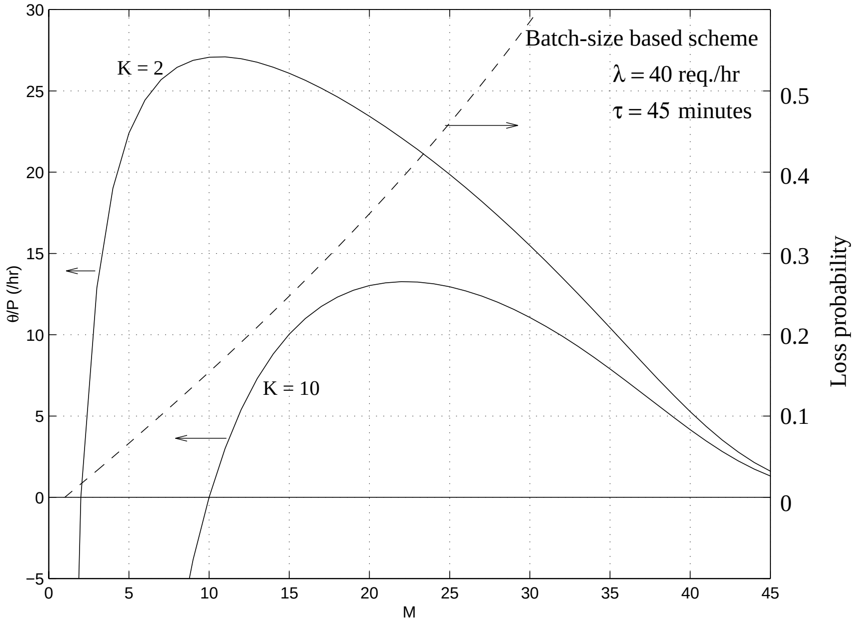


Figure 3.90: $\hat{\theta}$ vs. M given K, with $\lambda = 40$ req./hr

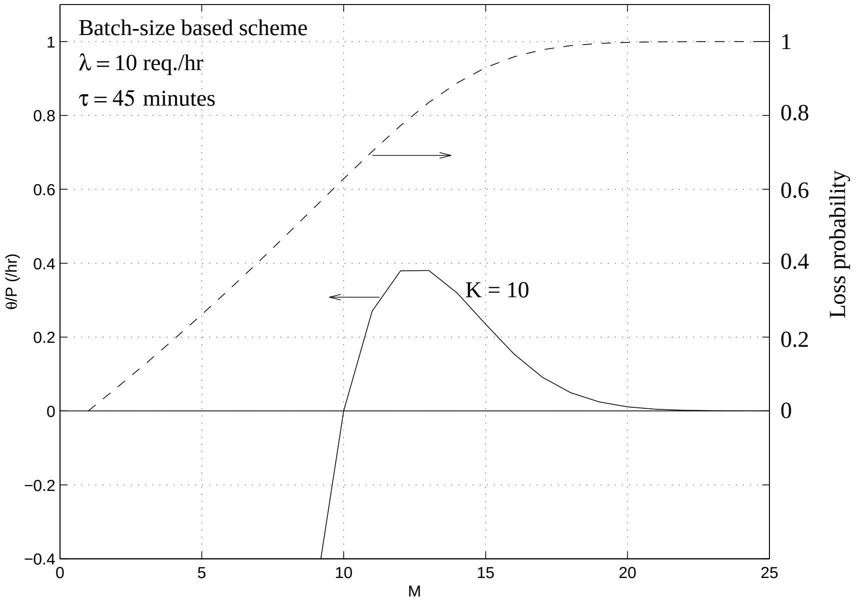


Figure 3.91: $\hat{\theta}$ vs. M, with $K = 10$

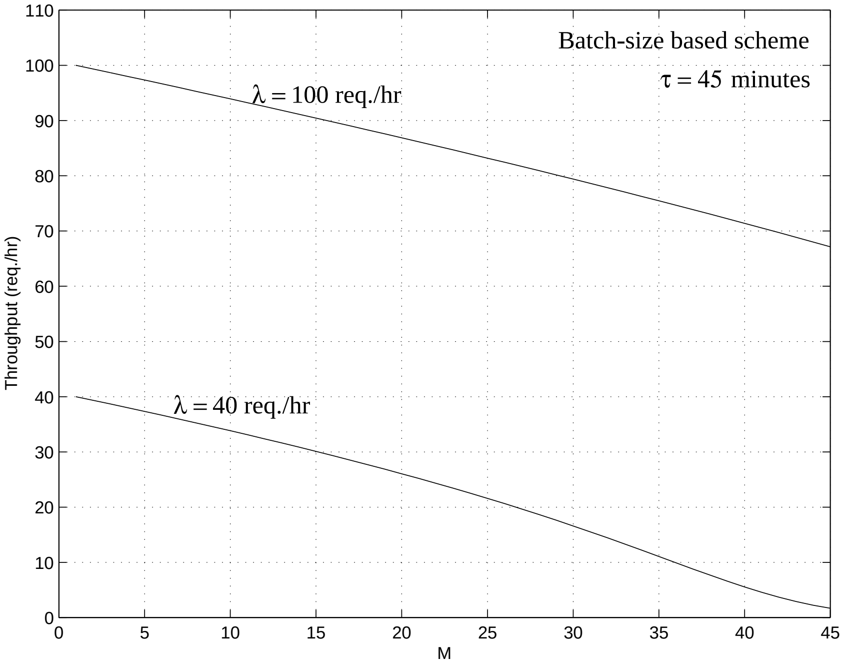


Figure 3.92: λ' versus M given λ

indicates that p_L is quite linear in M, as readily seen from Figs. 3.89 and 3.89.

We show in Fig. 3.93 $\overline{S}$ versus M, with $\tau = 45$ minutes and $\tau = \infty$. As M increases, more users can be batched and hence $\overline{S}$ decreases. We also see that most of the decrease in $\overline{S}$ occurs at low M. As τ decreases, users are more likely to renege and hence $\overline{S}$ decreases. However, there is no significant difference in $\overline{S}$ between $\tau = 45$ minutes and the case in which users do not renege.

We now study linear user reneging function. In Fig. 3.94, we plot $\hat{\theta}$ versus M with $U_{min} = 10$ minutes and $U_{max} = 20$ minutes. We again see the similarity in the optimal operating point between this plot and Fig. 3.81. We plot in Fig. 3.95 the case with $U_{min} = 0$ and $U_{max} = 90$ minutes, once again showing its similarity with auto-gated scheduling (Figs. 3.54 and 3.82). Therefore, as long as both systems are profitable, the optimal profit rates and the corresponding p_L and $\overline{N}$ in the batch-size based scheme and the auto-gated scheduling are similar.

We show in Figs. 3.96 and 3.97 $\hat{\theta}$ versus M with step reneging function, with $U_0 = 5$ minutes and 10 minutes, respectively. $\hat{\theta}^*$ are similar in both schemes. However, it is

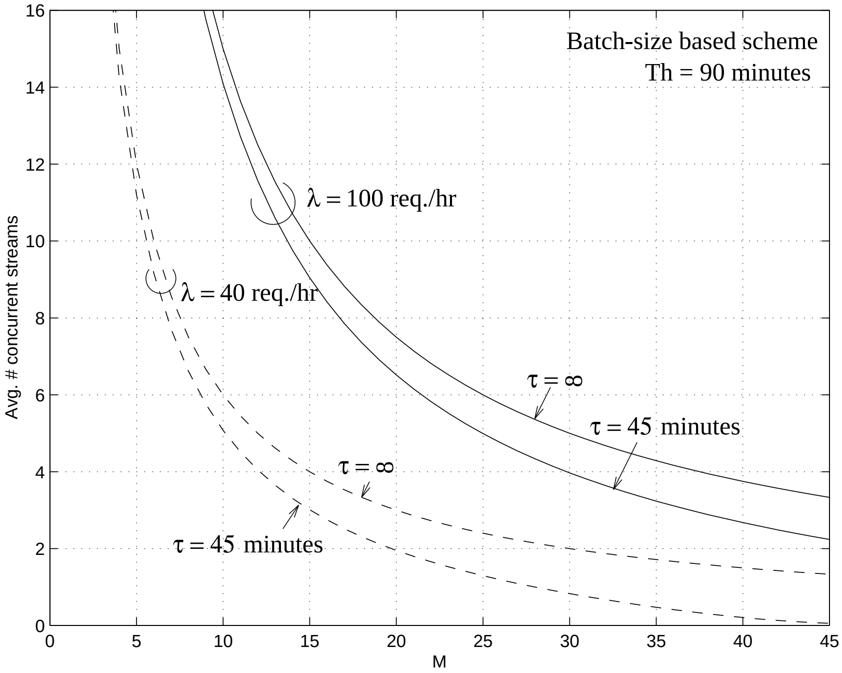


Figure 3.93: $\overline{S}$ versus M given λ

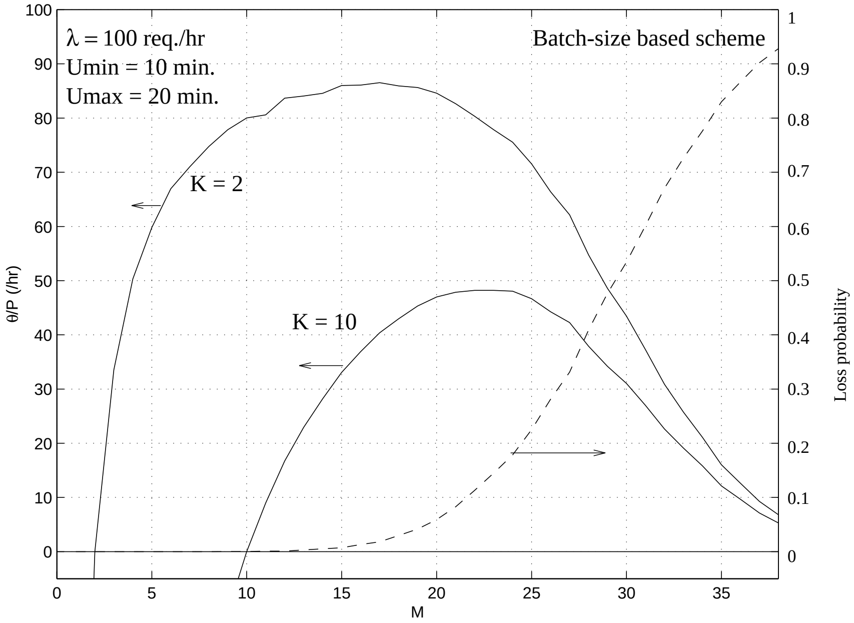


Figure 3.94: $\hat{\theta}$ versus M with linear reneging function

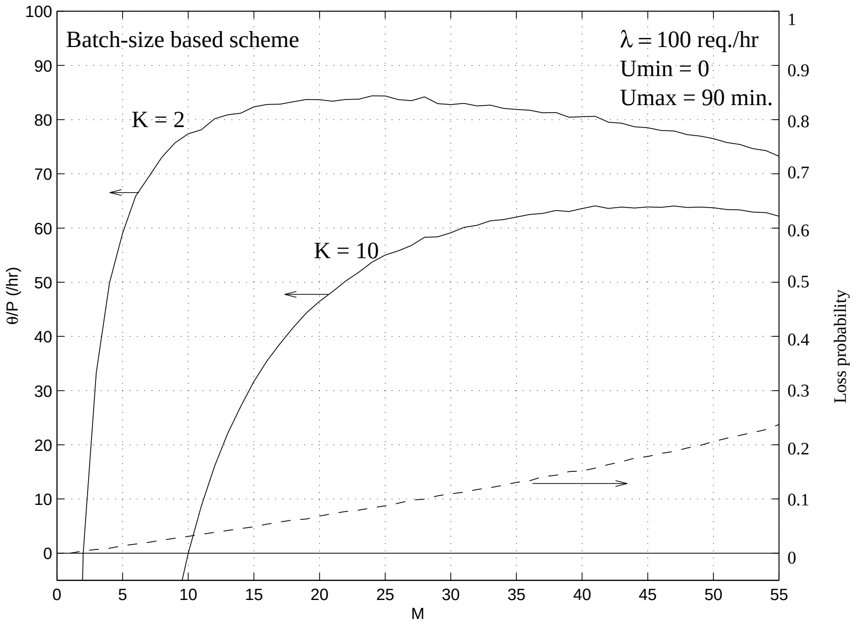


Figure 3.95: $\hat{\theta}$ versus M with linear reneging function

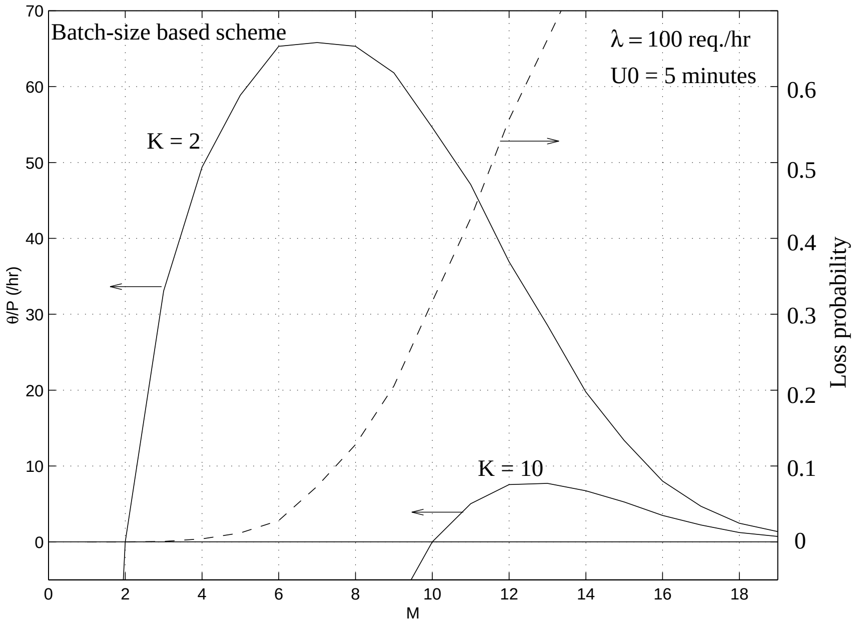


Figure 3.96: $\hat{\theta}$ and p_L versus M given K

worthwhile to note that, while p_L can be zero at $\hat{\theta}^*$ in the auto-gated scheduling (Fig. 3.83), there is some user loss at the optimum in the batch-size based case. Note that for $K = 10$ and $U_0 = 5$ minutes, auto-gated scheduling would not be profitable while the batch-size based scheme is profitable, but with a high p_L.

Loss-dependent arrival rate

So far, we have considered that λ is independent of p_L. In reality, since unsatisfied users (i.e., those reneged users) may never visit the system again, λ can be considered a function of p_L. In this section, we address how the window size or batch size can be specified to maximize per-batch profit or profit rate under such condition.

We first study maximizing per-batch profit. Recall that if λ were independent of p_L, we would always use a window size as large as the maximum delay tolerance of the users in the auto-gated scheduling, or an arbitrary large batch size in the batch-size based scheme. In Fig. 3.98, we show how the optimal solution would be changed if λ is dependent on p_L. First we show in solid lines the contours for constant batch size. For a particular λ,

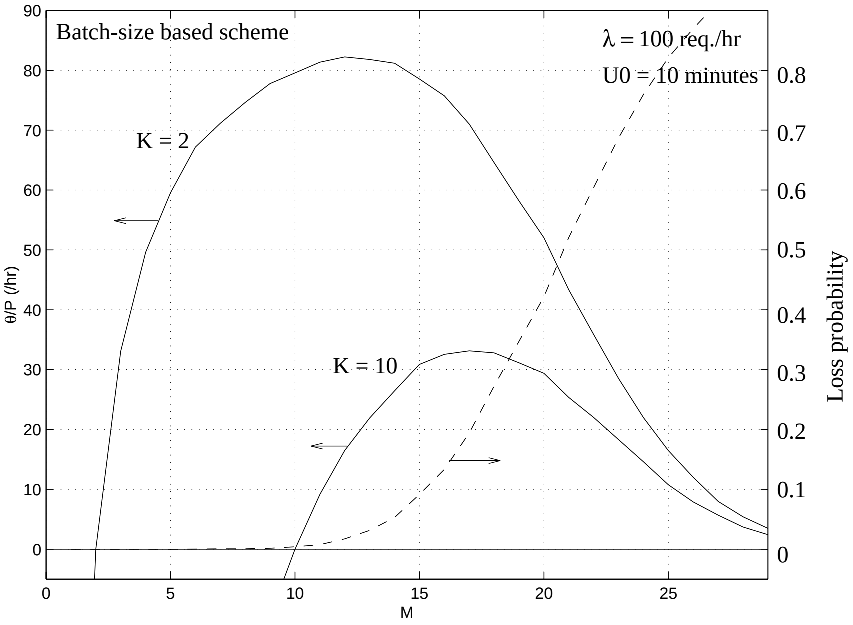


Figure 3.97: $\hat{\theta}$ and p_L versus M given K

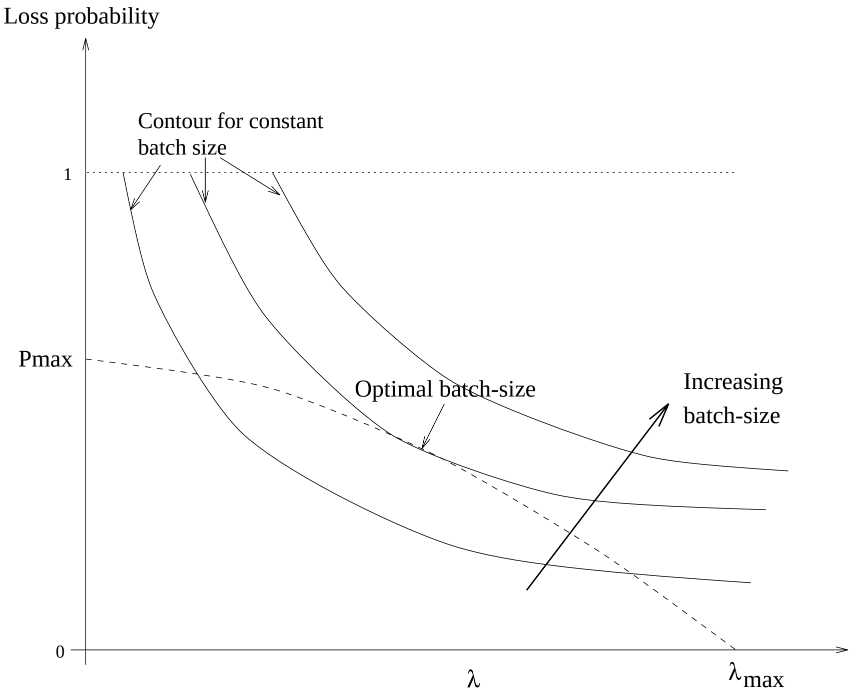


Figure 3.98: Maximizing per-batch profit with loss-dependent arrival rate

the x-axis corresponds to the pure-VOD case ($p_L = 0$ and $\overline{N} = 1$), and $\overline{N}$ increases with p_L, reaching its maximum at $p_L = 1$. Since for a given p_L, $\overline{N}$ increases with λ, the iso-$\overline{N}$ contours increases value as one moves outwards as shown in the figure. Supposed now λ depends on p_L. The dashed line is an arbitrary loss-dependent arrival rate (we show here that a user would not use a system at all if its loss rate is higher than p_{max}, and $\lambda = \lambda_{max}$ if $p_L = 0$). The optimal operating point to maximize the per-batch profit is where the contour line touches the dashed line. We see that at the optimum, $0 \leq p_L \leq p_{max}$ and $0 \leq \lambda \leq \lambda_{max}$.

We may similarly obtain the optimal operating point to maximize $\hat{\theta}$. We plot in Fig. 3.99 p_L versus λ; we show in solid lines the contours achieving constant profit rate $\hat{\theta}$ (such contours may be obtained from Fig. 3.66). Note that the iso-$\hat{\theta}$ contours increases outwards as shown, since the higher λ is, the higher the achievable profit. The dashed line is an arbitrary loss-dependent arrival rate — we show here that a user would not use a system at all if its loss rate is higher than p_{max}, and $\lambda = \lambda_{max}$ if $p_L = 0$ (loss-independent arrival

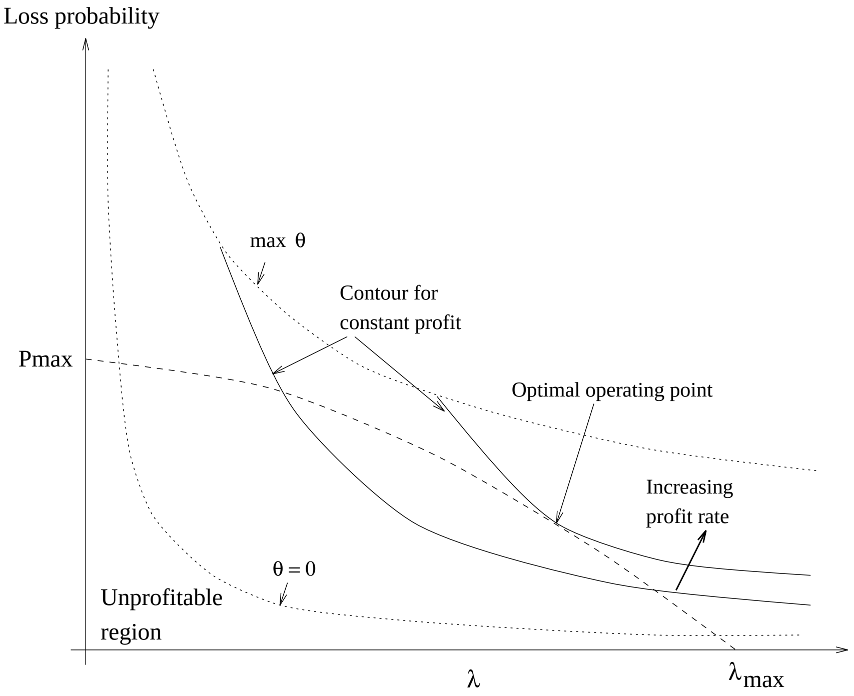


Figure 3.99: Maximizing $\hat{\theta}$ with loss-dependent arrival rate

rate is hence a verticle line). The optimal operating point is hence at which an iso-$\hat{\theta}$ lines touches the dashed line for the loss-dependent arrival rate. Note that if λ is dependent on p_L, the optimal $\hat{\theta}$ is lower, and the loss rate is always less than p_{max}.

3.4.6 A combined scheme

Note that we may optimize the batching parameters (i.e., W in the window-based schemes and M in the batch-size based scheme) for a movie with a certain request rate λ in mind. In reality, the request rate may fluctuate around λ. In the auto-gating scheme, the number of users in a batch is higher when the arrival rate gets higher, leading to high profit. However, the auto-gated scheduling cannot guarantee every batch to be profitable. Therefore, when the arrival rate fluctuates and drops to a low value, the batches are more likely served at a loss, and the system profit suffers greatly. Batch-size based scheme amends this problem by making sure each batch is profitable no matter what the arrival rate is (by preferentially serving those requests arriving closely together). However, since the batch size is fixed, the scheme may allocate a channel sooner than necessary when arrival rate gets higher.

Therefore we study a combined scheme which operates according to the window-based auto-gated scheduling when arrival rate gets high, and according to the batch-size based scheme when arrival rate drops. (Such scheme is very similar to the adaptive scheme given in [84].) This scheme keeps batching requests until the batch size is no less than a parameter $M \geq K$ (so as to ensure profit) *and* the batching period (the time between the first request in the batch and the movie show-time) is no less than a parameter W (so as to safeguard against under-sized batches).

The parameters M and W are chosen from the respective batching scheme optimized for a certain request rate λ (the profit rate of the combined scheme is hence at least the higher of the two "constituent" schemes). We have used simulation to study the performance of this scheme, since it is quite difficult to obtain analytically its performance.

We show in Fig. 3.100 $\hat{\theta}$ versus λ for the combined scheme based on simulation, with $W = 10$ minutes, $M = 15$ and exponential user reneging function ($U_{min} = 0$, $\tau = 45$ minutes and $K = 10$). Also shown are the auto-gated scheduling with the same W and the batch-size based scheme with the same M. The parameters are almost optimal for each of the scheme at the target arrival rate ≈ 100 req./hr (ref. Figs. 3.56 and 3.89). We see that as λ goes higher than the target rate, auto-gating achieves higher profit; and when it drops below it, batch-size based scheme achieves higher profit. The combined scheme traces out the outer "envelops" of the two basic schemes.

We show in Fig. 3.101 p_L versus λ of the same system. The combined scheme traces out the outer envelops of p_L of the basic schemes, settling to a limiting loss rate as λ increases. Note that the auto-gating offers a rather desirable low p_L with λ, while in the batch-size based scheme p_L can vary greatly with λ.

We plot in Fig. 3.102 $\hat{\theta}$ versus λ when τ is increased to 150 minutes, for the combined scheme and the two basic schemes. Here W and M are chosen close to the optimal values of the basic schemes at the targetted arrival rate $\lambda = 100$ req./hr ($W = 30$ minutes and

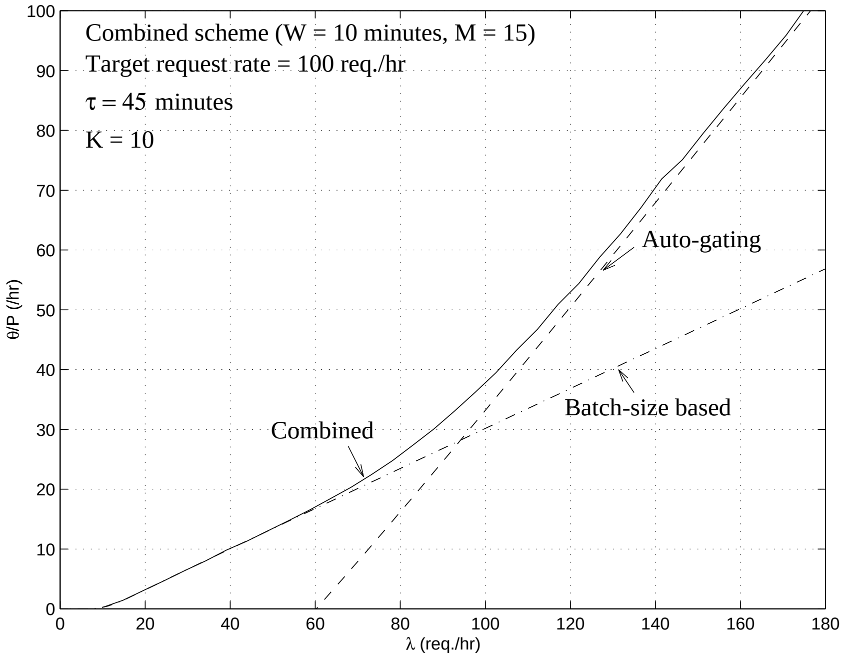


Figure 3.100: $\hat{\theta}$ versus λ for the composite scheme, with $\tau = 45$ minutes

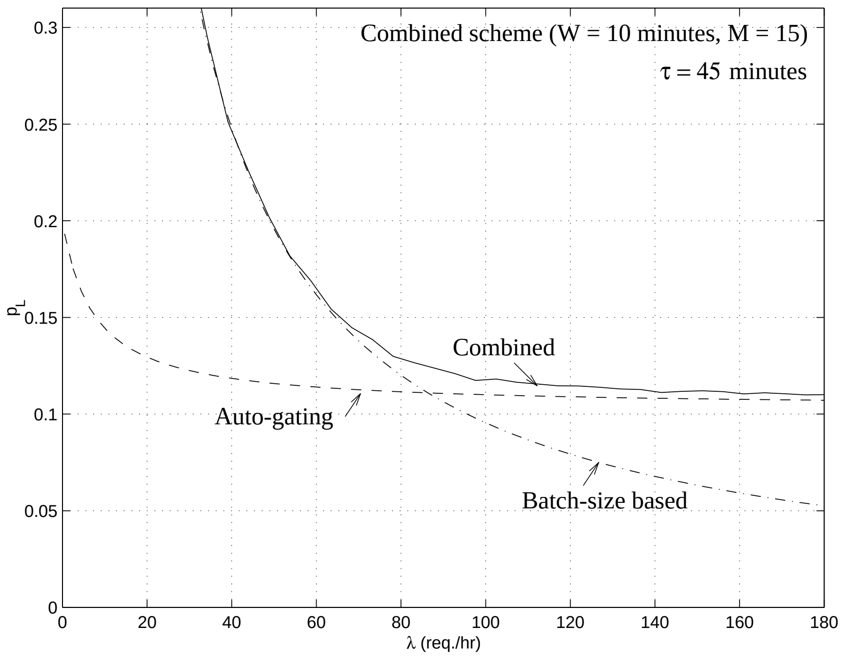


Figure 3.101: p_L versus λ for the composite scheme, with $\tau = 45$ minutes

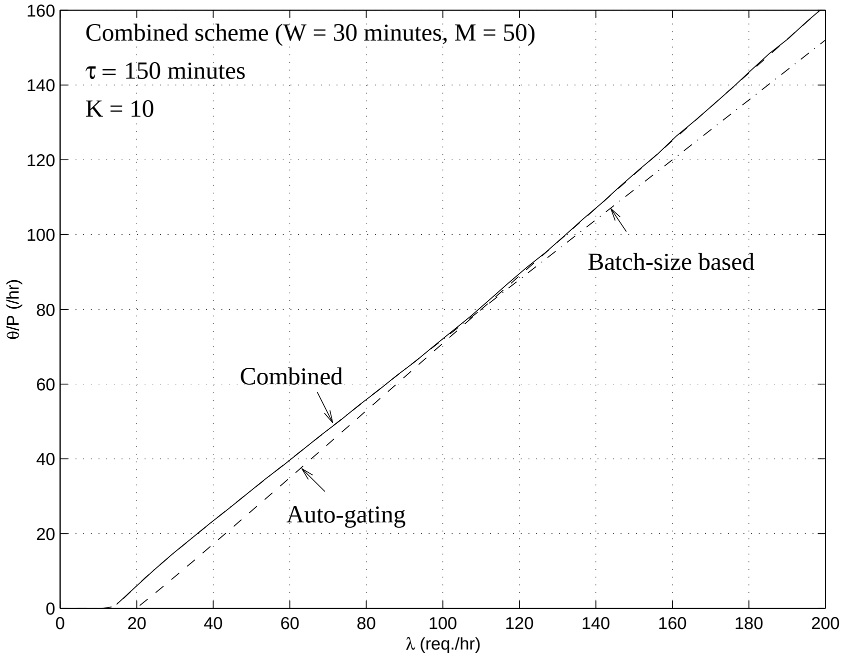


Figure 3.102: $\hat{\theta}$ versus λ for the composite scheme, with $\tau = 150$ minutes

$M = 50$).[3] With the parameters considered, the system is highly profitable and there is not much differences in $\hat{\theta}$ among the three schemes. The combined scheme is able to increase profit when it is low (due to, for example, high K or low user tolerance in delay). For the auto-gating, the minimum λ for $\hat{\theta} \geq 0$ greatly decreases.

We show in Fig. 3.103 the corresponding p_L. Even though there is not much different in $\hat{\theta}$, p_L for both the combined scheme and the batch-size based scheme increases quite rapidly as λ falls below the targetted λ (= 100 req./hr). Therefore, if the arrival rate is unlikely to fall much below the targetted λ at which $\hat{\theta}$ is already high, the auto-gating is a good choice.

[3] Note that for the auto-gating, $W^* = 46$ minutes, $\hat{\theta}^* = 73.09$ req./hr, and $p_L^* = 14.15\%$; with $W = 30$ minutes, $\hat{\theta} = 70.9$ req./hr and $p_L = 9.57\%$. For the batch-size based scheme, $M^* = 68$, $\hat{\theta}^* = 73.32$ req./hr, and $p_L^* = 14.04\%$; with $M = 50$, $\hat{\theta} = 71.9$ req./hr and $p_L = 10.12\%$.

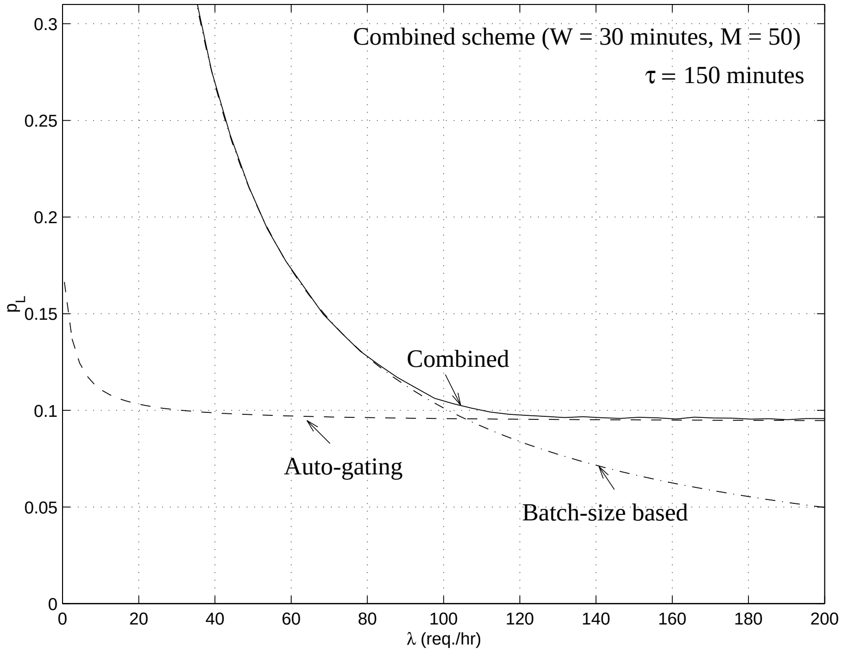


Figure 3.103: p_L versus λ for the composite scheme, with $\tau = 150$ minutes

3.5 Conclusions

A near video-on-demand system achieves streaming scalability by batching requests for a particular video and serving them with a single multicast stream. We have considered providing such a service when there is a cost associated with using a multicast channel. Batching has then to be designed so as to amortize such channel cost.

We have studied a number of batching schemes. In the window-based batching schemes, user delay is bounded by a certain maximum value W; and in the batch-size based scheme, revenue can be maintained in each batch by allocating a stream whenever a certain number M of users are collected. We have analyzed and compared these schemes in terms of user delay experienced, the number of concurrent streams used, the number of users in a batch, etc. When the arrival rate is high, it is advantageous to use window-based scheme due to its bounded delay and high profit, but when the arrival rate is not high, it is advantageous to use batch-size based scheme to maintain profitability. We therefore proposed a combined adaptive scheme in which system profit and service quality (in terms of user delay experienced) can be balanced.

We have also studied the minimum number of channels required to satisfy a certain user delay requirement, and we find that when the delay requirement is low and the request rate is not high, the number of channels required can be quite low.

We have considered how profit can be maximized in providing near VOD services, given a certain user reneging behavior. We find that maximizing profit per-batch generally leads to long batching period and high user loss rate. Maximizing profit rate, on the other hand, encourages more frequent smaller returns. However, the loss rate may still be undesirably high, and hence a shorter batching period should be used in reality. Generally, the higher user delay tolerance is, the longer the batching period is. Our results also show that the optimal operating point also does not depend very much on the reneging function besides its mean. We find that the maximum profit rate for the window-based scheme and the batch-size based scheme are very similar, if both schemes are profitable. However, the batch-size based scheme is able to always maintain profit even when the arrival rate is low. We have therefore studied how the batch-size based scheme can be combined with the window-based scheme so that the window size can be dynamically adjusted to improve profit. We have shown that such a scheme is able to adapt to fluctuating request rete and still achieves high profit.

Chapter 4

Distributed Servers Architecture

4.1 Introduction

In Chapter 3, we discussed the use of request batching and multicast channels to limit the streaming bandwidth required (as well as the communication cost) but at the expense of user delay. Here we look at decreasing this delay by means of streaming servers which cache the requested movies locally and stream them to the users. This has the advantage of providing zero start-up delay to the users. This advantage, however, comes with the cost of additional servers and storage. It is important to note that depending on the acceptable user delay associated with request batching, it is possible that the total cost of a distributed servers architecture be yet lower.

It is considered that a number of repository servers exist which store all the video contents of interest to a large pool of geographically distributed users. To achieve large and scalable storage, the repository servers may be tertiary libraries or jukeboxes which store videos of wide range of popularity. If users were to stream their videos directly from the repository servers, then the number of users that could be served would be limited by the streaming capacity available at those servers. For example, if the central repository is a disk farm consisting of many independent disks, the number of concurrent users for a movie in a disk would be limited by the streaming capacity of that disk. Striping may be used to increase the streaming capacity, but there is generally a limit on how many disks a movie can be striped across (due to issues in access overheads, complexity in data management, disk synchronization, reliability, etc.). Movies can certainly be duplicated in a disk farm to increase the streaming capacity, but this would lead to additional storage cost. To increase the storage capacity, we may use tertiary storage systems such as libraries or jukeboxes, which consist of a large number of removable, high capacity medium such as optical disks or tapes sharing a number of drives. A robot is used to automate the loading and unloading of the disks/tapes into and from the drives. This exchange latency incurred can be quite low, in the range from several seconds to tens of seconds (if optical disks is used, this is the latency of accessing a file). In any case, the streaming capacity at a library is limited by its drives. Such limitation can be partially solved by striping data across several disks/tapes or multiple libraries [62, 63].

Streaming capacity can be increased by using a hierarchy of servers as mentioned in Chapter 2, in which multiple streaming servers get the requested movies from the libraries and then stream them to the users. This in fact may introduce some delay for movies of low popularity. If the streaming servers are co-located with the central repository, the cost of the long-distance channels needed to stream the requested videos to the remote users may be high (e.g., through the use of some satellite links or long-haul transmission lines). Therefore, streaming videos in this way to the users would not be cost-effective.

To overcome the above limitation in channel usage cost and taking advantage of locality of usage, the streaming servers may be placed closer to the users (or clusters thereof), thus forming a distributed servers architecture. A number of repository servers (collectively referred to as a central repository) stores all the video titles and deliver them to the local servers through a communication network. The local servers cache movies locally, and hence multiple requests for a movie may be served from the local cache rather than from the central repository. In this way, the bandwidth requirement in the repository and channel usage cost can both be reduced. Such a system in fact has been discussed previously in the literature [3, 94]. By putting more repository servers and local servers, the system offers

Table 4.1: Values of β and γ (based on 1 GB of disk of \$200, amortized over a year with 6 hours/day usage; and $b_0 = 5$ Mb/s)

β (\$/(channel·minute))	0.3	0.03	0.003
γ (\$/minute)	2×10^{-4}	2×10^{-3}	2×10^{-2}

scalable storage and streaming capacities.

We consider that there is a cost associated with storing a movie in a local server depending on how much and how long the storage is used. Suppose that a 1 GB of disk (which costs about \$200 today) has to be amortized over a year and in use for a number of hours in a day, say 6 hours. The storage cost per hour is hence $\$200/(365 \times 6) = \$0.09/(\text{GB}\cdot\text{hr.})$.[1] Consider the streaming rate of a movie to be $b_0 = 5$ Mb/s (MPEG-II quality), then each minute of video storage costs $\$3.42\times10^{-3}$/minute. Therefore the longer we store a movie, the more expensive it is.

Consider that there is a cost associated with a central repository streaming a movie. The cost depends on the distance and the type of network, e.g., through the use of internet, ATM, or more expensive satellite channels. Such network channels more often provide multicast capability. In Table 4.1, we show the cost of using a channel per minute β and the corresponding ratio γ between the storage cost given above (in \$/min. per minute of video) and β. (With $\beta = 0.03$, the cost of delivering a 100-minutes movie from the central servers would be \$3.) For on-demand video services, γ in general would range from around 10^{-4} (relatively expensive channels) to 10^{-2} (relatively cheap channels).

Accordingly, there is a trade-off between the cost of storing a movie locally and the cost of using expensive channels. It would pay to store a movie locally if the storage cost can be offset by a big saving in channel cost: if the demand for the movie is high, we should not

[1] Note that if redundant storage is used for fault tolerance (e.g., through mirroring), the cost of storage would be higher.

request it directly from the repository since it would incur too high the long-haul channel cost (we hence should store the movie locally); on the other hand, if the demand is low, we should not store the movie locally since it would cause too high a storage cost (we should hence request it directly from the repository servers).

Indeed, not all movies have identical popularity, some are popular while some are not, and the popularity may change over time. There is hence a need to decide what to store locally given the local request rates. It is important that such a decision be continuously made over time to take into consideration the dynamic nature of video popularity. In other words, the two systems (i.e., the central repository and local servers) should not work independently and in parallel serving their respective users; instead, there should be constant exchange of movies between the two. For example, the introduction of a recent lecture, or a new or hot movie title can upset the popularity of some movies and make some no longer worth storing locally. As an example, consider a geometric video popularity model, i.e., the access probability of the movies in the system follows a geometric distribution [14, 15]. We show in Fig. 4.1 the request rate of movie i in a system of 500 movies with the total request rate of 4000 req./hr, given a certain video popularity (r/m video popularity means that $r\%$ of the requests ask for $m\%$ of the movies). We see that the request rate of a movie can differ quite widely (by many orders of magnitudes). The more skewed the video popularity is, the wider the discrepancy in the movie request rates.

A local server may not have global information about the video contents of the other servers in the network. This may be due to, for example, limited processing capability (for constant and frequent content exchanges and updates), network limitation (e.g., disjoint networks or limited network capacity for frequent updates), or lack of incentive to exchange content information (e.g., due to un-cooperating service providers or security issues). In this case, the local servers can only request their data from the repository and they operate independently from each other. We show in Fig. 4.2 an example of which for a cable TV system. Multiple head-end servers serve their local communities through coaxial local drops. The servers operate independently from each other (since they may be run by

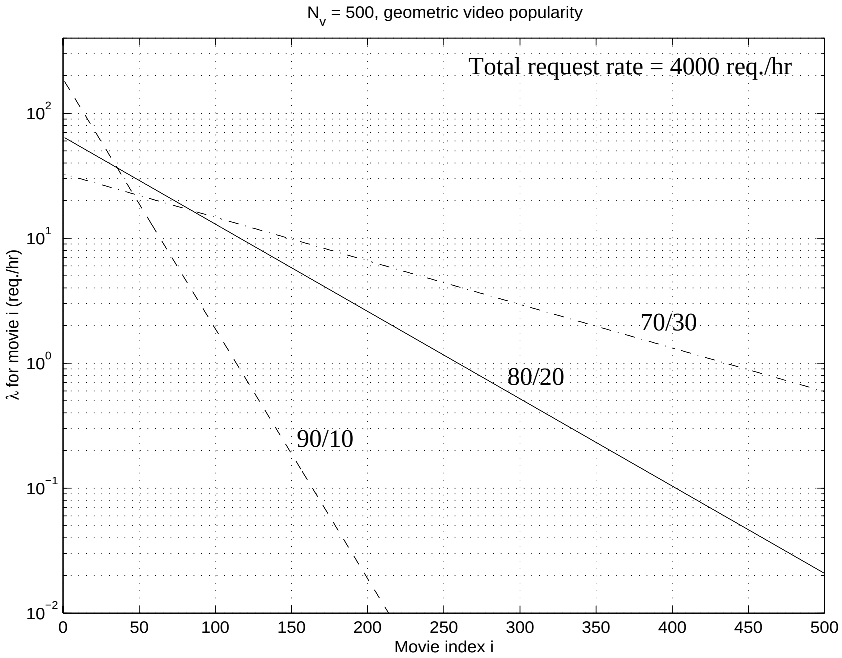


Figure 4.1: λ for movie i in a video system given skewness in movie popularity

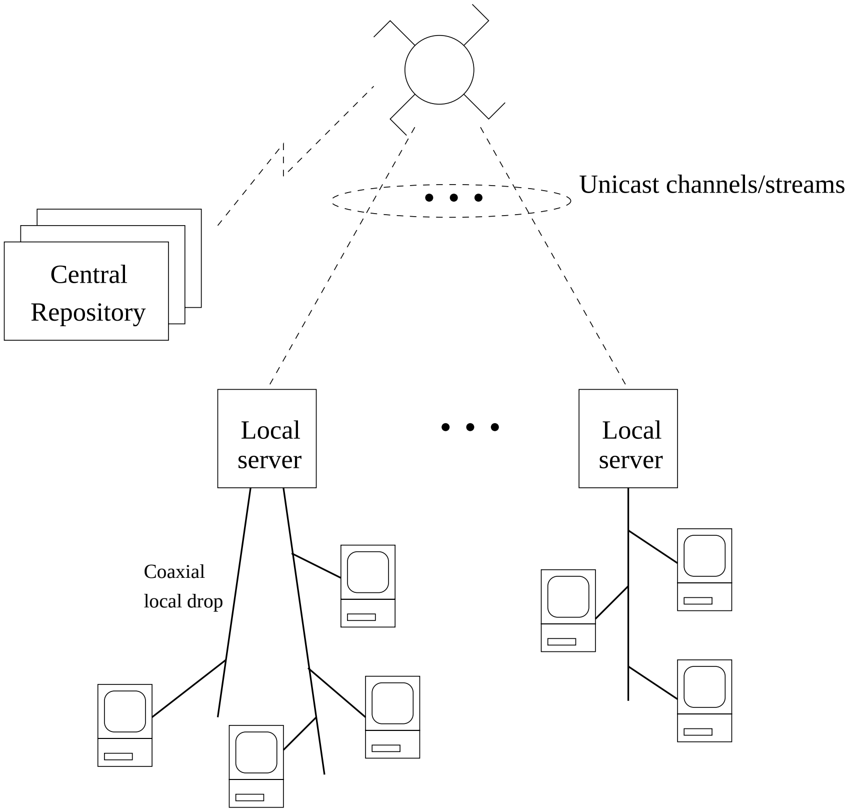


Figure 4.2: A cable system with unicast video delivery from the repository through a satellite

different providers). They get their movies from the central repository through the use of unicast channels (via a satellite system or through some low-cost channels such as internet). A certain number of network channels may be leased by the video content service providers and always available (already paid for), in which case the issue is the same as for the limited streaming capacity of the repository. Network channels may also be requested on-demand at some cost. This is the case when, for example, satellite or ATM channels are used or some tolls are to be paid in order to use a channel.

Suppose now a service provider operates a group of servers serving some regions. If multicast channels are available, they may be used to deliver a movie to the local servers in order to decrease the channel requirement and channel cost. We show in Fig. 4.3 an example of such a system, in which a provider pays for the multicast channels used in the system. A local server can freely join any existing multicast groups (but they do not

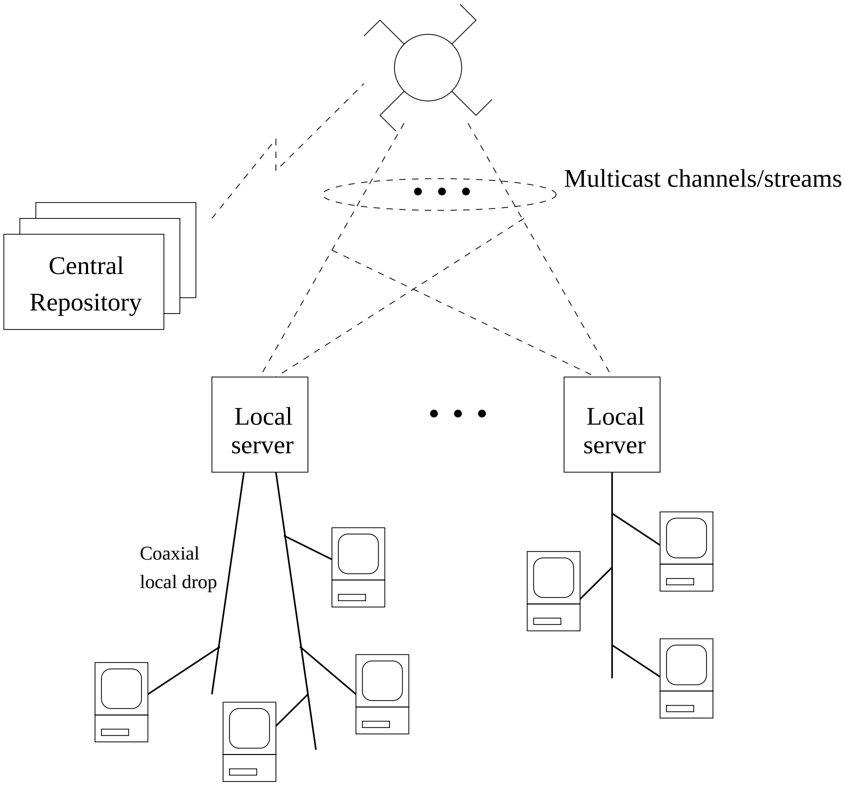


Figure 4.3: A cable system with the repository multicasting movies to the local servers

communicate with each other).

We show in Fig. 4.4 a cable video system in which the local servers are connected by a network. Therefore, instead of getting their data from the remote repository and incurring long-haul communication cost, servers can exchange content information among themselves and get their data from the other nearby servers. This is possible for servers co-located in, for example, a campus network, an entertainment network with cooperative servers/service providers, a network with low communication cost, etc. Video content may be streamed from any local server in a group to any other servers through the network. If data is not copied locally, the remote server would have to be capable of serving all the requests for that movie in the server group. By making a copy of the streamed data locally, an immediate subsequent request to the local server can be served directly from itself (instead of streaming from a remote server through the network); hence decreasing the network bandwidth. The repository may multicast a movie to the group of servers, or

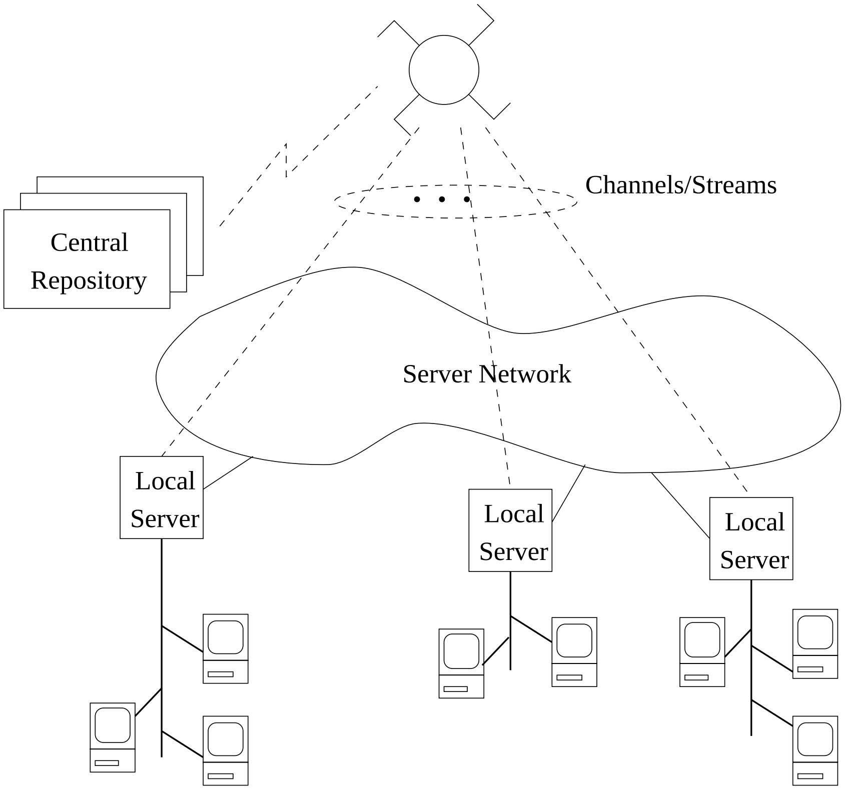


Figure 4.4: A system with communicating/cooperating servers

it may unicast the movie to a local server which in turn multicast the movie to the other servers through the network.

In this chapter, we investigate the conditions under which a movie should be stored in order to minimize the cost, given cost functions for storage and channels. We consider a number of cases:

- Independent servers with unicast delivery — In this case, the local servers operate independently and communicate only with the central repository which serves the servers by unicast (dedicated) streams. Under this condition, movies are likely to be either entirely stored or not at all and the cut-off point in the request rate for this is well-known;

- Independent servers with multicast delivery — In this scheme, the local servers still operate independently, but the repository multicasts movies to the servers so that they can store them for future use. The optimal strategy is no longer to store a movie in its entirety or not at all; instead, a portion of the movie should be stored (corresponding to a moving window). Multicast is able to lower the cost compared with the unicast case; and

- Communicating servers — Much bandwidth can be saved if the local servers can exchange video contents with one another, so that a movie not cached locally can be obtained from another local server in the network where it is cached.

The local servers cache movies of interest to users. If a local server has limited amount of storage, it would be of interest to devise some schemes to make use of the available storage so as to minimize the stream cost.[2] Here we consider that storage comes with a cost depending on the size and length of its usage. Since the service provider pays for the streams and storage, it is of interest to minimize the total cost of both streams and storage.

We study a number of caching schemes which allocate a certain amount of buffer for a movie when the movie is initially brought in; therefore all requests arriving before the beginning part of the movie is deleted can be served directly from the local cache (the group of requests will be called a *cache group*), hence saving extra repository streams. In this way, the caching scheme keeps a moving window and is able to keep a portion of a movie locally (i.e., *partial* movie caching). Keeping a portion of movie locally also provides interactive flexibility to the users in viewing the movies.

Since movies can be cached partially, the schemes achieve trade-off between channel cost and storage cost. We consider that the network channels can be acquired on demand with a cost, and that the local servers have sufficient bandwidth to serve the requests from the users. We will consider that the latency in the central repository is low (and can be ignored).

Previous work in distributed servers architecture mainly focuses on either storing a file completely in a local server or not storing it at all [14, 13, 95, 96, 97, 98, 94]. Here

[2] Obviously if movies can only be stored either completely or not at all, the optimal strategy is to store the most popular movies in the local servers.

we consider schemes in which movie can be partially cached locally, and hence is able to achieve better trade-off. We address in our study when and how much to cache. We have considered using multicasting as a means to deliver data to the local servers, which has not been studied before in this context. Previous work has been reviewed in Sect. 4.2. Note that this system is very similar to the distributed storage architectures as discussed in Chapter 2. The difference is that, while we consider leased channel case there, we consider here network channels can be acquired on a demand basis. Furthermore, in Chapter 2 we consider sizing system storage and bandwidth to meet a certain user delay requirement, while in here we study sizing the system parameters to minimize the on-going total system running cost.

This chapter is organized as follows. After a brief review of the previous work in Sect. 4.2, we describe in Sect. 4.3 the caching schemes we study in this chapter. In Sect. 4.4, we analyze the schemes in terms of their storage and channel requirements, and how cost can be minimized given the trade-off conditions. In Sect. 4.5 we provide some numerical results, and in Sect. 4.6 we compare the cost of the distributed servers architecture with that of a batching system and shows its cost advantage. We conclude in Sect. 4.7.

4.2 Previous Work

Prior work related to storage scalability and streaming scalability has been respectively reviewed in chapters 2 and 3. We discuss here the work related to distributed servers architecture, in which multiple servers in a network serves a pool of users and a movie may be duplicated or cached in order to satisfy some performance objectives. (The work related to hierarchical storage systems in a distributed environment has been reviewed in Chapter 2.)

Barnett and Anido in [94] compare a centralized video system with a distributed system in terms of their storage and streaming costs. In their centralized model, a number of repository servers serve a large pool of users, while in the distributed system, a number of front-end servers serve their respective local users; the local servers store permanently the most popular video titles and the repository servers is used to stream the unpopular movies. Given that the total cost of a server in the system is proportional to the product of the number of streams and the amount of its storage, they show that an optimally designed distributed system may achieve lower cost than a centralized system.

Giovanni et. al. consider a hierarchical storage system in which local servers get their videos from a central repository through a network. They minimize the total cost of the system given a certain network bandwidth cost and local storage cost, in order to meet a certain level of user blocking probability [14]. The popular movies may be replicated in the local storage in order to serve all the requests. If the local storage is large, storage cost is too high; while if the local storage is too low, network transmission cost is high. They show that there is an optimal local storage to achieve minimal cost.

Schaffa and Nussbaumer study a VOD system in which servers are configured as a hierarchical balanced tree [13]. All video files are replicated at a certain level in the tree. The trade-off between the cost in bandwidth and storage in a server is studied. If the bandwidth cost is negligible, all the video files should be at the root; while if the storage cost is negligible, all the video files should be at the leaves and hence each client store the whole video bases. It is found that given a certain (nonlinear) cost function in bandwidth and storage, there is an optimal replication level to minimize the total cost.

Wang et. al. consider in [99] transmitting movies with constant bit rate. By storing part of the "bursty" movie frames in a local servers, the requirement in network bandwidth can be reduced. There is hence a trade-off between the network bandwidth and the local storage. The paper discusses such trade-off, and offers a number of heuristics in how much a movie should be stored given certain constraints in local storage and bandwidth.

Papadimitriou et. al. in [95] consider a VOD system in which a central repository has to deliver the videos to its users through a series of storage (caching) nodes. The storage cost in the storage nodes depends on how close the node is to the users and how long a file is stored there (the file transmission time from node to node is negligible and there is a certain transmission cost in each network link). Users reserve movies for viewing in a future time and the scheduler has to deliver the movies to the users with minimum storage cost. By examining the display schedule of the movies, the scheduler can optimally determine when, where and how long a file should be stored in a storage node, so that the storage cost can be minimized. The paper formulates the problem using a dynamic program (which proves to be not run-time efficient) and proposes a more efficient greedy heuristic which achieves less than 10% deviation from the optimum in most of the cases.

Little and Venkatesh consider in [96] a system in which multiple servers serve a large pool of users through a switching fabric. Video files can be replicated in the servers in order to satisfy user requests. Users are rejected if there is no available bandwidth. To minimize such rejection probability, they find that load should be balanced among all the disks in the system.

Lie et. al. study a distributed VOD system in which multiple servers are used to serve user demands [98]. Each server has finite storage and bandwidth, and movies are dynamically replicated and deleted to achieve load balancing among the servers. The main performance measure is user rejection rate. The authors consider a number of policies on how a movie can be replicated (such as parallel replication). They have also studied algorithms in selecting the source nodes and target nodes, the triggering threshold to replicate and delete a movie in the servers, the influence of movie popularity, etc. It is shown that there is a trade-off between the complexity of the replication and replacement schemes and the system performance.

Ghafir and Chadwick in [97] study a hierarchical VOD system in which a central repository serves a number of servers, which in turn serve a large pool of users through a switch. Files are transferred from the repository to the servers (in no time) before they are streamed. They compare a pure VOD system with a near VOD system in terms of their storage and bandwidth requirements so that user rejection probability is negligible. Users only interact with the videos through pause/resume and cancel (in the case of near VOD, a separate stream is allocated to the user interacting with the movie). Popular movies may be replicated in the local servers to increase system bandwidth. They show that substantial saving in storage and bandwidth is possible in both VOD system, especially when the movie popularity is very skewed.

Dan et. al. in [100] consider a server with a number of independent disks. To achieve load-balancing, a movie stored in a disk is divided into a number of segments each of which are dynamically replicated to one of the other disks depending on the load. The replicated segment can be deleted when storage space runs out. Though partial movie storage is considered in the paper, network bandwidth issues (its requirement and cost) have not been addressed.

Among all the aforementioned previous work, [94, 14, 13, 99] are the most relevant to this work and hence we will discuss them here. In [94], the cost of setting up a video system (consisting of a central server and some local servers) is discussed while we discuss the running cost of the system (in which users are charged on a per-usage basis), in which network channels are acquired on-demand to satisfy users' requests. Their objectives and cost functions used are also different from ours. In [14], the channels are assumed to be fixed and leased, while our work discusses on-demand channels. Our work bears great similarity to [13] (for which the server tree has only two levels). While in [13], user traffic to the servers forms a balanced tree and channel cost is a function of the distance between a local server and the users, we consider here that the arrival rate to a local server may differ from each other and the local servers are close to the users and hence the cost of streaming from a local server to its users is low. (The star topology we consider is hence a special case of [13].) In [99], the bandwidth profile of a movie (i.e., the size of each frame) has to be known beforehand in order to do the optimization and movies are transmitted with CBR, while no such assumptions are needed in our work. Furthermore, while video data is statically stored in a local server in [99], in our work video data is dynamically deleted or stored to achieve the channel-storage trade-off.

Almost all previous work mentioned above treat a video file as a single unit, and hence is either completely stored (or replicated) in a local server or not at all. We consider here movies can be partially stored locally, hence achieving lower system cost. We address when and how much to cache in our study. We have also considered on-demand network channels with a cost assoicated in using them, which has not been discussed in the previous work above. We have also studied using multicasting as a means to deliver data to the local servers, which has not been studied before in this context. Most of the previous work also focuses on multiple serving nodes being co-located and serving a large pool of users through a switch (with the servers being able to communicate with each other, and transfer movies among themselves). We have considered a distributed servers environment as a means to reduce network channel usage, in which multiple geographically-distributed servers serving their local users may or may not be able to communicate with each other.

4.3 Caching Schemes

In this section, we describe the caching schemes we study, first for the local servers operating independently, with and without multicast network channels. We then describe a scheme for communicating servers.

4.3.1 Independent servers

Unicast delivery from the repository

We consider a system in which local servers are operating independently from each other, and the central repository delivers the movies using unicast streams (e.g., Fig. 4.2). We focus on a central repository-local server pair, and describe the following caching schemes:

Lifetime caching: Let's first consider that initially there is no user being served in a local server and no movie is cached. An arrival to the local server requires a stream from the repository. Movie data is cached locally for a time W minutes before it is deleted (W is therefore the data lifetime). Users for the same movie arriving within the window W after the movie is streamed hence form a cache group (in other words,

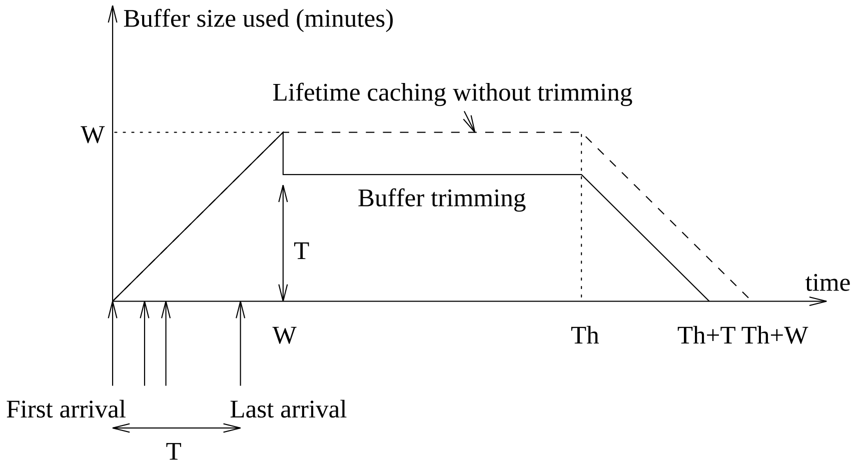


Figure 4.5: Lifetime caching with and without buffer release after a movie is displayed for time W

an arrival more than W from the first arrival in the cache group would have to start a new cache group and require another stream from the repository.)

Obviously, buffer requirement can be reduced if, after a movie has been displayed for W minutes, we "trim the buffer" and keep only the amount of data just enough to serve the first and the last arrival in the cache group. We show in Fig. 4.5 the buffer size used for lifetime caching with or without such trimming. With trimming, the buffer size corresponding to W video minutes is trimmed to T (the inter-arrival time between the first arrival and the last arrival in the cache group) after time W.

Regenerative-lifetime caching: In lifetime caching, a request arriving more than W minutes from the first user in a cache group will have to start a new stream from the repository. The regenerative-lifetime caching take into account of temporal locality of the requests: Data first brought into a local server initially has a lifetime W if there were no request arriving within this window. However, if a new request arrives within this time, the data lifetime is reset to W (and hence regenerated). Here, the more popular a movie is, the longer it will stay in the server.

We may of course consider buffer trimming as above, in which once we know that there is no arrival within W, we keep the buffer corresponding to the interarrival time between the first and last arrival in a group served by a repository stream.

Multicast delivery from the repository

We now consider that a service provider operates a group of servers serving a number of regions in which multicast delivery channels can be used (e.g., Fig. 4.3). We consider the following caching schemes in this environment:

Pre-caching: In this scheme, the central repository multicasts the movies to the local servers and a local server decides if it should cache the data in advance (i.e., pre-caching) or not. If data is cached, future requests can be served from the local cache; the buffer, however, is wasted if there is no arrival within the caching period. If the data is not cached, the local server would request streams from the repository on a request-by-request basis (i.e., the repository will *stream-through* the movies to the users). We consider the following two pre-caching schemes: periodic multicasting with pre-caching, and on-demand pre-caching.

In the periodic multicasting with pre-caching, there is a periodic multicast schedule for each movie, in which the repository multicasts a movie at regular intervals every T minutes. If data is cached, it would be cached for a lifetime $W = T$ minutes (the maximum cache size allocated for the multicast stream). If there is any arrival in the pre-caching servers within W minutes from the beginning of a multicast (as known from the presence of a server request to continue a multicast stream), the multicast channel from the repository would be held for the length of the movie; otherwise, the multicast stream would be aborted at the end of W. If there is no arrival in a pre-caching server at the end of W minutes, the local cache would be flushed.

On-demand pre-caching is very similar to the above. However, the multicast is now initiated on-demand by a local server. Other servers pre-cache the data if decided to do so, in which case if there is no request within W, the buffer will be flushed, otherwise lifetime caching will be used.

Latching: A local server in the pre-caching scheme pre-buffers data in anticipation of its use within W minutes after a movie multicast. Buffering space is hence wasted if there is no user arriving within that time. In latching, a server caches a movie by joining it multicast group (if any) once it is requested, and transient streams are used to recover the "time lag" between requests so that requests can catch up with one another and finally be served with a single multicast stream. Therefore this technique trades off buffer space with the use of additional transient streams. Since in this scheme a server has to process multiple incoming streams for a movie, this scheme takes advantage of the current processing power of a computer.

We will consider two kinds of latching: periodic multicasting with latching and on-demand latching. In periodic multicasting with latching, a movie is multicast at regular intervals of T minutes. For each multicast stream, there is a maximum buffer size $W = T$ minutes allocated in a local server (hence data is cached for at most W minutes). Supposed that initially there is no movie cached in the system. An arrival comes to a server, which requests a (transient) stream from the repository to supply the missing (beginning) portion of the movie, while at the same time it latches on (i.e., joins) the current multicast stream for the movie. Both streams will be cached, so that any arrival of the same movie from the arrival time of the first request to the next multicast time can be served directly from the local cache. After a period of time (equal to the time from the the start of the current multicast interval to the arrival time of the first request in the interval), the missing data would be supplied and the transient stream relinquished. In this scheme, if there is no request at the end of a multicast interval, the multicast stream would be aborted and a new one started; otherwise, the multicast stream will be held for the movie duration.

On-demand latching is similar to the above, except that there is no regular multicast schedule in that streams are allocated on a demand basis. Supposed that initially there is no on-going request and no movie is cached. An arrival first comes to a local server, which requests a multicast stream from the repository to stream the movie (The server may then cache the movie for a time W in anticipation of future local arrivals). Such multicast stream is called the *basic* stream, to be held for the movie duration. If another arrival for the same movie comes at a different server within W minutes, a transient stream is allocated to supply the beginning portion of a movie, while at the same time, the local server joins the basic stream and starts buffering both streams (W is hence the cache size used for the cache group). After a period of time of at most W minutes, the requests will be served solely by the basic stream. A new basic stream would be started for the first arrival more than W minutes from the previous basic stream.

Certainly transient streams do not have to be unicast streams — they can also be multicast. If so, a local server can then latch on multiple on-going multicast groups; in this case the transient multicast streams can be dropped faster, leading to lower repository bandwidth requirement. We illustrate this scheme in Figs. 4.6 and 4.7. We show in Fig. 4.6 the sequence of requests in three local servers and the corresponding buffer content on their arrivals. Initially, user 1 comes in, starting a multicast stream from the repository (the basic stream). We show the movie display line (with movie length T_h minutes), with the left end of the buffer corresponding to the position at which the movie is displayed. User 2 then comes in another local server. It first receives a (transient) multicast stream while it joins the multicast group of user 1 and starts buffering its data. User 3 arriving to yet another local server can now join the multicast groups of both users 1 and 2, while it receives its own transient stream supplying the missing portion of the movie. We show in Fig. 4.7 the result when the transient streams of users 2 and 3 are relinquished. Also shown are the display points of the movies. We see that except for user 1, both users 2 and 3 display movie at an earlier point than its receiving end.

A local server, due to its finite processing capacity, may only latch on a maximum

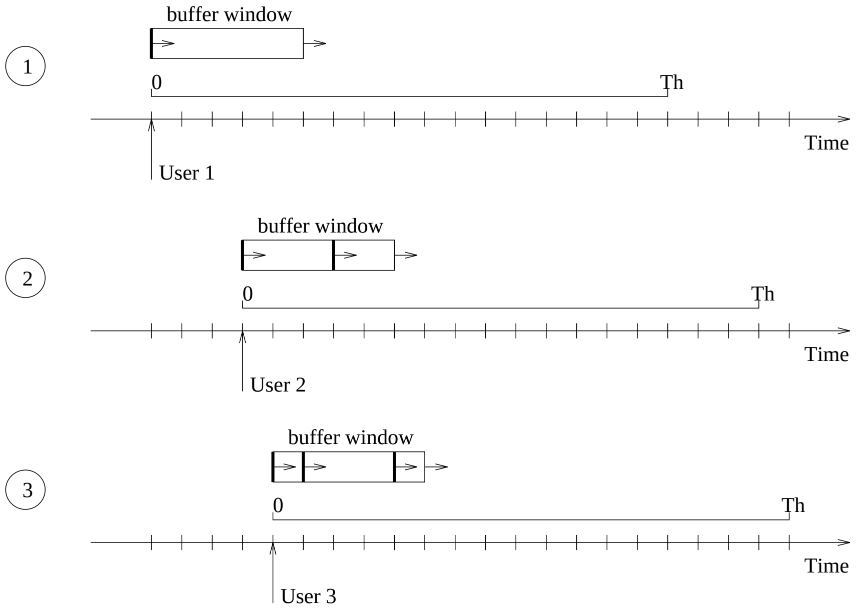


Figure 4.6: Sequence of requests and their corresponding buffer on arrival

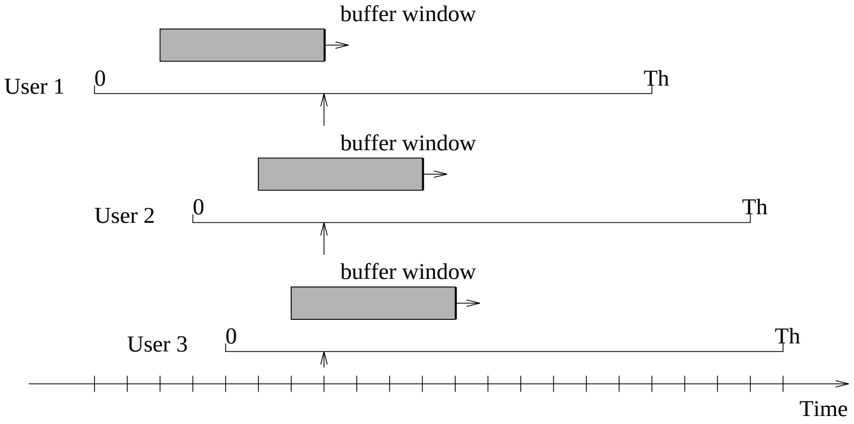


Figure 4.7: Result when all the users are finally served by the basic stream corresponding to the multicast stream of user 1

of R multicast streams in addition to its own initial stream ($R \geq 0$). Clearly, $R = 0$ corresponds to lifetime caching, and $R = \infty$ corresponds to the case in which a server can latch on all the on-going multicast streams simultaneously. For finite R and on-demand latching, a local server would latch on, in addition to its own stream, a maximum of R oldest on-going streams within W minutes from a basic stream.

Periodic multicasting and on-demand latching are quite similar to the schemes mentioned in [101, 83]. While [101, 83] focuses on offering user interactive capabilities for an on-going movie, we are interested in offering virtually no initial start-up delay here. We also consider the case $R > 1$ in our study. Simulation studies are conducted in [101, 83]; here we provide the analysis for the schemes.

4.3.2 Communicating servers

We now consider a caching scheme as used for communicating servers (e.g., Fig. 4.4), in which data is copied locally from a remote server so that future requests for the movie can be served directly from the local cache.

Chaining: Each local server uses lifetime or regenerative-lifetime caching with data lifetime W minutes. If within W minutes from the start of a movie another request of the same movie arrives at a different server, the data is copied to the server with a lifetime of W (and the movie is streamed to the user from the server). Therefore, the beginning of a movie is relayed among the local servers in the network, and hence servers are "chained." A chain is broken when successive requests from two different servers are separated by more than W minutes, in which case the later arrival will start a new chain by requesting a new stream from the repository. This scheme is similar to that mentioned in [82]. In [82], the saving in bandwidth using this technique has been studied using simulation; we quantify such saving using analysis here.

4.4 Analysis

We study the amount of storage and stream requirement for each caching scheme. We consider that the network channels can be acquired on a demand basis with a cost. Accordingly, the servicing of requests pertaining to a given movie is independent of the servicing of requests for other movies, and hence it suffices to consider the single movie case. In the following, we consider a movie of length T_h, with its request process being Poisson and its streaming rate being b_0 Mb/s. We are interested in the following performance measures:

- Average number of channels (or concurrent streams) required for the movie from the repository;

- Average storage for the movie in a local server (in minutes of video time), which may be calculated as follows: If $B(t)$ is the amount of storage (in minutes of video time) used for the movie at time t, then $\overline{B} \triangleq \lim_{T\to\infty}(1/T)\int_{t=0}^{T} B(t)dt$. Equivalently, if $B(t)$ is a regenerative process with average cycle time T_c, and define $\hat{B}$ as the time-integral of the buffer used in a cycle, then $\overline{B}$ is given by $(1/T_c)E[\hat{B}]$. $\overline{B}$ may also be obtained by $\lambda T_h \overline{b}$, where $\overline{b}$ is the time-average of the storage allocated to each request; and

- Cost: We are interested in the total system cost comprising the channel cost and local storage cost (the cost in streaming from the local servers to their users is comparatively negligible). A service provider sets up either a local server or a group of servers in providing video services, and pays for the total number and duration of the streams used, and the amount of storage used over time. Therefore his/her interest is to minimize such cost.

  Note that the storage and channel costs may be any arbitrary function in terms of the total storage used and the total streaming time. For illustration purpose, we consider linear cost functions, though our study is not limited to that (note that concave function means economy of scale). Let β be the cost per minute in using a network channel (\$/(minute·channel)), and α be the (linear) storage cost per minute for each MB used (\$/(minute·MB)).

Consider independent servers. Let $\overline{S}$ be the average number of channels used for the movie, $\overline{B}$ be its average storage, then the average cost per minute is $\alpha(60/8)b_0\overline{B} + \beta\overline{S}$. Defining $\hat{C}$ as the (dimensionless) normalized cost with respect to β and $\gamma \triangleq (60/8)b_0\alpha/\beta$, the system cost is therefore,

$$\hat{C} = \gamma\overline{B} + \overline{S}. \tag{4.1}$$

If a provider operates a group of servers, the total system cost is the sum of the local storage cost in all the local servers and the channel cost (a multicast channel cost is only incurred once when serving multiple servers at the same time.) Let $\overline{S}$ be the average number of multicast channels used in the repository. Given that there are N_s local servers, each of which using buffers $\overline{B}_i$, the total normalized system cost then becomes,

$$\hat{C} = \gamma\sum_{i=1}^{N_s}\overline{B}_i + \overline{S}. \tag{4.2}$$

Given $\hat{C}$, the average cost per user is then $\beta\hat{C}/\lambda$.

4.4.1 Independent servers with unicast delivery

Lifetime caching

Let λ be the request rate for a movie at a local server. We first consider that the buffer is not trimmed at the end of W minutes after serving the first user in a cache group. The average time between successive stream allocations from the repository (and hence successive cache group) is given by $W + 1/\lambda$, and hence the average number of channels is

$$\overline{S} = \frac{T_h}{W + 1/\lambda}. \tag{4.3}$$

Since each bit of the movie is held in the local server for a time equal to W, the average buffer size is $\overline{B} = T_hW/(W+1/\lambda)$ (we consider that buffer can be acquired in small chunk), i.e.,

$$\overline{B} = \frac{\lambda W}{1+\lambda W} T_h. \tag{4.4}$$

Therefore, $\overline{S} = \lambda T_h - \lambda \overline{B}$. Hence, $\hat{C} = \gamma \overline{B} + \overline{S} = (\gamma - \lambda)\overline{B} + \lambda T_h$. Therefore, to minimize $\hat{C}$, we either don't cache at all (the case when $\gamma > \lambda$, making $\overline{B} = 0$) or store the whole movie in the local server (the case with $\gamma \leq \lambda$, making $\overline{B} = T_h$).

Let's now consider the case in which buffer is trimmed by the end of $W \leq T_h$ after serving the first user in the cache group. Refer to Fig. 4.5 again. Let T be the inter-arrival time between the first and the last request in the cache group. Note that $\hat{B} = W^2/2 + (T_h - W)T + T^2/2$, where the first term corresponds to the first portion of the graph in which the beginning portion of the movie is being cached, the second term corresponds to the "flat" portion of the graph right after trimming, while the last term corresponds to the last portion of the graph in which buffer space is being released. Hence, $E[\hat{B}] = W^2/2 + (T_h - W)E[T] + E[T^2]/2$. Given that there are $n \geq 0$ arrivals in time W, all of them would be uniformly distributed over W; hence $P(T \leq t|n) = (t/W)^n$ (for $0 \leq t \leq W$). Therefore, $E[T|n] = nW/(n+1)$ and $E[T^2|n] = nW^2/(n+2)$. After removing the condition on n and noting the time between successive repository stream allocations as $W + 1/\lambda$, we have,

$$\overline{B} = \frac{1}{W + 1/\lambda}\left[T_h W \left(1 - \frac{1 - e^{-\lambda W}}{\lambda W}\right) + \frac{1 - e^{-\lambda W} - \lambda W e^{-\lambda W}}{\lambda^2}\right]. \tag{4.5}$$

For $W \geq T_h$, we have $E[\hat{B}] = T_h^2/2 + (W - T_h)T_h + E[T^2]/2$, where $E[T^2|n] = nW^2/(n+2)$ as given above. We hence have,

$$\overline{B} = \frac{1}{W + 1/\lambda}\left[WT_h - \frac{T_h}{\lambda}\left(1 - \frac{1 - e^{-\lambda T_h}}{\lambda T_h}\right)\right]. \tag{4.6}$$

$\overline{S}$ is the same as the un-trimmed case.

Regenerative-lifetime caching

Let λ be the request rate for a movie in a local server. We first consider the case without buffer trimming. An arrival will reset its lifetime if its arrival time is within W minutes from the previous request, i.e., with probability $p = 1 - e^{-\lambda W}$. Therefore, the expected inter-arrival time, given that successive requests are within W, is,

$$\begin{aligned} T_a &= \frac{1}{p}\int_0^W \lambda x^{-\lambda x} dx && (4.7) \\ &= \frac{1}{\lambda} - \frac{W}{e^{\lambda W} - 1}. && (4.8) \end{aligned}$$

The average time data stays in the local server is hence $L = W + \sum_{i=0}^{\infty} p^i(1-p)iT_a = W + T_a/(1-p) = W + T_a e^{\lambda W}$. The average time between successive requests of a stream is $T_s = L + 1/\lambda$; and hence the average number of concurrent streams is $\overline{S} = T_h/T_s$, i.e,

$$\overline{S} = \frac{\lambda T_h}{\left(1 - \lambda W/(e^{\lambda W} - 1)\right) e^{\lambda W} + \lambda W + 1}. \tag{4.9}$$

The average amount of buffer used for a user is given by $T_a p + W(1-p) = (1 - e^{-\lambda W})/\lambda$. Since there are on average λT_h concurrent users, the average storage used is

$$\overline{B} = T_h(1 - e^{-\lambda W}). \tag{4.10}$$

The cost is hence $\hat{C} = \gamma \overline{B} + \overline{S}$.

We now consider the case with buffer trimming. Note that, for $W \leq T_h$, the average buffer size for each request is T_a if it is not the first request in the cache group, and $W^2/(2T_h)$ if it is the first request in the group. Therefore,

$$\begin{aligned} \overline{B} &= \lambda T_h \left(T_a p + (1-p)\frac{W^2}{2T_h} \right) && (4.11) \\ &= T_h(1 - e^{-\lambda W}) - \lambda W \left(T_h - \frac{W}{2} \right) e^{-\lambda W}. && (4.12) \end{aligned}$$

4.4.2 Independent servers with multicast delivery

Pre-caching

Obviously, if a server does not pre-cache multicast streams, the average number of repository streams used is given by the movie request rate times T_h, and the buffer requirement is zero. We concentrate on the pre-caching servers in the following. Let N_s be the number of pre-caching local servers in the system, with the request rate for a movie in server i being λ_i $(i = 1 \ldots N_s)$ and the aggregate request rate among them $\lambda = \sum_{i=1}^{N_s} \lambda_i$.

Periodic multicasting with pre-caching

In this scheme, if there is no arrival within W minutes in a local server (with probability $e^{-\lambda_i W}$), an average buffer space $W/2$ minutes gets used; on the other hand, if there are arrivals, a buffer space of the movie size is used. Hence, average storage used is,

$$\overline{B}_i = e^{-\lambda_i W}\frac{W}{2} + (1 - e^{-\lambda_i W})T_h. \tag{4.13}$$

With N_s servers caching the movies at the same time, total buffering is given by $\sum_{i=1}^{N_s} \overline{B}_i$.

When there is not any request among all the servers within W, the multicast stream would be used for only W minutes before it is aborted; otherwise it would be used for a time T_h. Hence the average number of concurrent streams is

$$\overline{S} = \frac{1}{W}\left(We^{-\lambda W} + (1 - e^{-\lambda W})T_h\right). \tag{4.14}$$

We formulate the cost optimization problem as follows. Let N be the total number of local servers in the system, with the request rate in server i ($i = 1 \ldots N$) given by λ_i. Associated with each server i is an pre-caching index x_i with $x_i = 1$ indicating that the server pre-caches a multicast stream and $x_i = 0$ otherwise. We then solve the following programming for the optimal multicasting schedule W^* and caching strategy x_i:

$$\begin{array}{rl}
\text{Minimize} & \gamma \sum_{i=1}^{N} \left(\frac{W}{2}e^{-\lambda_i W} + (1 - e^{-\lambda_i W})T_h\right) x_i + \\
& e^{-\sum_{i=1}^{N} \lambda_i x_i W}(1 - \prod_{i=1}^{N}(1 - x_i)) + \\
& (1 - e^{-\sum_{i=1}^{N} \lambda_i x_i W})\frac{T_h}{W} + \\
& \sum_{i=1}^{N} \lambda_i T_h (1 - x_i), \\
\text{With respect to} & W, x_1, x_2, \ldots, x_N, \\
\text{Subjected to} & W \geq 0, \\
& x_i \in \{0, 1\}, \text{ for } i = 1, \ldots, N.
\end{array}$$

Note that for uniform load (i.e., $\lambda_i = \lambda/N$), either all $x_i = 0$ (all servers do not cache) or $x_i = 1$ (all servers pre-cache). The solution in this case is then simpler.

On-demand pre-caching

In this scheme, the average time between successive channel request is given by $W+1/\lambda$, and hence,

$$\overline{S} = \frac{\lambda T_h}{1+\lambda W}. \tag{4.15}$$

In a local server, a pre-cache data is flushed at the end of W minutes after a multicast has started if there is no request within the time (with probability $e^{-\lambda_i W}$) *and* the multicast is not initiated by the local server (with probability $1-\lambda_i/\lambda$). Such probability occurs with probability,

$$\pi_0 = e^{-\lambda_i W}(1-\frac{\lambda_i}{\lambda}). \tag{4.16}$$

Therefore, using the regenerative property of the multicast request, the average buffer size used in local server i is given by,

$$\overline{B}_i = \frac{1}{W+1/\lambda}\left[\pi_0\frac{W^2}{2} + (1-\pi_0)T_h W\right]. \tag{4.17}$$

Using $\overline{S}$ and $\overline{B}_i$, the total cost function can then be formulated (similar to the case of periodic multicasting).

Latching

We will analyze periodic multicasting with latching and on-demand latching for $R=1$, and study the performance using simulation for $R>1$. We let $W(=T)$ the buffer size used to cache a movie in a local servers, λ be the aggregate request rate for the movie.

Periodic multicasting with latching

The average number of streams used in this scheme will be the average number of streams in the pre-caching case (given in Eq. 4.14) *plus* an additional term due to transient streams, $\overline{S}_{trans}$. Let the request rate at local server i is λ_i, with $\lambda = \sum_{i=1}^{N_s}\lambda_i$. Given that $n \geq 1$ requests arrives in the server within W, the earliest arrival would be on average $W/(n+1)$ minutes from the start of a multicast, which is also the time needed for a transient stream to supply the missing data for all the n users. Therefore, removing the condition on n, the average number of transient streams used for server i is given by $(1/W)\sum_{n=1}^{\infty} W/(n+1)(\lambda W)^n/n!e^{-\lambda_i W} = (1-(1+\lambda_i W)e^{-\lambda_i W})/\lambda_i W$. Summing over the N_s servers, the average number of streams in this case is therefore given by,

$$\overline{S} = \sum_{i=1}^{N_s} \frac{1}{\lambda_i W}\left(1-(1+\lambda_i W)e^{-\lambda_i W}\right) + e^{-\lambda W} + (1-e^{-\lambda W})\frac{T_h}{W}. \tag{4.18}$$

Since W is typically much smaller than T_h, and transient streams are used shorter than W, the additional storage requirement due to latching is therefore negligible compared with the total used, and hence $\overline{B}$ is given by,

$$\overline{B} = \sum_{i=1}^{N_s}(1-e^{-\lambda_i W})T_h. \tag{4.19}$$

Note that as $N_s \to \infty$ (and hence $\lambda_i \ll \lambda$), $\overline{S} = \lambda W/2 + e^{-\lambda W} + (1-e^{-\lambda W})T_h/W$ and $\overline{B} = \lambda T_h W$, as expected.

On-demand latching

Let A be the time at which a transient stream is allocated to a new request. The corresponding local server also simultaneously latches on M multicast streams ($M \leq R$), with their start-time being ordered increasingly as $A_1, A_2, \ldots, A_M$ (A_1 is the start-time of the basic stream). In order for the new request to be finally served by the basic stream, we only need to consider $A - A_1 \leq W$ (otherwise a new basic stream will be started). The transient stream serving the new request will be initially scheduled to end at time $E = A + (A - A_M) = 2A - A_M$. Since the existing M multicast streams have to be long enough to deliver the missing data to the new request and it takes $(A_m - A_{m-1})$ minutes for the mth stream to catch up with the $(m-1)$th stream in the buffer for the new request ($m = 2 \ldots M$), the ending time for the mth stream has to be updated to at least $A + (A_m - A_{m-1})$. Hence the updated ending time $E_m^{'}$ for the mth stream from its previous value E_m right before the arrival of the new request is

$$E_m^{'} = \max(E_m, A + A_m - A_{m-1}). \tag{4.20}$$

Let $\overline{S}(R)$ be the average number of concurrent streams used given R and W (Clearly, the pure-VOD case corresponds to $\overline{S}(0) = \lambda T_h$). Since the repository is serving many local servers, we will consider that each new request comes from a local server not currently being served with a multicast stream (i.e, number of servers $\gg \lambda W$). The analysis for $R \geq 2$ is difficult, and hence we have used simulation to study its performance. However, $\overline{S}(1)$ can be readily obtained by conditioning on the number of requests N eventually served as a group. Given that $N = n + 1$, the first user would hold the (basic) stream for a time T_h while the remaining n users would have arrival time uniformly distributed between 0 and W; hence the average total time of stream usage in this group is $T_h + nW/2$. The average total stream-time for each group is then $T_h + \lambda W^2/2$. Since the time between successive stream allocation is given by $W + 1/\lambda$, the average number of concurrent streams used is,

$$\begin{aligned} \overline{S}(1) &= \frac{1}{W + 1/\lambda}\left(T_h + \frac{\lambda W^2}{2}\right) && (4.21) \\ &= \frac{\lambda T_h}{1 + \lambda W}\left(1 + \frac{\lambda W^2}{2T_h}\right). && (4.22) \end{aligned}$$

Note that $\overline{S}(1)$ first decreases and then increases with W. This is expected since when W increases from zero, more users can be latched to a stream and hence $\overline{S}(1)$ decreases. However, when W further increases, the transient streams becomes longer (due to the later arrivals in a group). Hence total stream time increases, leading to high $\overline{S}(1)$. Therefore, W should not be increased beyond $\hat{W}$ given by $d\overline{S}(1)/d\hat{W} = 0$, i.e., $\lambda \hat{W}^2 + 2\hat{W} - 2T_h = 0$.

A lower bound on $\overline{S}(R)$ may be obtained by observing that $\overline{S}(R)$ achieves its minimum when $R \to \infty$, in which each new request requires an additional stream time of at least its inter-arrival time. By conditioning on $N = n$ (with $P(N = n) = p_n = (\lambda W)^n e^{-\lambda W}/n!$), the average inter-arrival time is $W/(n+1)$ and hence,

$$\begin{aligned} \overline{S}(R) &\leq \overline{S}(\infty) && (4.23) \\ &\leq \frac{T_h}{W + 1/\lambda} + \frac{1}{W + 1/\lambda}\lambda W\left[\sum_{n=1}^{\infty} p_n \frac{W}{n+1}\right] && (4.24) \\ &= \frac{\lambda T_h}{1 + \lambda W}\left[1 + \frac{W}{T_h}\left(1 - e^{-\lambda W} - \lambda W e^{-\lambda W}\right)\right]. && (4.25) \end{aligned}$$

In terms of the storage used, the average number of concurrent requests is λT_h, each of which holds a buffer of size $\approx W$. Hence $\overline{B} \approx \lambda T_h W$. The total system cost function is hence $\hat{C} = \gamma \lambda T_h W + \overline{S}(R)$.

4.4.3 Communicating servers

Chaining

Let λ be the request rate of a movie in the system. The average number of concurrent streams has already been given in Eq. (4.9), with $\lambda_i = \lambda$. By considering that the repository serves N_s local servers with N_s large (and hence each new request comes from a different local server from the servers already chained, i.e., the pessimistic case), the average buffers in the system is given by $\overline{B} = \lambda T_h W$. The cost is hence $\gamma \lambda T_h W + \overline{S}$.

4.5 Numerical Examples

In this section, we will study, for each caching scheme, the trade-off between $\overline{S}$ and $\overline{B}$, followed by discussing how much data should be cached in order to minimize system cost.[3]

4.5.1 Independent servers with unicast delivery

Lifetime caching

We first discuss lifetime caching without trimming. We show in Fig. 4.8 $\overline{S}$ versus data lifetime W, given λ ($T_h = 90$ minutes). Also shown in the figure is the maximum number of streams T_h/W attained when $\lambda \to \infty$. When $W = 0$, $\overline{S} = \lambda T_h$ corresponds to the stream-through case from the repository to the local server. As W increases, $\overline{S}$ decreases from λT_h. The advantage in bandwidth is more important for larger λ than for smaller λ.

We plot in Fig. 4.9 $\overline{S}$ versus λ for $W = 60$ minutes ($T_h = 90$ minutes). For comparison, we have also shown the stream-through case from the repository to the local users. As λ increases, $\overline{S}$ first increases and then settles slowly to a limiting value T_h/W. Compared with

[3] Note that λ is shown in req./hr in all our plots.

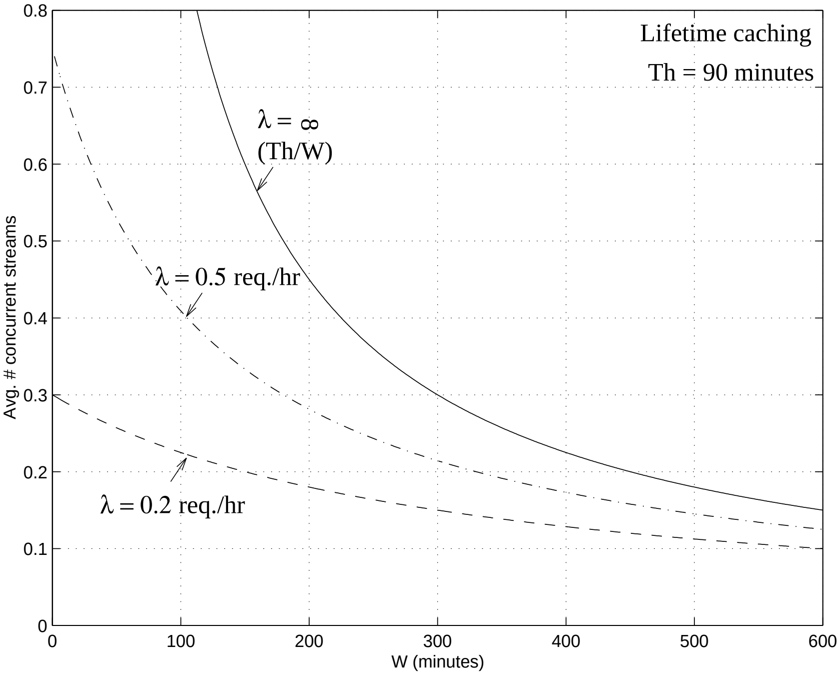


Figure 4.8: $\overline{S}$ vs. W given λ

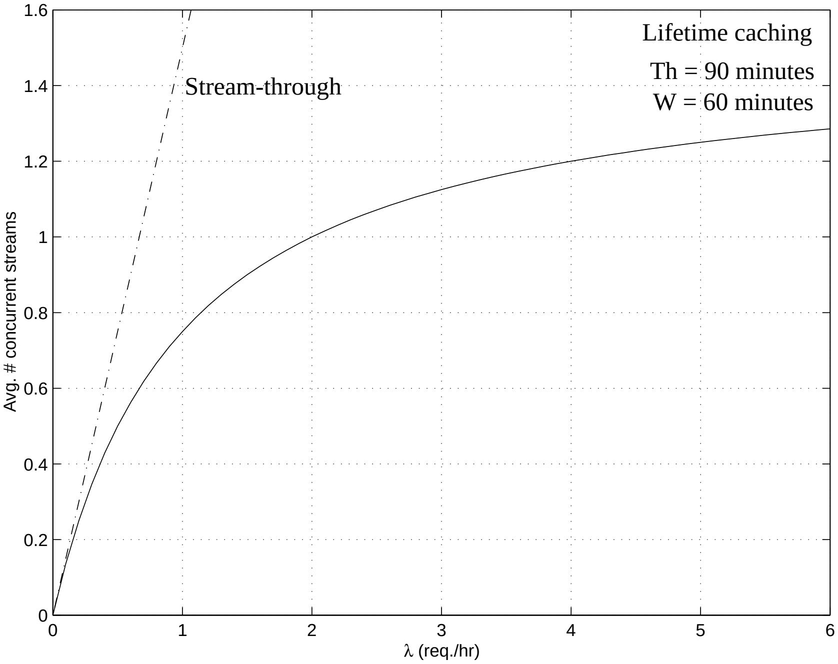


Figure 4.9: $\overline{S}$ vs. λ given W

the stream-through case, We see that a buffer size of a few tens of minutes can achieve remarkable saving in $\overline{S}$, especially when λ is not lower; this is because more and more requests will find their requested movies cached, leading to great reduction in $\overline{S}$.

We show in Fig. 4.10 the trade-off between $\overline{S}$ and $\overline{B}$ given λ. The trade-off is linear with slope $-\lambda$, beginning at $\overline{S} = \lambda T_h$ for $\overline{B} = 0$ and ending at $\overline{S} = 0$ for $\overline{B} = T_h$. Also shown is a line indicating the combination of $\overline{S}$ and $\overline{B}$ to achieve the same cost (iso-cost line); with the line moving outwards indicating increasing cost. Note that the point at which the cost line touches the trade-off curve minimizes the cost for the video (with request rate λ). With arbitrary stream and storage cost function, the cost line may touch any point on the trade-off curve. However, with linear cost function (i.e, $\hat{C} = \overline{S} + \gamma\overline{B}$), the optimal point is either $\overline{B} = 0$ (corresponding to $W = 0$) or $\overline{B} = T_h$ (corresponding to $W = \infty$), depending on $\lambda < \gamma$ or not. This condition is shown in Fig. 4.11. We see that when γ is low (storage is cheap) or λ is high (popular title), the entire movie should be stored locally.

We plot in Fig. 4.12 the normalized cost $\hat{C}$ with respect to W ($\lambda = 0.5$ req./hr, $T_h = 90$ minutes). Also indicated are the components of storage cost and stream cost. As W

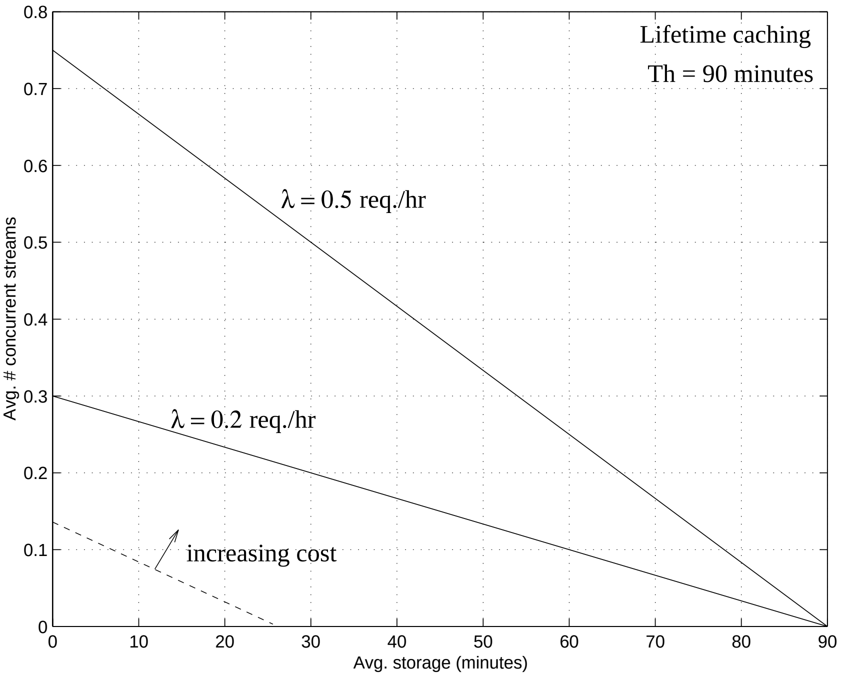


Figure 4.10: $\overline{S}$ vs. $\overline{B}$ given λ

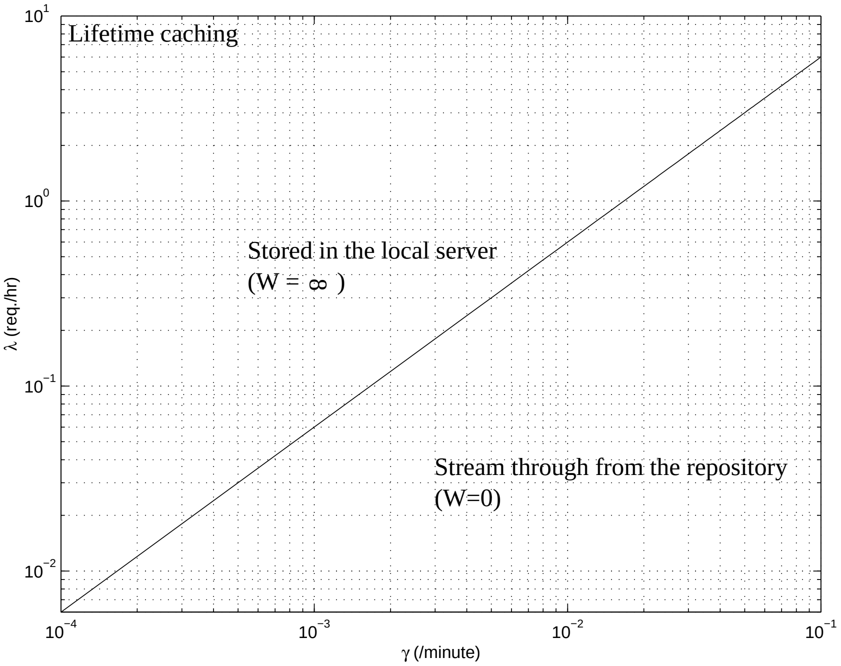


Figure 4.11: Relationship between λ and γ to minimize cost

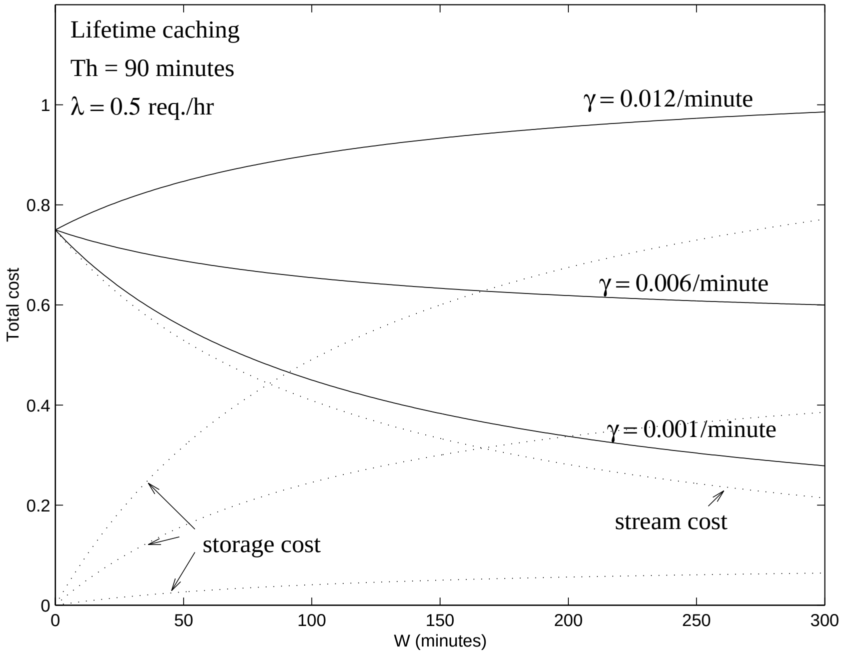


Figure 4.12: $\hat{C}$ vs. W, given γ

increases, the main component in the total cost changes from stream cost to storage cost. To minimize cost, indeed we should have either $W = 0$ (stream through) or $W = \infty$ (storing the entire movie in the local server).

We now study the effect of buffer trimming. We show in Fig. 4.13 $\overline{B}$ versus W for caching with or without trimming ($\lambda = 0.50$ req./hr and $T_h = 90$ minutes). Clearly, buffer trimming leads to good saving in buffer. We show in Fig. 4.14 the trade-off between $\overline{S}$ and $\overline{B}$ with buffer trimming. Due to its reduction in the storage requirement, the trade-off curve is no longer linear but “bends” slightly downwards, and hence we expect slightly lower cost with trimming.

We show in Fig. 4.15 the W^* required in order to minimize cost as a function of λ, given γ. As λ increases, W^* increases quite linearly until a point beyond which $W^* = \infty$. While in the un-trimmed case $W^* = 0$ is zero when $\lambda < \gamma$, in the trimmed case there is a non-zero and finite W^* to minimize cost. For a given λ, as γ decreases, W^* increases (a larger portion of the movie is stored locally).

We plot in Fig. 4.16 the corresponding $\overline{S}$ and $\overline{B}$ versus λ when $W = W^*$ ($\gamma = 0.002$

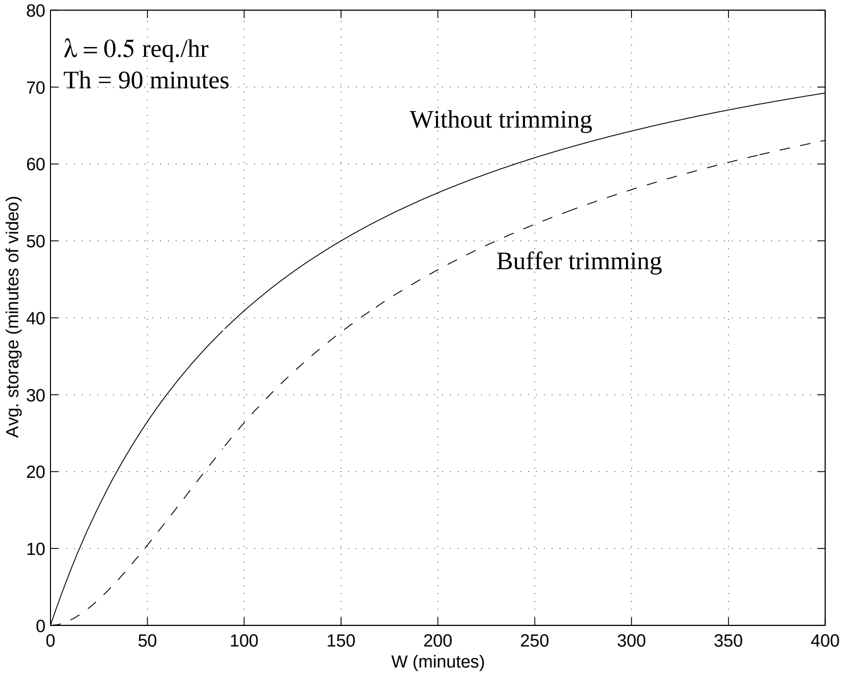


Figure 4.13: $\overline{B}$ vs. W for lifetime caching with or without trimming

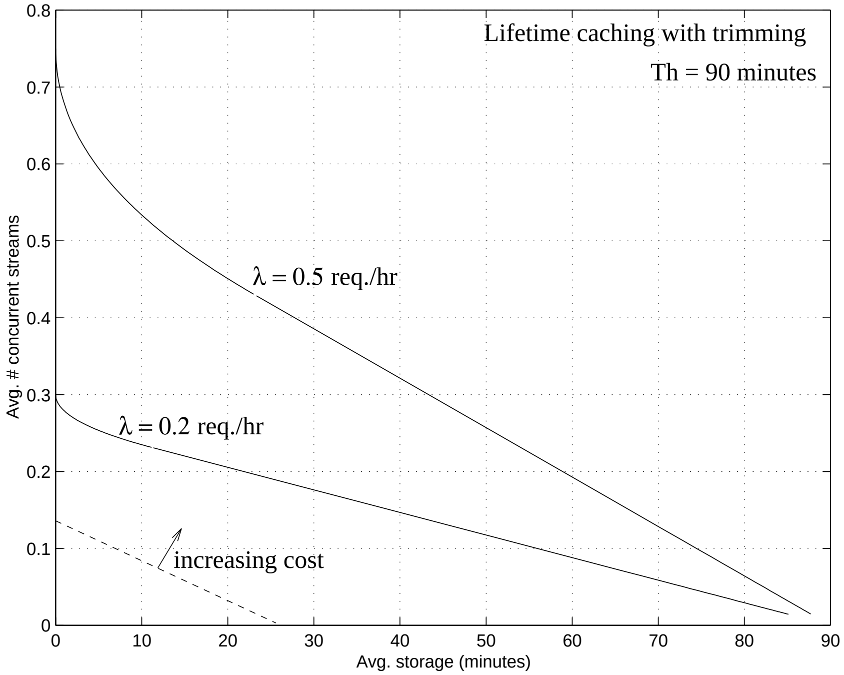


Figure 4.14: Trade-off between $\overline{S}$ and $\overline{B}$ given λ with buffer trimming

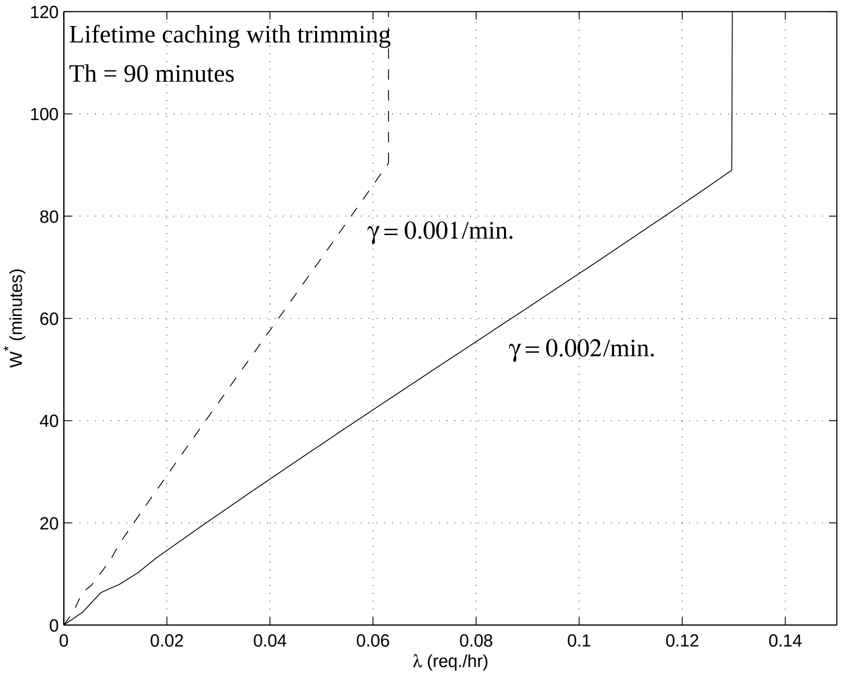


Figure 4.15: W^* versus λ given γ

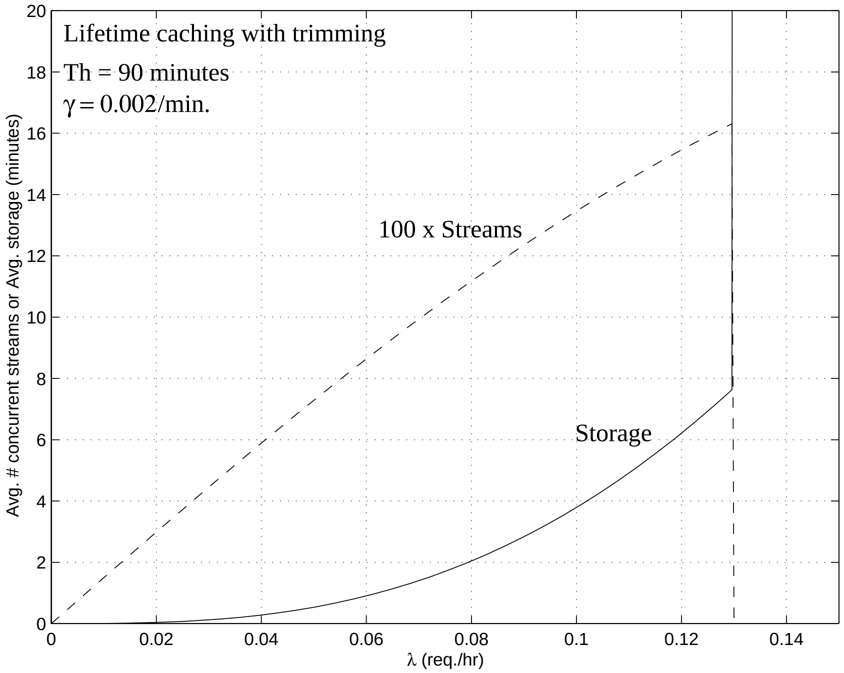


Figure 4.16: $\overline{S}$ and $\overline{B}$ versus λ with optimal W

and $T_h = 90$ minutes). As video popularity increases, video storage space increases, first slowly and then more quickly, until the point it is completely stored. $\overline{S}$ increases rather linearly with λ, and then dropped to zero as $\overline{B}$ reaches T_h.

We show in Fig. 4.17 the minimal normalized cost $\hat{C}^*$ versus λ (achieved when $W = W^*$), with and without buffer trimming ($\gamma = 0.002$ and $T_h = 90$ minutes). (Recall that in the case of no trimming, when $\lambda < \gamma$, the movie should be streamed through from the central repository; and when $\lambda \geq \gamma$, the movie should be completely stored in the local server.) We see that despite that trimming achieves lower buffer requirement, due to the low buffer cost, the cost reduction does not turn out to be great. Trimming only slightly lowers the total system cost, with the maximum difference being at $\lambda = \gamma$. Such saving is increased as γ increases.

In Figs. 4.18, 4.19 and 4.20, we show W^*, the corresponding $\overline{S}$ and $\overline{B}$, and $\hat{C}^*$ versus λ, respectively, when γ is increased to 0.01/minute. We see that, given λ, W^* is now decreased (since storage is now relatively more expensive) and hence $\overline{S}$ increases. With an increase in γ, trimming leads to more saving in system cost.

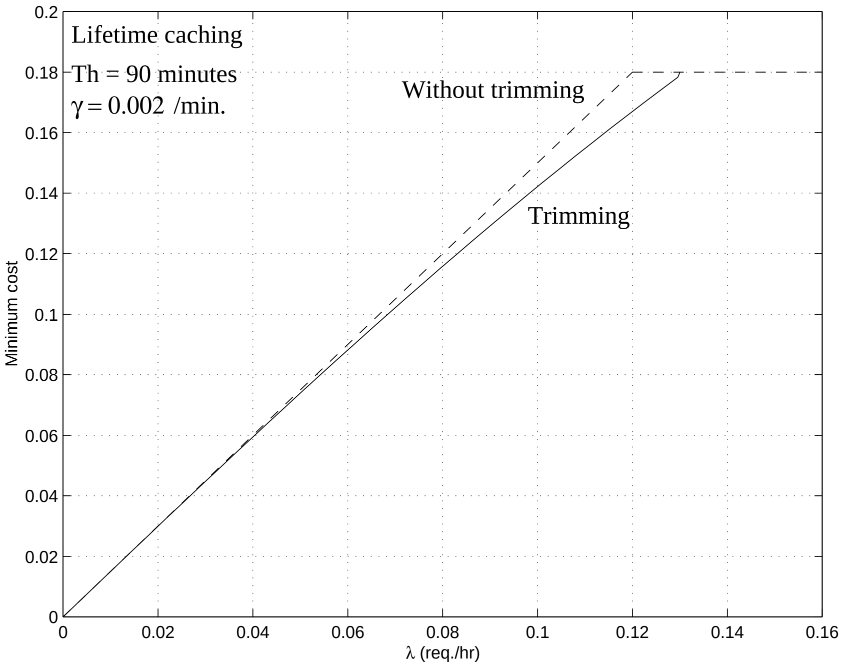


Figure 4.17: $\hat{C}^*$ versus λ, with or without buffer trimming

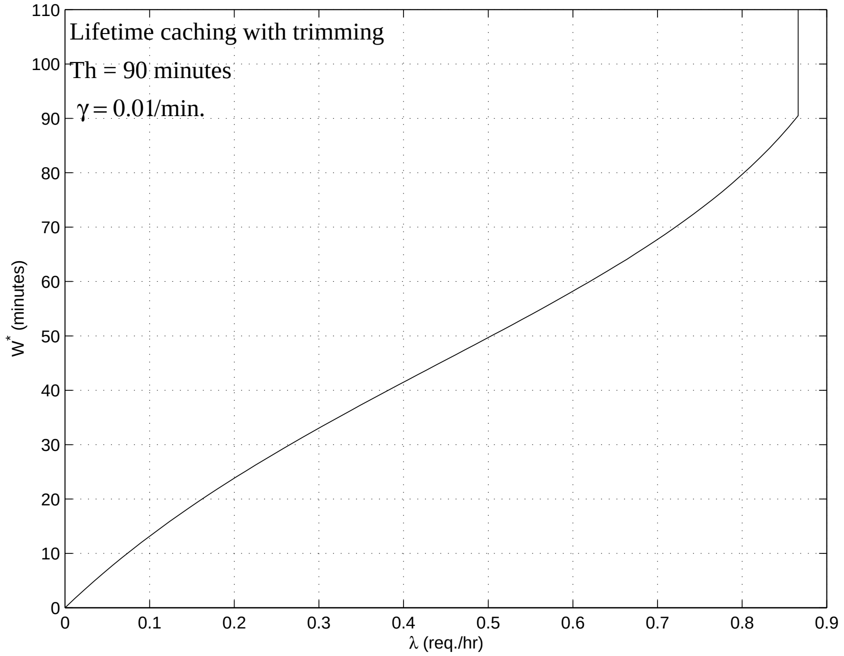


Figure 4.18: W^* versus λ with $\gamma = 0.01$

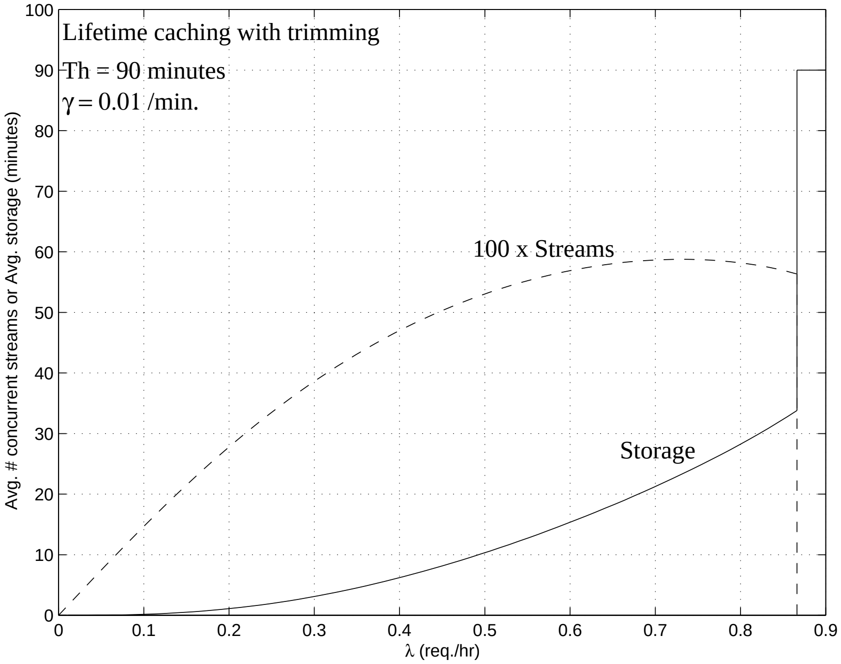


Figure 4.19: $\overline{S}$ and $\overline{B}$ versus λ with optimal W

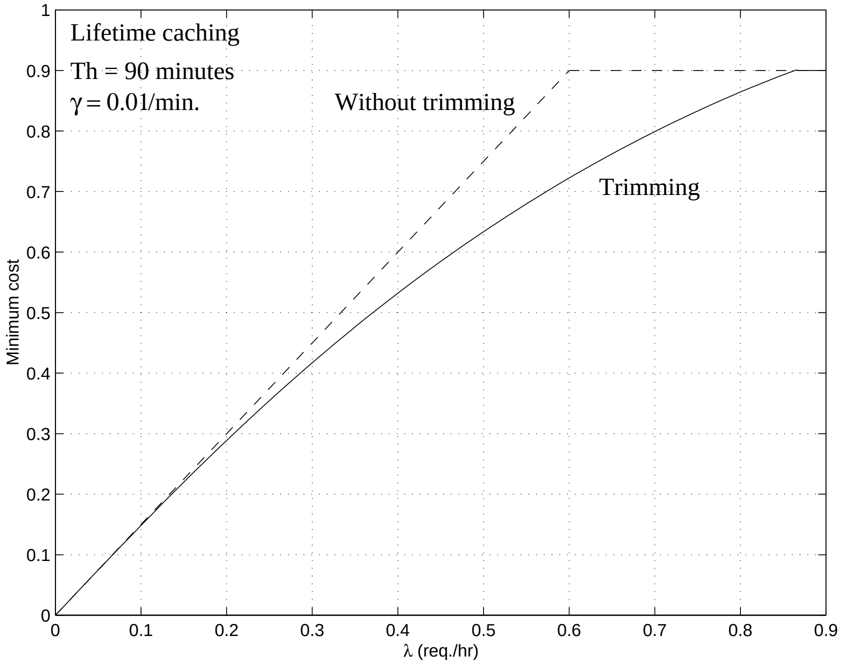


Figure 4.20: $\hat{C}^*$ versus λ, with or without buffer trimming

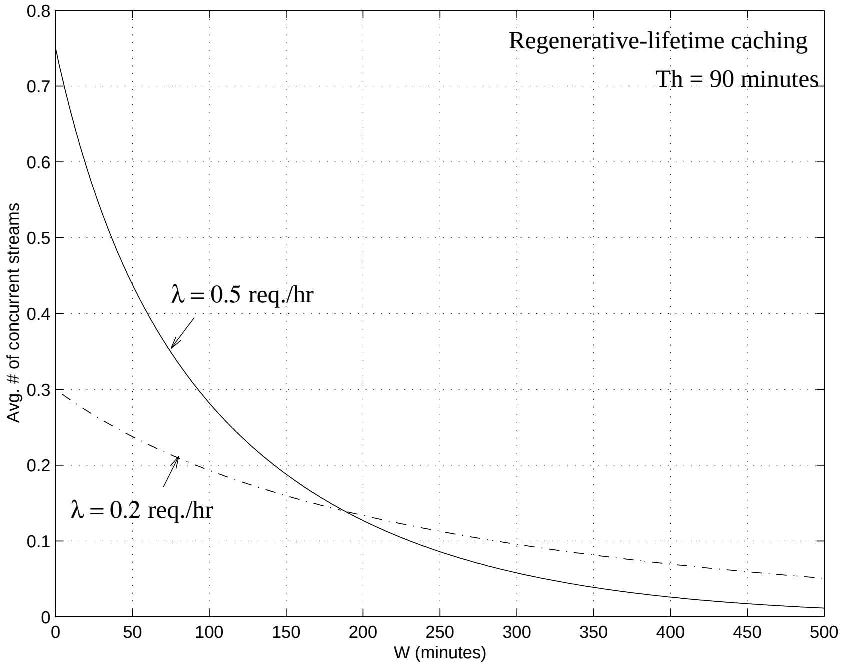


Figure 4.21: $\overline{S}$ vs. W given λ

Regenerative-lifetime caching

We show in Fig. 4.21 $\overline{S}$ with respect to W for regenerative-lifetime caching, given λ. As W increases, $\overline{S}$ decreases, first quite sharply and then quite slowly. Great bandwidth saving is achieved when $W \approx 2/\lambda$ (note that the data lifetime is likely regenerated by a successive request with this W). We show in Fig. 4.22 $\overline{S}$ versus λ, given W. As λ increases, $\overline{S}$ first increases (due to the increase in request rate) and then decreases (since data lifetime is likely to be regenerated). Given W, there is a maximum $\overline{S}$ as λ increases (attained when there is a request every W, i.e., $\lambda \approx 1/W$).

We show in Fig. 4.23 the trade-off between $\overline{S}$ and $\overline{B}$ given λ. The trade-off line is quite linear (with slope $\approx -\lambda$), beginning at $\overline{S} = \lambda T_h$ for $\overline{B} = 0$ ($W = 0$) and ending at $\overline{S} = 0$ for $\overline{B} = T_h$). To minimize cost with linear cost function, therefore, we either stream through the video or store it completely in the local server depending on if $\lambda < \gamma$ or not (as in the lifetime caching case). Fig. 4.11 can hence be used for the regenerative-lifetime caching to determine when to cache.

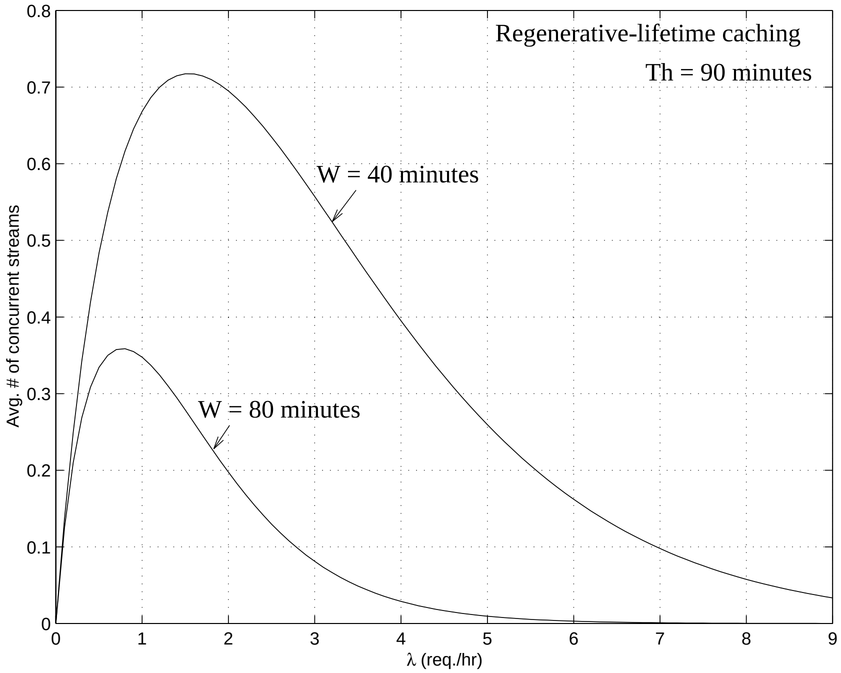


Figure 4.22: $\overline{S}$ vs. λ given W

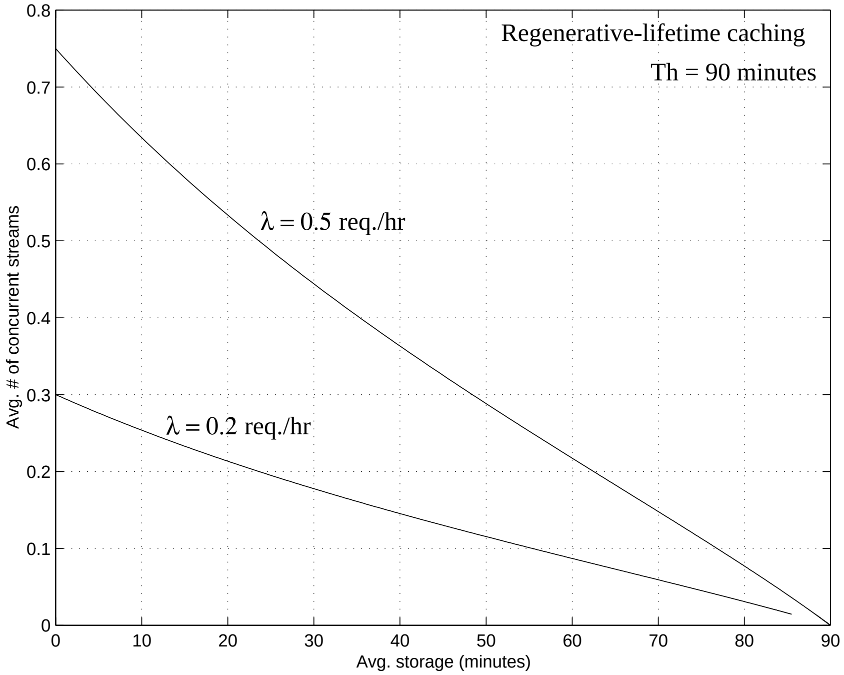


Figure 4.23: $\overline{S}$ versus $\overline{B}$ given λ

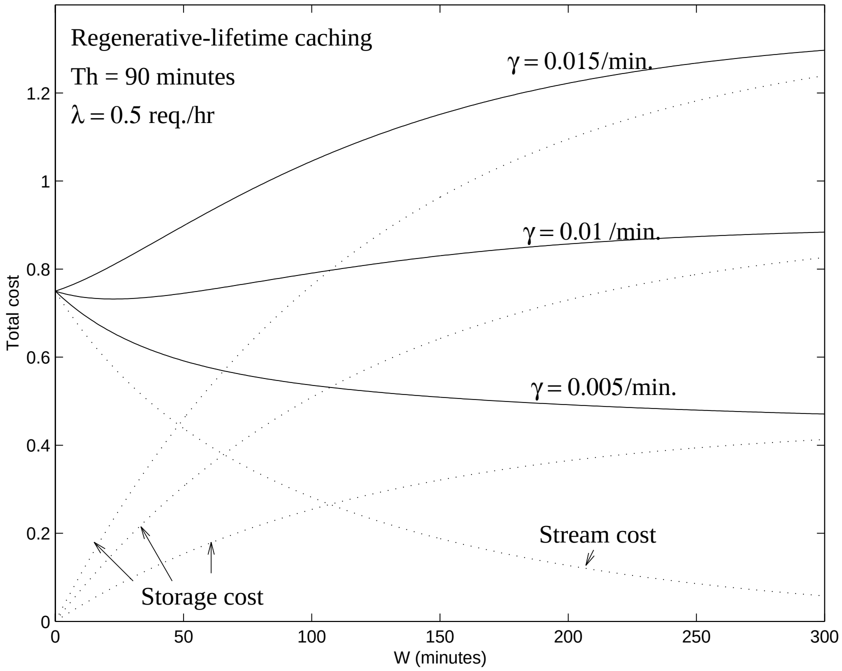


Figure 4.24: $\hat{C}$ versus W given γ, along with the components of storage cost and stream cost

We show in Fig. 4.24 $\hat{C}$ with respect to W given γ, along with the components of storage cost and stream cost. As W increases, the total cost shifts from mainly stream cost to mainly storage cost. We see that either we should have the movie streamed-through from the repository ($W = 0$) or completely store the movie ($W = \infty$), depending on if γ is higher than λ or not.

4.5.2 Independent servers with multicast delivery

Pre-caching

Periodic multicasting with pre-caching

We now consider periodic multicasting with pre-caching. We show in Fig. 4.25 $\overline{S}$ versus W given λ. We see that $\overline{S}$ decreases with W, first quite fast and then quite slowly. Most of the saving in $\overline{S}$ is achieved when W is low (10–30 minutes). This is so especially when λ is high. Note that when $W \approx 0$, $\overline{S} \approx \lambda T_h + 1$ (ref. Eq. (4.14)). The one additional stream

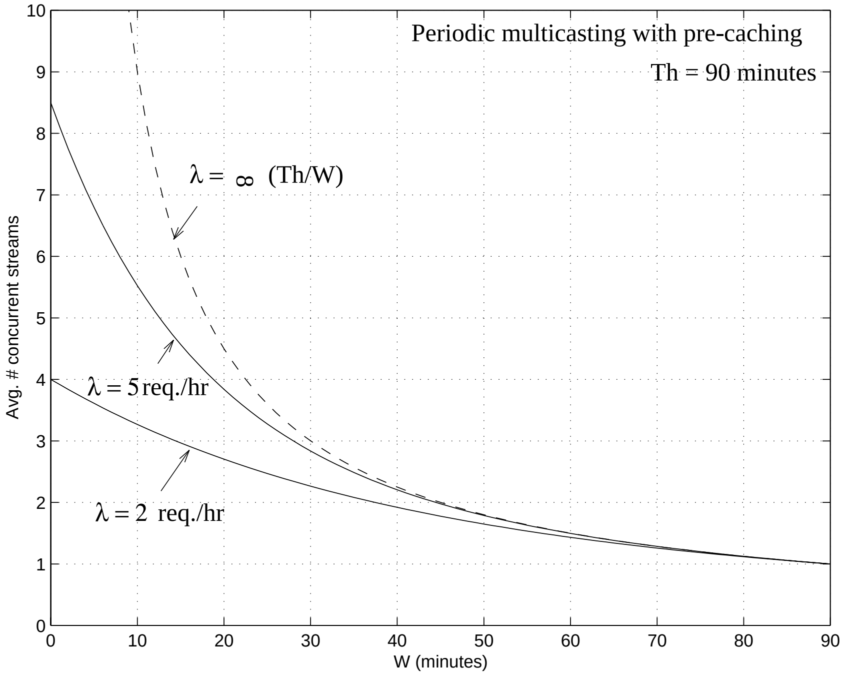


Figure 4.25: $\overline{S}$ versus W given λ for the periodic multicasting with pre-caching scheme

compared with λT_h is due to the wastage in streaming time when there is no request in all the local servers within W.

In Fig. 4.26, we show $\overline{S}$ as a function of λ given W. Compared with the stream-through case (the linear line), periodic multicasting with pre-caching generally achieves much lower $\overline{S}$, especially when W is high. As λ increases, $\overline{S}$ settles to a limiting maximum value (T_h/W) while the $\overline{S}$ for the stream-through case increases indefinitely.

From Fig. 4.26 we observe that given W and when λ is low, $\overline{S}$ for the stream-through case is lower since multicasting time would not likely be wasted (stream-through achieves at most 1 stream lower than the pre-caching scheme). We show in Fig. 4.27 the combination of λ and W so that the caching scheme achieves lower $\overline{S}$ than the stream-through case (by setting $\overline{S} \leq \lambda T_h$ in Eq. (4.14)). We see that most of the time periodic multicasting with pre-caching consumes fewer streams; only under very low W or λ would stream-through is likely better.

Recall that in a distributed servers architecture, multiple local servers may be used to increase the streaming capacity. Since these servers cannot share storage, total storage cost

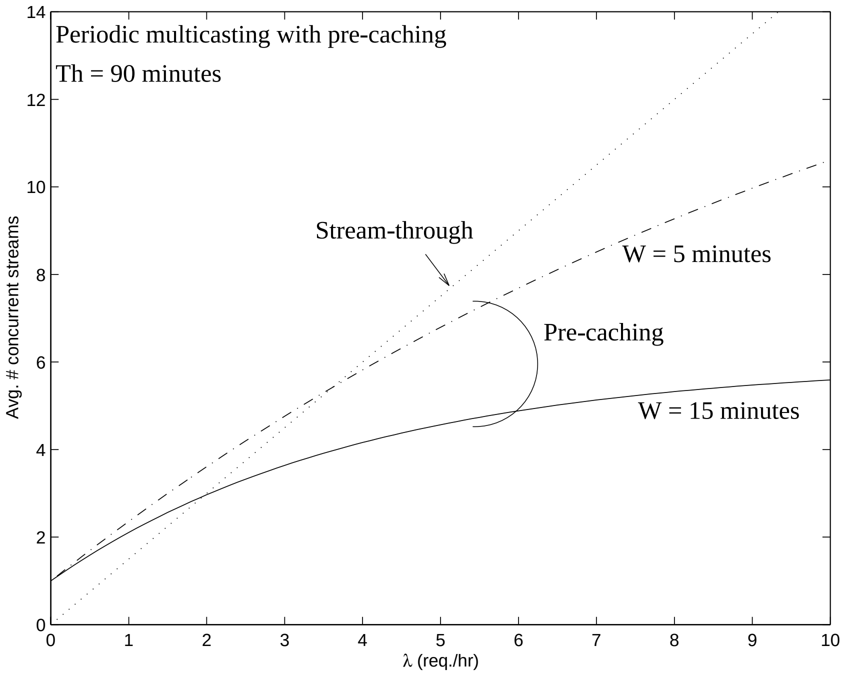


Figure 4.26:

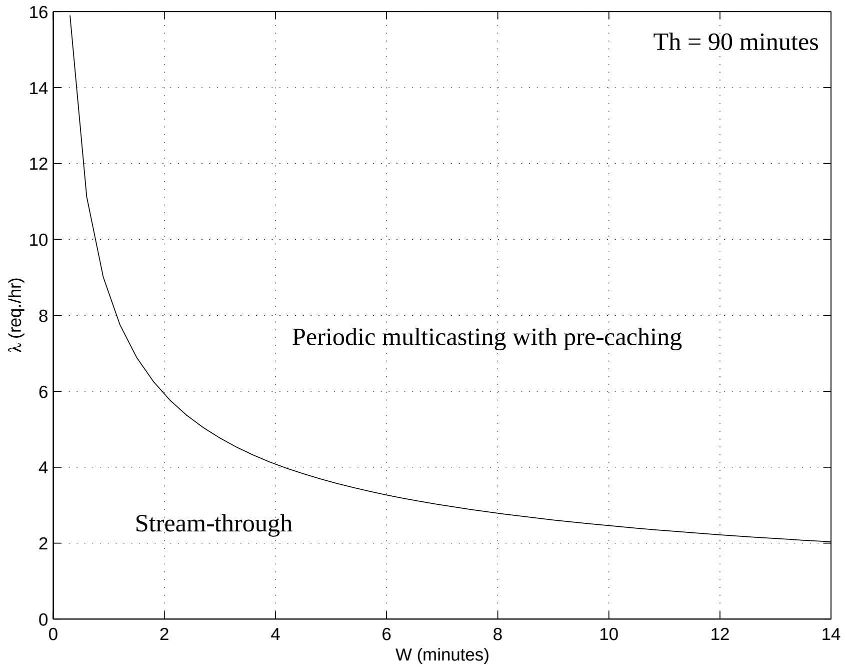


Figure 4.27: Combination of λ and W so that periodic multicasting with pre-caching achieves lower average number of streams

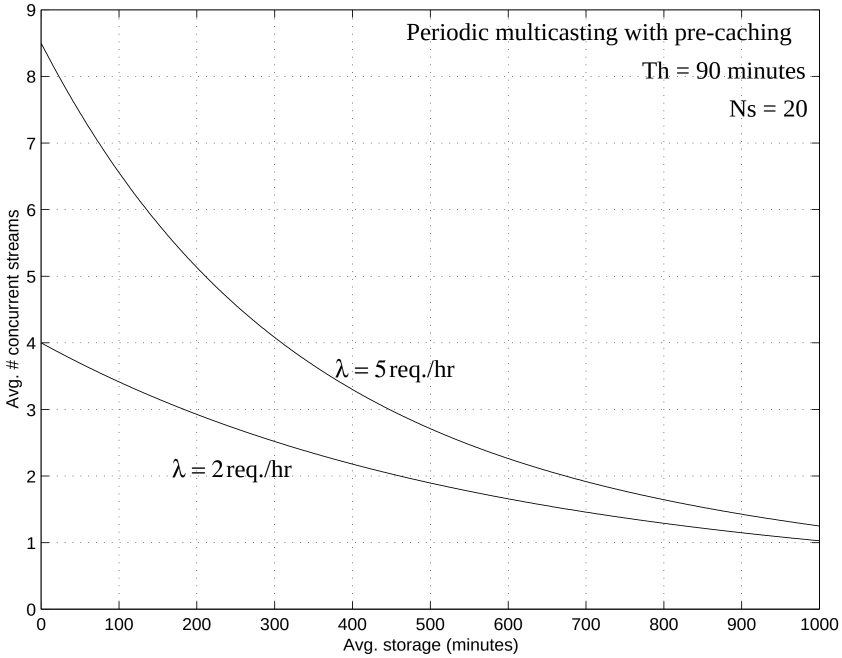


Figure 4.28: $\overline{S}$ versus $\overline{B}$ given λ, with $N_s = 20$

will be increased. We show in Fig. 4.28 the trade-off curve with $N_s = 20$, given λ (with uniform load $\lambda_i = 5/N_s$ req./hr). The trade-off curve is convex, indicating that with linear cost function W does not have to take on extreme in order to minimize cost. We plot in Fig. 4.29 the total cost $\hat{C}$ versus W when $N_s = 20$. There is indeed an optimal window W^* to minimize $\hat{C}$. When γ is relatively high, W^* is low and the optimal cost would not be much different from the stream-through case. When γ decreases, W^* increases and $\hat{C}$ is quite flat at the optimum, indicating that W do not have to be preciously W^*.

We plot in Fig. 4.30 W^* versus γ given that $N_s = 20$ ($\lambda_i = 5/N_s$ req./hr and $T_h = 90$ minutes). When γ is below a certain low value, we have $W^* = \infty$ indicating that the movie should be replicated among the local servers to take advantage of the low storage cost. As γ increases beyond this point, W^* decreases. From the rather linear slope, we can deduce that W^* decreases exponentially with γ. As γ further increases, there is a point at which it is too expensive to do any caching at all ($W^* = 0$), and stream-through operation becomes cheaper.

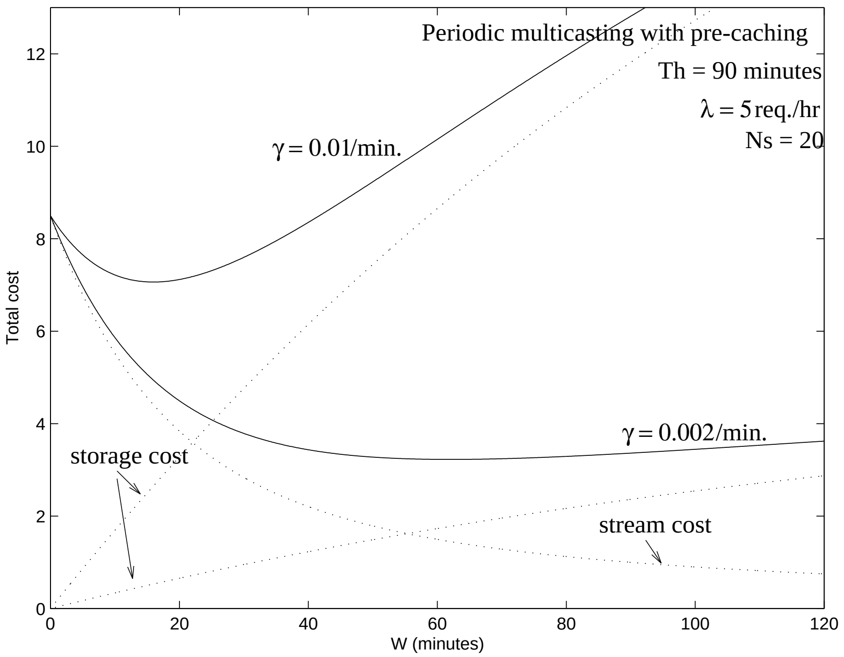


Figure 4.29: $\hat{C}$ and its cost components versus W given γ

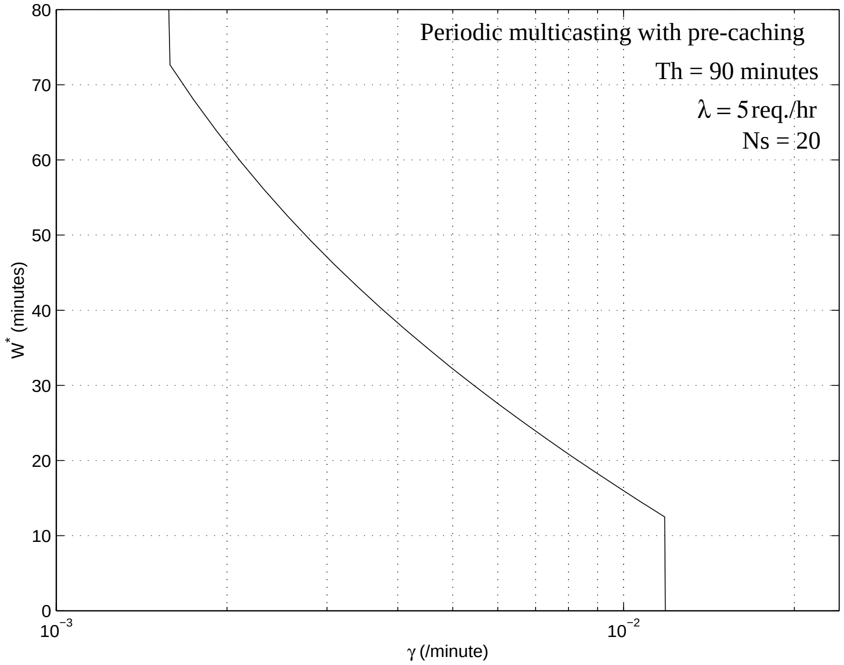


Figure 4.30: W^* versus γ

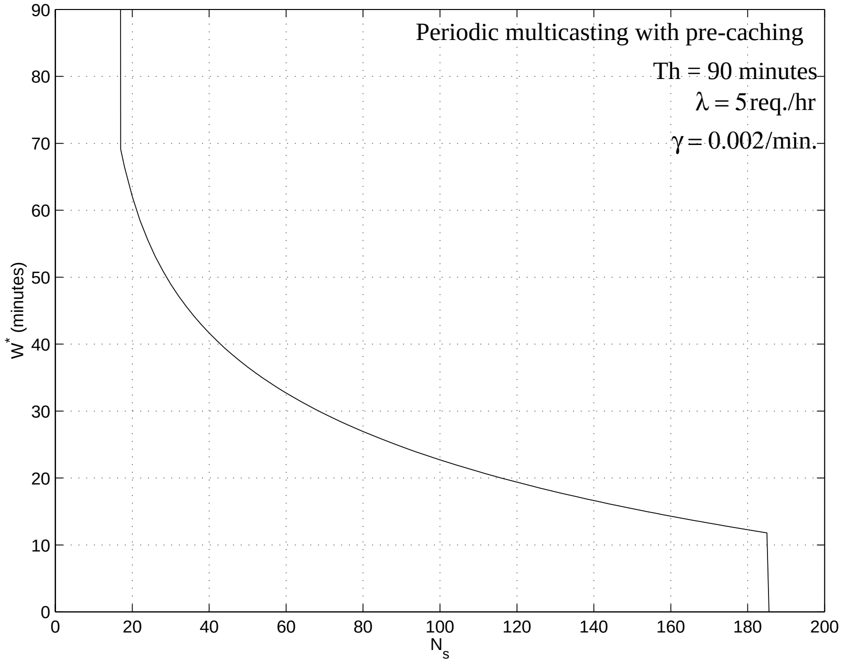


Figure 4.31: W^* versus N_s given γ

We show in Fig. 4.31 W^* as a function of N_s ($\gamma = 0.002$, $\lambda = 5$ req./hr and uniform load). When N_s is below a certain number, the local request rate is quite high and hence the movie should be completely stored. When N_s increases beyond this value, W^* decreases. When N_s further increases, the local request rate for the movie becomes low and the total storage cost increases. A point is reached beyond which the movie should be streamed-through to the users directly.

We show in Fig. 4.32 the optimal W^* as a function of λ given γ (with uniform server load $\lambda_i = \lambda/N_s$). There is a minimum λ below which pre-caching is not worthwhile due to high storage cost. As λ increases beyond this point, W^* increases very sharply and then remains quite constant. As λ further increases, the movie should be stored locally ($W^* = \infty$).

We show in Fig. 4.33 the corresponding $\overline{B}_i$ in each local server and $\overline{S}$ as a function of λ when $W = W^*$. As λ increases, the storage for a movie increases, and jump to T_h when the movie is completely stored. As regards to $\overline{S}$, as λ increases, $\overline{S}$ first increases with slope T_h (stream-through case), and then drops (due to local caching); it remains quite constant

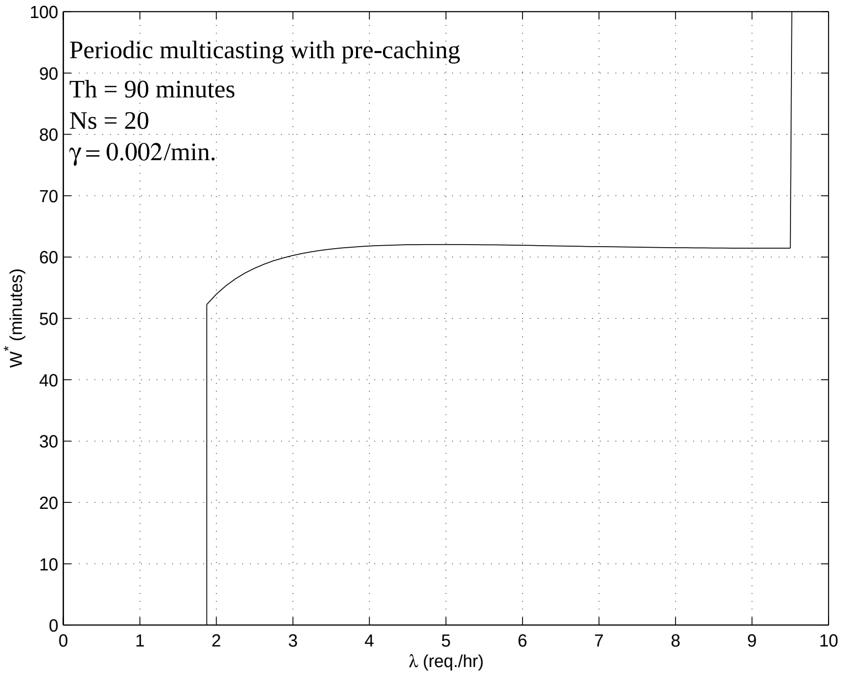


Figure 4.32: W^* versus λ with $\gamma = 0.002$

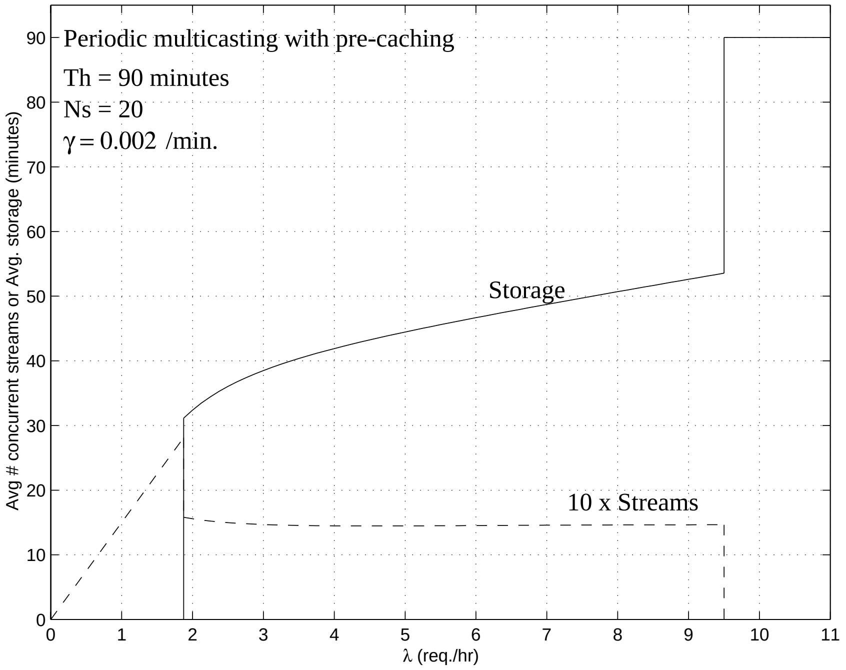


Figure 4.33: $\overline{B}_i$ and $\overline{S}$ versus λ when $W = W^*$

until the point when the movie is completely stored locally, at which $\overline{S}$ dropped to zero.

We show in Fig. 4.34 $\hat{C}^*$ versus λ, with or without using multicast channels (we assume that unicast channels are charged the same as multicast channels here). We see that multicast channels can decrease system cost for a range of λ, and there is slight cost reduction.

We show in Fig. 4.35, 4.36 and 4.37 the W^*, the corresponding $\overline{S}$ and $\overline{B}_i$, and $\hat{C}^*$ versus λ as γ is increases to 0.01/minute. Since the channel cost is relatively less expensive, given a λ, W^* decreases (so that $\overline{B}_i$ decreases) while $\overline{S}$ increases. Since more movie deliveries can take advantage of the multicast channels, multicasting leads to significant cost reduction compared with the unicast case. We therefore see that as channel cost becomes less and less expensive, multicasting will lead to more and more cost advantage over unicasting.

On-demand pre-caching

We now study on-demand pre-caching. We show in Fig. 4.38 $\overline{S}$ versus W given λ. We see similar trend as compared with the periodic multicasting case (Fig. 4.25). However,

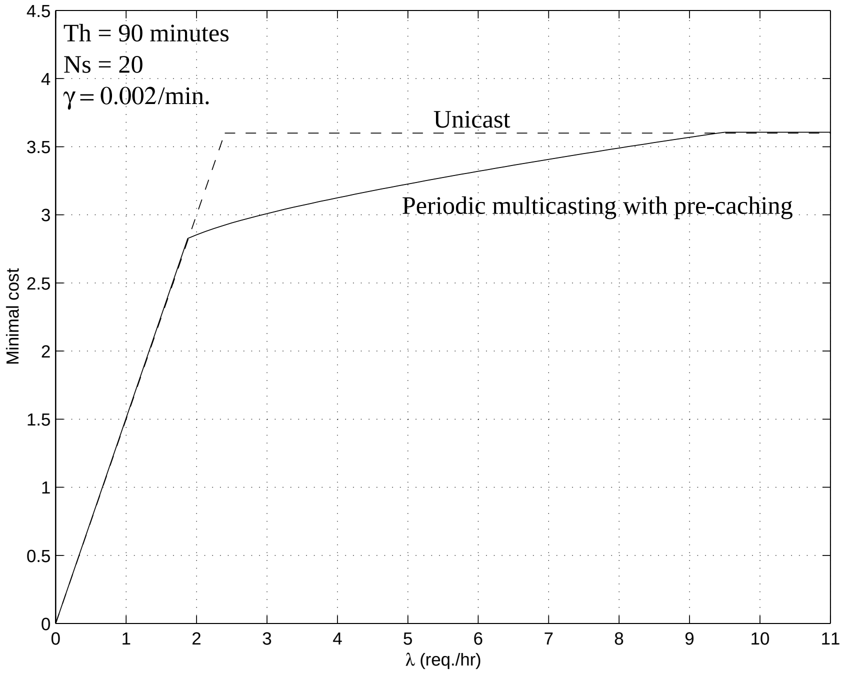


Figure 4.34: $\hat{C}^*$ versus λ, with and without multicasting

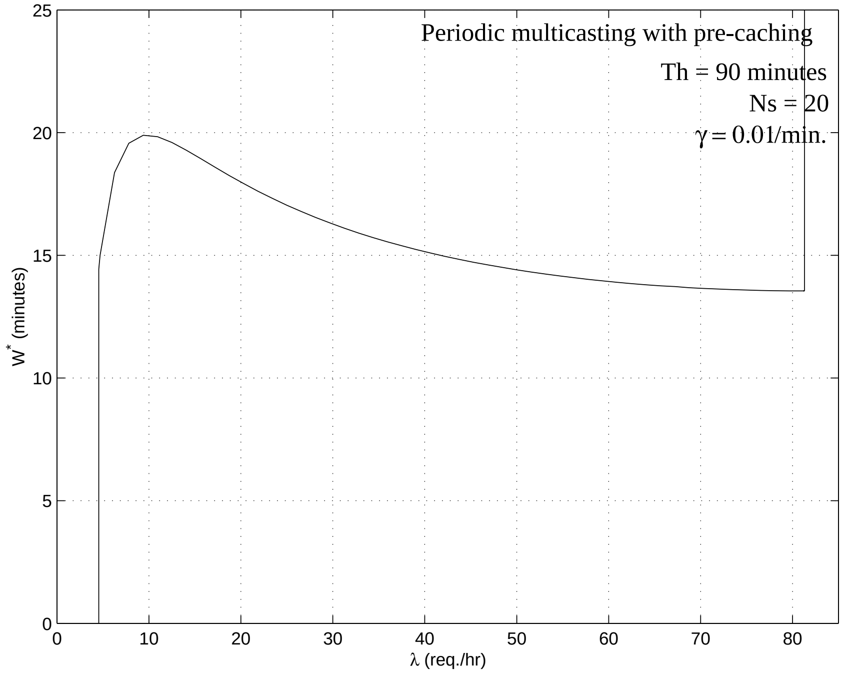


Figure 4.35: W^* versus λ with $\gamma = 0.01$

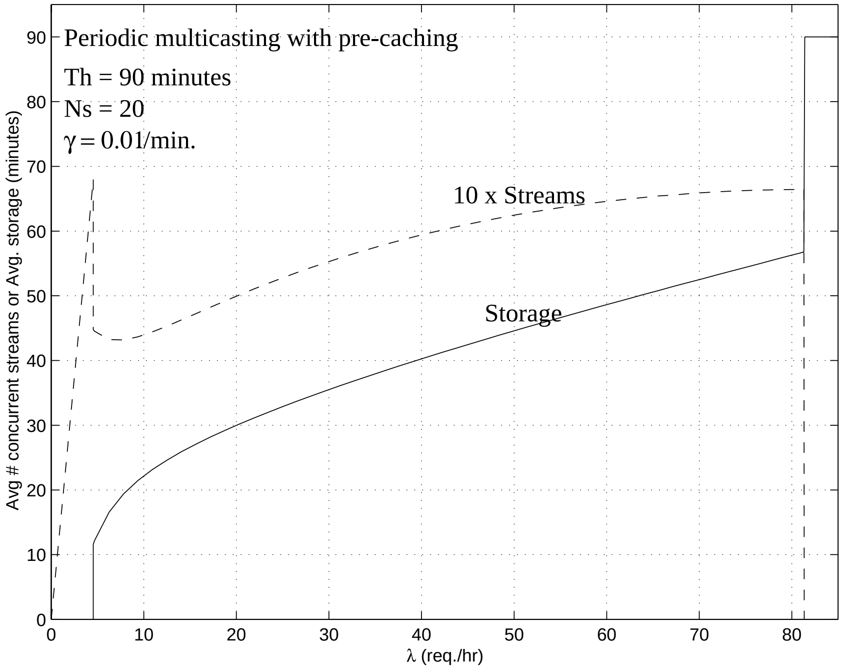


Figure 4.36: $\overline{B}_i$ and $\overline{S}$ versus λ when $W = W^*$

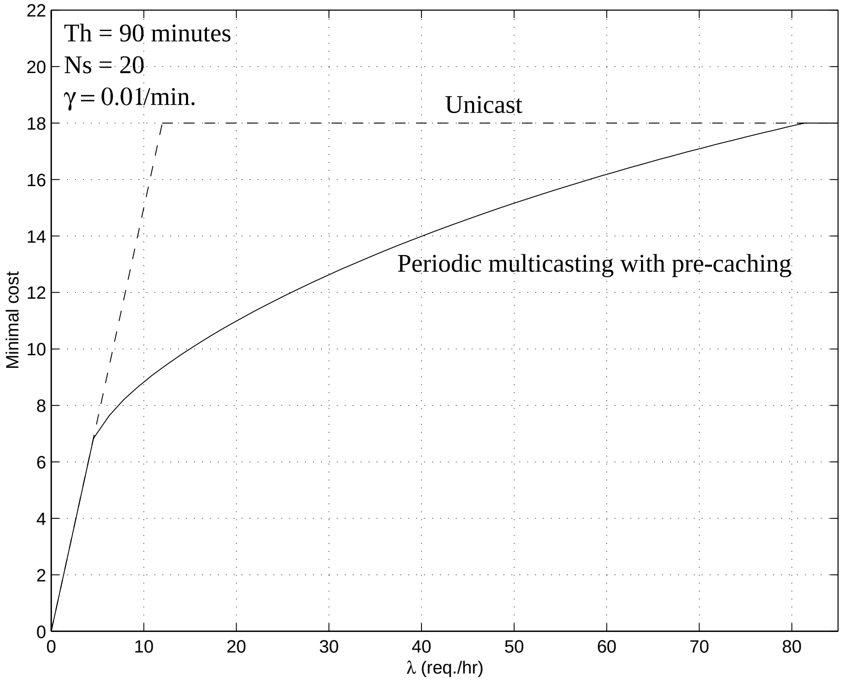


Figure 4.37: $\hat{C}^*$ versus λ, with and without multicasting

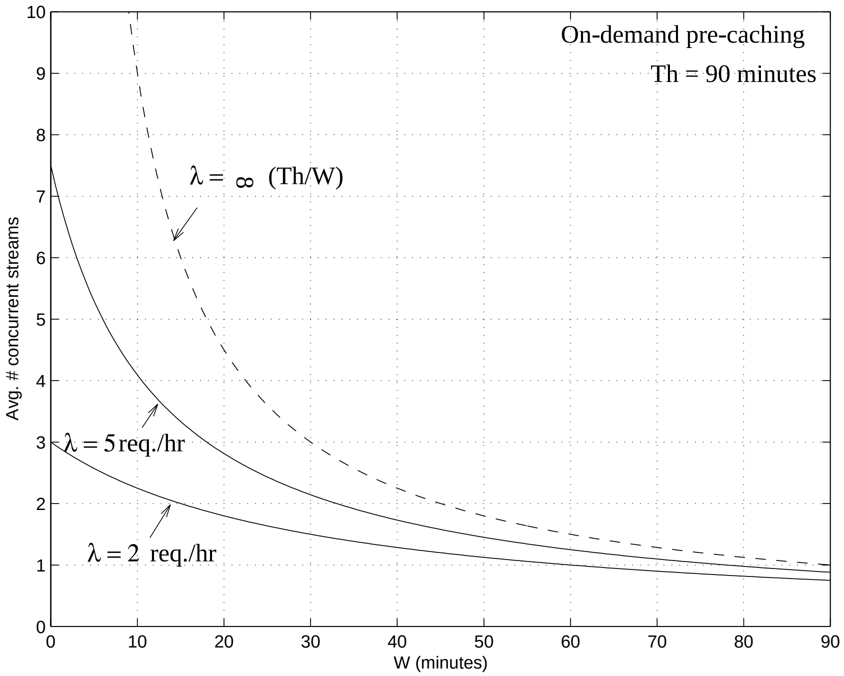


Figure 4.38: $\overline{S}$ versus W given λ for the on-demand pre-caching

on-demand caching achieves lower $\overline{S}$ since it does not need to abort a multicast stream.

We show in Fig. 4.39 the trade-off between $\overline{S}$ and $\overline{B}$ (average total system storage), given λ ($N_s = 20$ and uniform load). We see that there is rather strong trade-off between $\overline{S}$ and $\overline{B}$. It is again worthwhile to compare it with the case of periodic multicasting (Fig. 4.28), and note the similarity between the two graphs. Since the trade-off curve of on-demand pre-caching lie below that of the periodic multicasting case, we expect that, given certain cost functions of storage and channels, on-demand pre-caching has a lower cost.

We show in Figs. 4.40, 4.41 and 4.42 W^*, the corresponding $\overline{S}$ and $\overline{B}_i$, and $\hat{C}^*$ versus λ, for $\gamma = 0.002$/minute ($N_s = 20$ and uniform load). As expected, there is an arrival rate below which $W^* = 0$ (the movie is directly streamed through from the central server), and an arrival rate above which the movie should be completely stored. For the intermediate value of λ, the movie would be partially stored locally. W^* first increases quite sharply, and then remains quite constant before it shoots up to ∞ (stored locally). Compared with the periodic multicasting case, on-demand pre-caching shows similar trend, but at optimum it

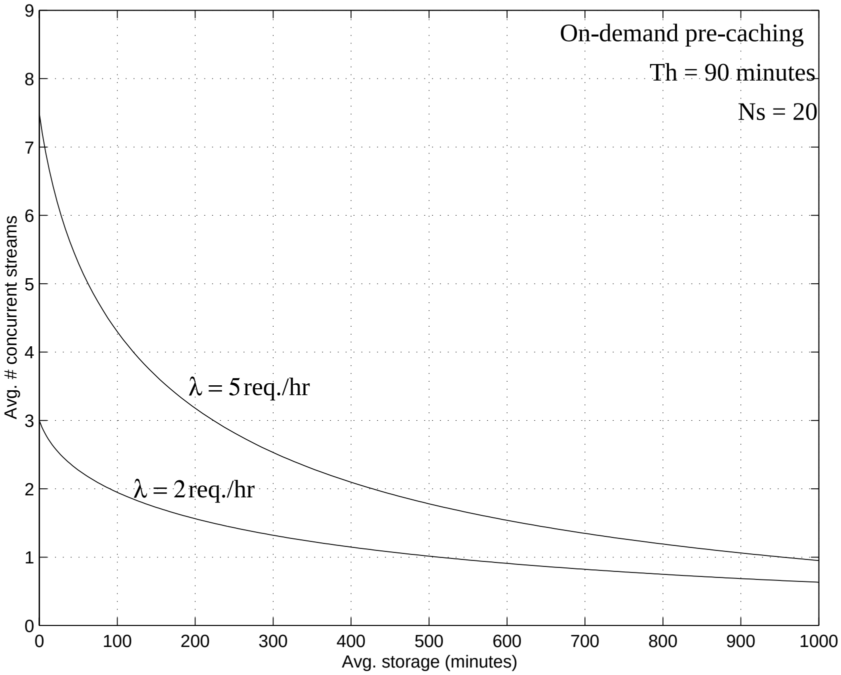


Figure 4.39: $\overline{S}$ versus $\overline{B}$ given λ, with $N_s = 20$

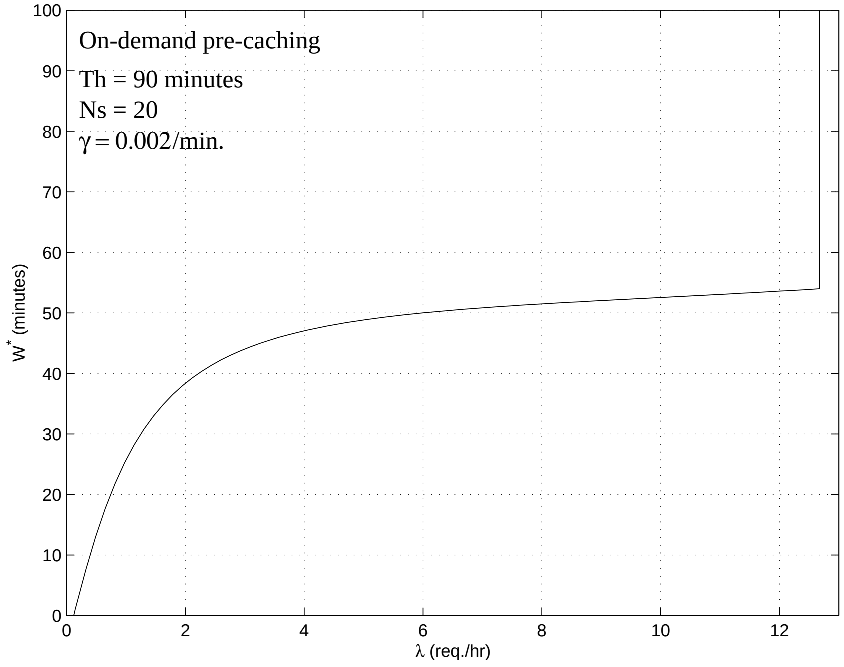


Figure 4.40: W^* versus λ with $\gamma = 0.002$

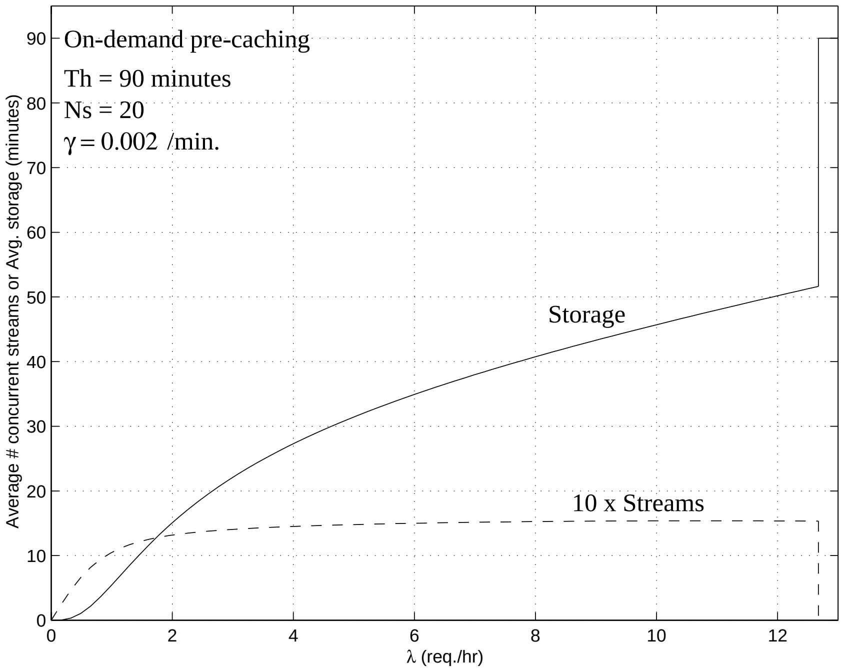


Figure 4.41: $\overline{B}_i$ and $\overline{S}$ versus λ when $W = W^*$

achieves both lower storage and lower streams, and hence more significant cost reduction compared with the unicast case. It is therefore a better choice.

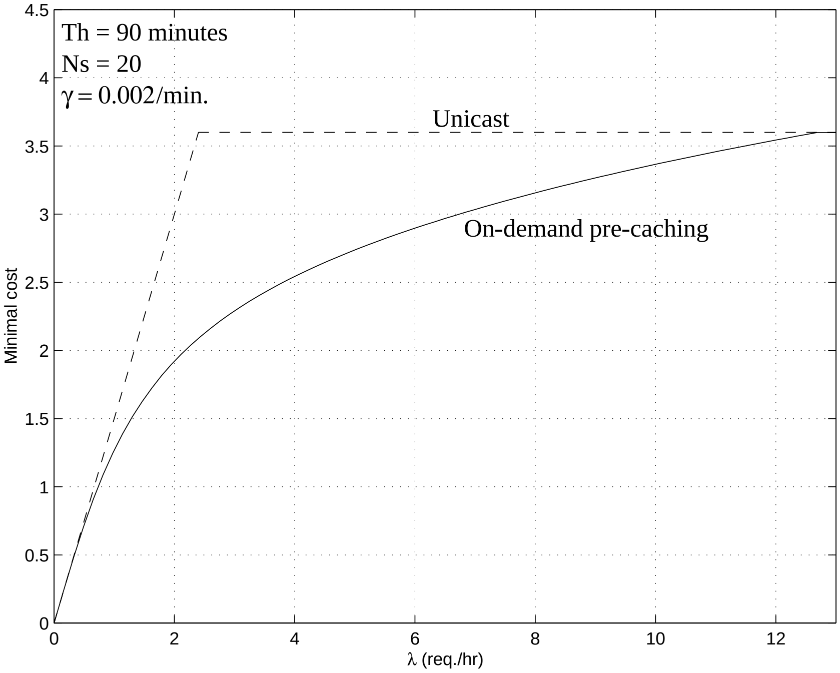


Figure 4.42: $\hat{C}^*$ versus λ, with and without multicasting

Latching

In this section, we consider latching. We first consider the periodic multicasting case, and then the on-demand case. First we study the influence of N_s with $R = 1$, and then study the influence of R. Since the performance of the on-demand latching is better, we will then focus in it in the later part of this section.

Periodic multicasting with latching

In this section, we study periodic multicast with latching, for $R = 1$. We show in Fig. 4.43 $\overline{S}$ versus W given N_s, with $\lambda = 5$ req./hr and $\lambda_i = \lambda/N_s$ for all i (uniform load). $\overline{S}$ first decreases with W, and then for high N_s increases again. Most of the bandwidth saving is achieve for low W ($W \approx$ 10–30 minutes). As N_s increases, $\overline{S}$ also increases since it is less likely for an arrival to share a cache. Note that for high N_s, $\overline{S}$ actually increases beyond a certain W due to long transient streams. Therefore the system will be likely designed with W less than such point. In this range, there is no much difference in $\overline{S}$ for different N_s.

In Fig. 4.44 we show the average total storage $\overline{B}$ versus W given N_s, again with $\lambda_i = \lambda/N_s$ for all i ($\lambda = 5$ req./hr and $T_h = 90$ minutes). As W increases, $\overline{B}$ approaches $N_s T_h$. Note that for large N_s, $\overline{B}$ increases quite linearly for a large range of W (with the initial slope being λT_h).

We show in Fig. 4.45 the trade-off between $\overline{S}$ and $\overline{B}$, given λ ($N_s = 10$ and $R = 1$). $\overline{S}$ first decreases with $\overline{B}$; however, as $\overline{B}$ further increases, the large W leads to more and longer transient streams and hence $\overline{S}$ increases. In minimizing system cost, the optimal point is hence likely below the turning point in the trade-off line.

On-demand latching

We now study on-demand latching. We have done simulation to study its performance. As evident in Fig. 4.43, the performance with $N_s = \infty$ is not much different from finite N_s (if the system is designed properly) and is also the pessimistic case (in terms of $\hat{C}$, $\overline{S}$ and $\overline{B}$); therefore, the case will be considered here.

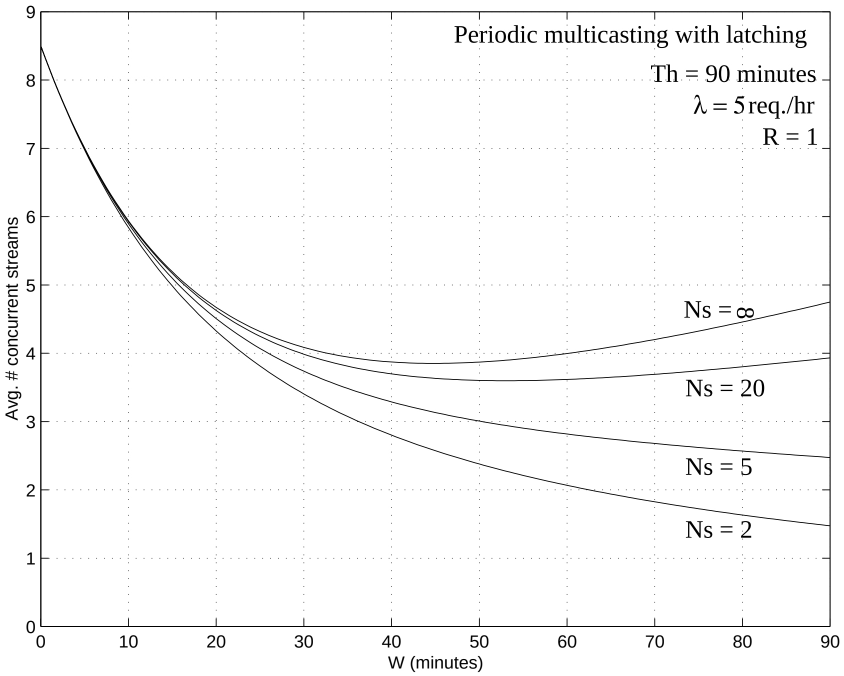


Figure 4.43: $\overline{S}$ versus W given N_s

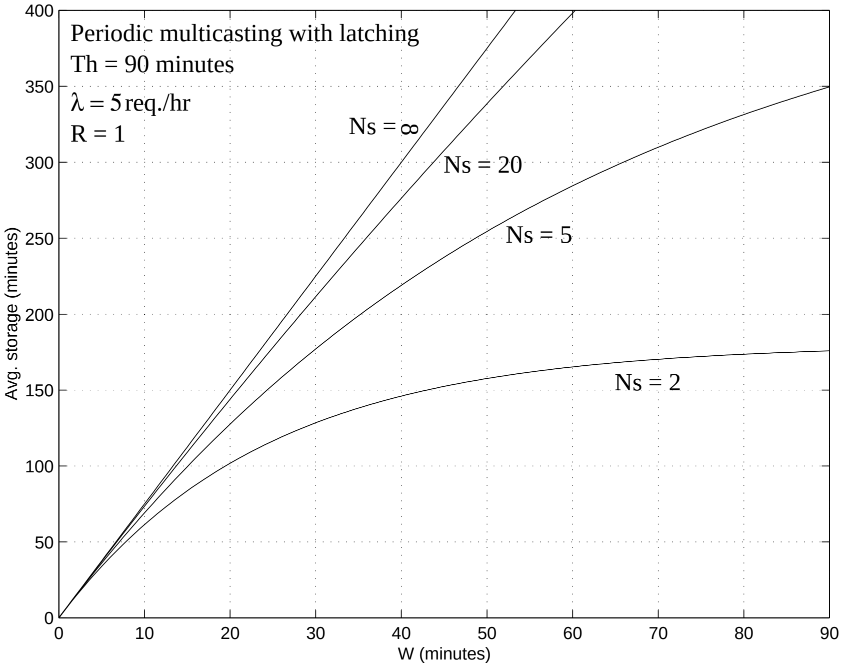


Figure 4.44: $\overline{B}$ versus W given N_s

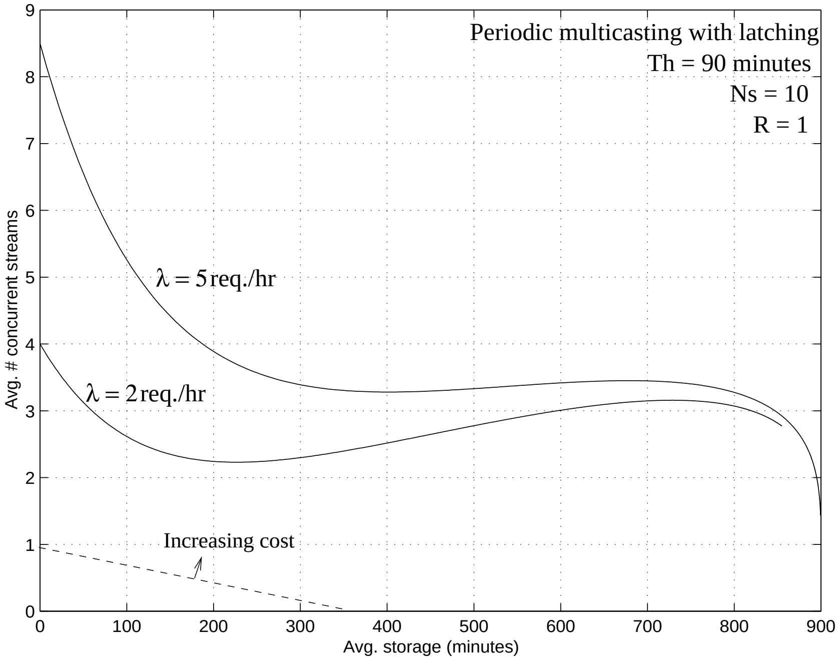


Figure 4.45: $\overline{S}$ versus $\overline{B}$ given λ, for $N_s = 10$ and $R = 1$

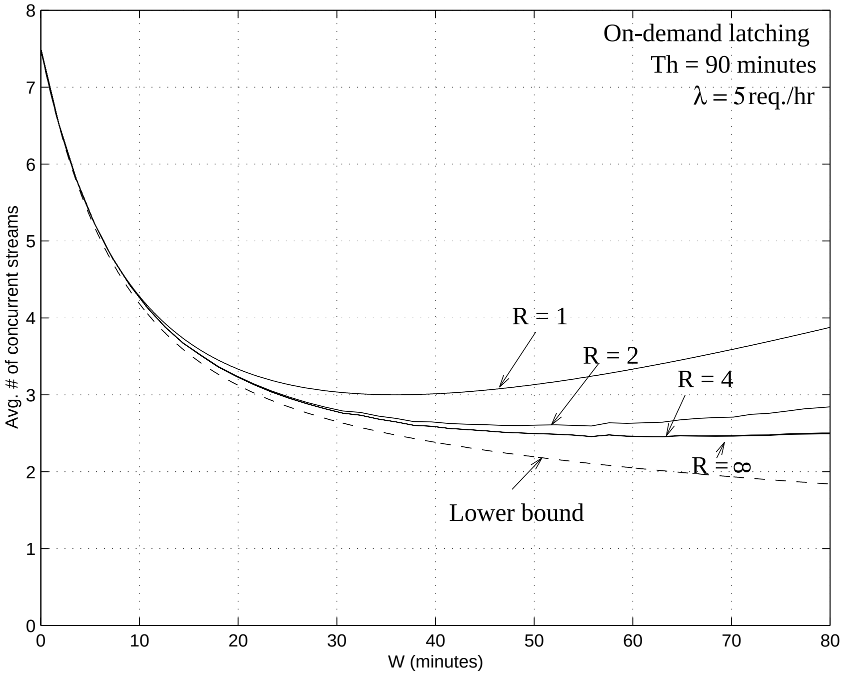


Figure 4.46: $\overline{S}$ versus W given R ($\lambda = 50$ req./hr, $T_h = 90$ minutes)

We show in Fig. 4.46 $\overline{S}$ versus W for latching, given R ($\lambda = 50$ req./hr, $T_h = 90$ minutes). As W increases, more on-going streams can be latched on, and hence $\overline{S}$ decreases initially, first quite sharply and then very slowly. For low R, however, as W increases beyond a value $\hat{W}$, $\overline{S}$ increases again due to more transient streams with longer length (Note that the case of $R = 1$ in this figure with the case of the periodic multicasting with latching in Fig. 4.43 with $N_s = \infty$ differs by at most one stream, showing that the two schemes are very similar). As R increases, $\overline{S}$ decreases (as more on-going streams can be latched). However, there is not much decrease in $\overline{S}$ as R increases beyond 3 or 4.

We show in Fig. 4.47 $\hat{W}$ versus λ for for $R = 1$ ($T_h = 90$ minutes). $\hat{W}$ first decreases quite sharply with λ and then more slowly.

We show in Fig. 4.48 $\overline{S}$ versus λ given W, for $R = 2$. As λ increases, $\overline{S}$ increases, first quite linearly (because of no latching) and then slowly (due to latching). Compared with the stream-through case, there is substantial saving in $\overline{S}$ with latching.

We show in Fig. 4.49 $\overline{S}$ versus R, given W ($\lambda = 5$ req./hr). We see that there is a marked decrease in $\overline{S}$ when R is increased from 0 (corresponding to the stream-through

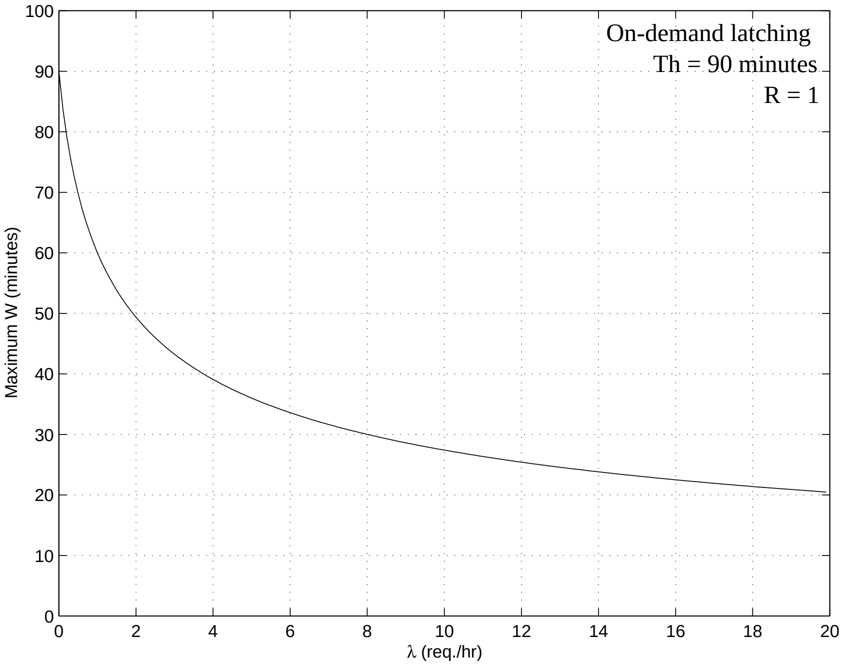


Figure 4.47: $\hat{W}$ versus λ given T_h $(R = 1)$

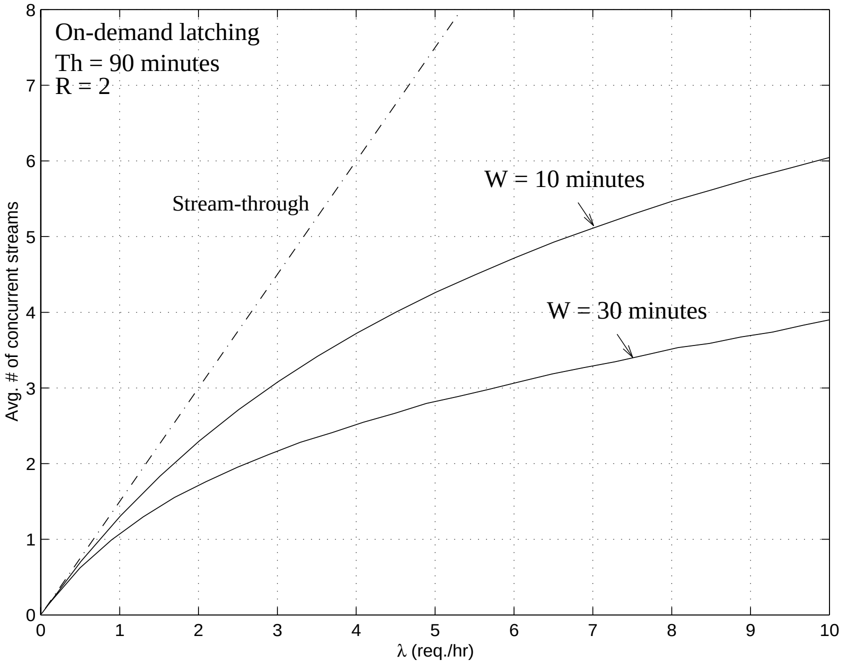


Figure 4.48: $\overline{S}$ versus λ given W $(R = 2,\ T_h = 90$ minutes$)$

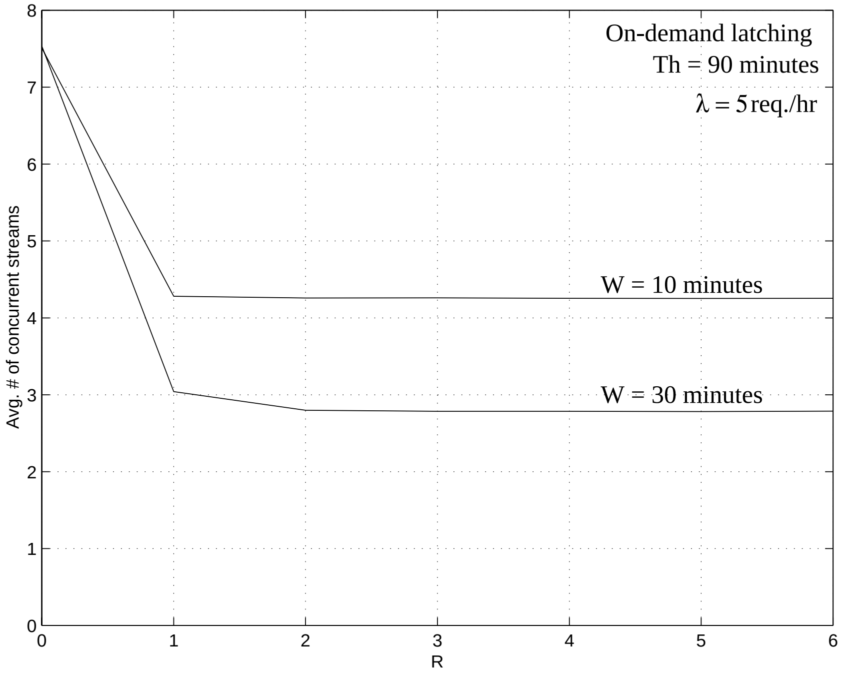


Figure 4.49: $\overline{S}$ versus R given W

case) to 1; afterwards the decrease is very little. Therefore in reality, there is no need to latch on more than 1 or 2 streams.

Note that the buffering cost can be completely eliminated if users are willing to wait for their video: users arriving within a time W can be grouped (i.e., batched) together and served with a single multicast stream. Hence a maximum user delay of W is incurred. If the first user in a batch always experiences a delay of W, it can be easily shown that the average number of streams used is $\lambda T_h/(1 + \lambda W)$ (the auto-gated scheduling in chapter 3). In latching, therefore, buffering and streams are used to trade off user delay. Fig. 4.50 shows the number of additional streams compared to $\lambda T_h/(1 + \lambda W)$ so as to provide an on-demand video services, given R ($\lambda = 50$ req./hr, $T_h = 90$ minutes). Only very little additional streams are required compared with the batched case.

Since $\overline{B} \approx \lambda T_h W$, the trade-off between $\overline{S}$ and $\overline{B}$ follows the same shape as $\overline{S}$ versus W. In Fig. 4.51 we show $\hat{C}$ versus W given γ ($R = 1$, $\lambda = 50$ and T_h=90 minutes). We see that there is an optimal W^* to achieve minimum cost $\hat{C}^*$. As γ increases, W^* decreases, $\hat{C}^*$ becomes sharper and the total cost increases.

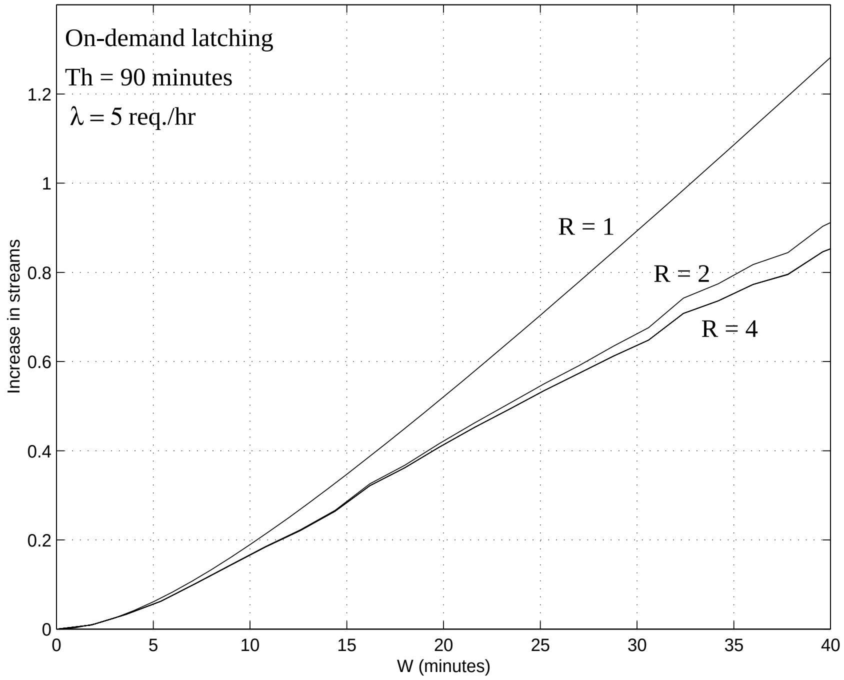


Figure 4.50: Increase in $\overline{S}$ in latching as compared with $\lambda T_h/(1+\lambda W)$ versus W, given R

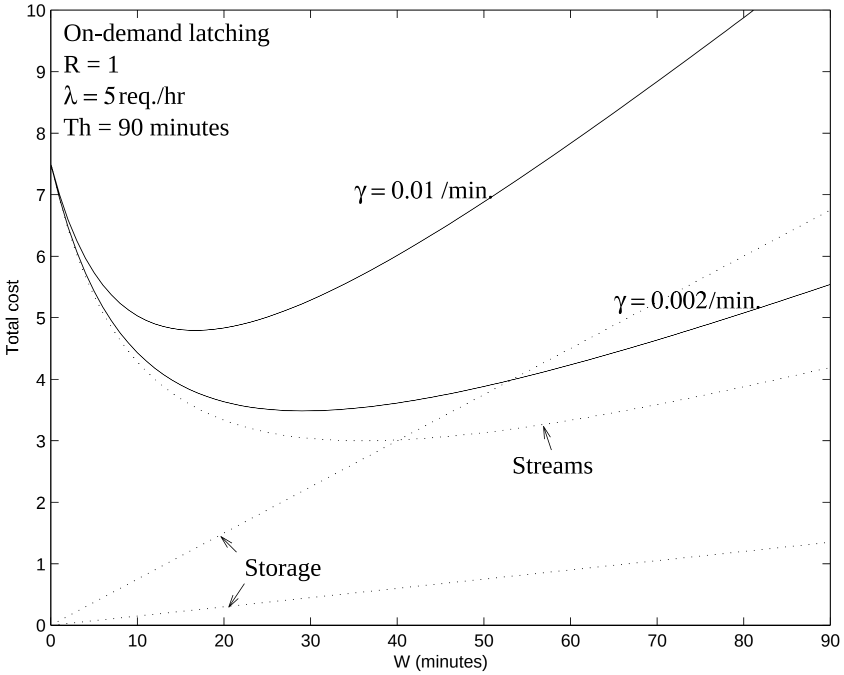


Figure 4.51: $\hat{C}$ versus W given γ

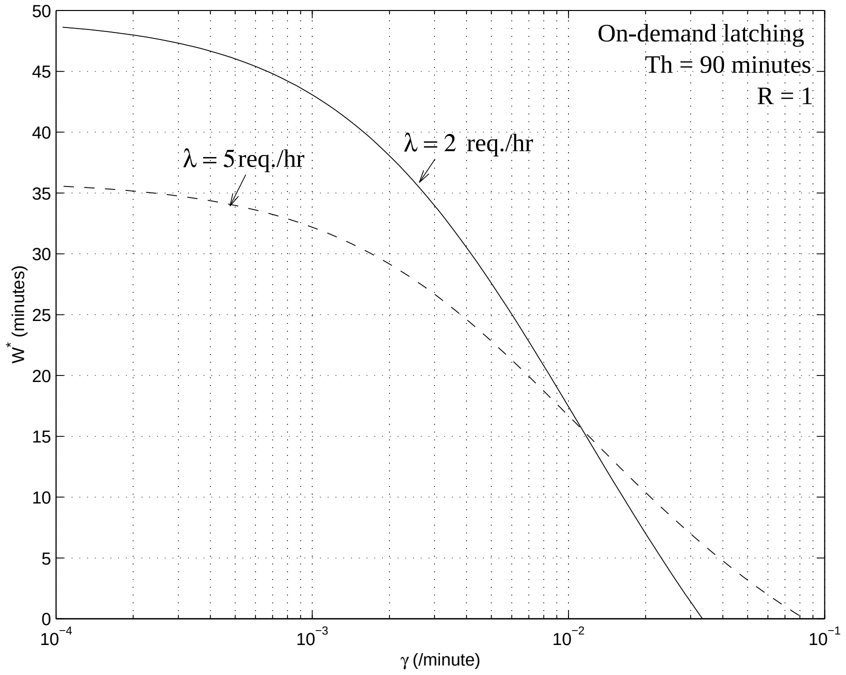


Figure 4.52: W^* versus γ given λ

We show in Fig. 4.52 W^* versus γ given λ ($R = 1$). As γ increases, W^* first stay pretty constant at $\hat{W}$, and then drops sharply. There is a γ beyond which $W^* = 0$, showing that the storage cost is too high and movie should be streamed through from the repository. Such γ decreases as λ decreases, showing that unpopular movie is more likely not to be cached in the local servers.

We show in Fig. 4.50 W^* versus λ (($R = 1$ and $\gamma = 0.002$. There is a minimum value of λ (at $W = 0$) below which the movie should be streamed through from the central repository; after which W^* first increases very sharply and then decreases very gradually with λ.

We show in Fig. 4.54 the $\overline{S}$ and the total storage at $W = W^*$. We see that both $\overline{S}$ and $\overline{B}$ increases with λ. We show in Fig. 4.55 $\hat{C}^*$ versus λ. As expected, the cost increases with λ (since both $\overline{S}$ and $\overline{B}$ increases with λ). There is a significant cost advantage over the stream-through case. For finite N_s, since the cost is pessimistic, we expect lower cost (by a little amount), and the movie will be completely stored (i.e., $W^* = \infty$) at λ when the cost curve intercepts with the horizontal line $N_s\gamma T_h$. It is also worthwhile to note the

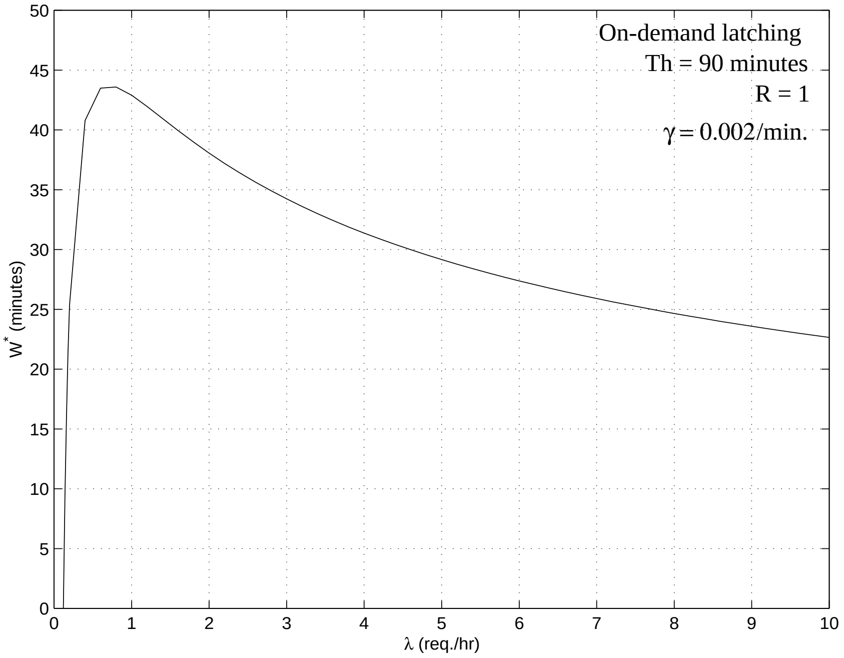


Figure 4.53: W^* versus λ given γ

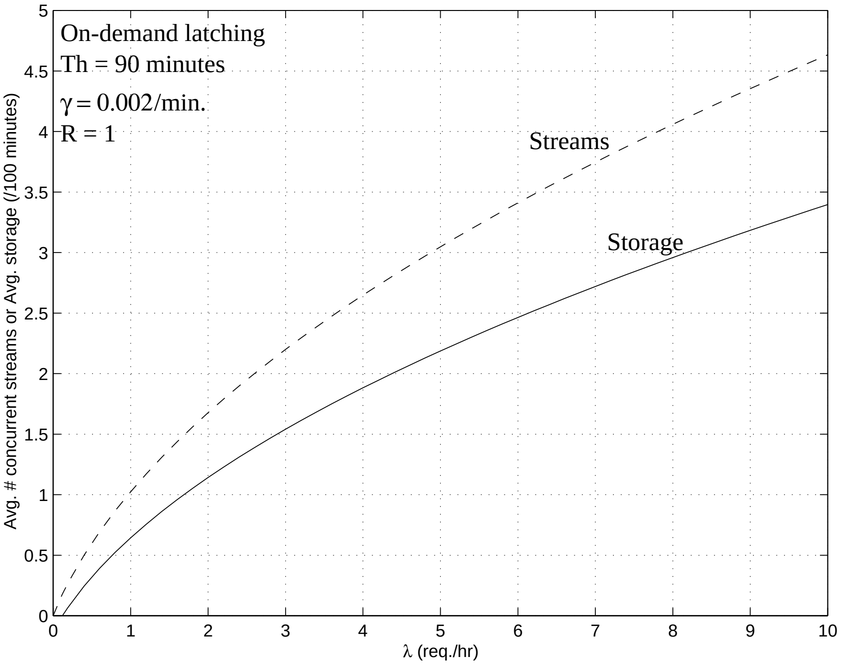


Figure 4.54: $\overline{B}$ and $\overline{S}$ versus λ for $W = W^*$

remarkable resemblance in the cost of latching with that of pre-caching (Fig. 4.42), showing that the simpler pre-caching scheme may be preferred in reality.

4.5.3 Communicating servers

Chaining

Note that $\overline{S}$ in chaining is the same as that of the regenerative-lifetime case (ref. Figs. 4.21 and 4.22), while $\overline{B}$ is linear in W. Since $\overline{B} = \lambda T_h W$, the trade-off between $\overline{S}$ and $\overline{B}$ follows the same shape as $\overline{S}$ versus W in that figure. We hence will focus in this section its cost optimization (we consider $N_s = \infty$ since it is a pessimistic case). We show in Fig. 4.56 $\hat{C}$ versus W given γ. There is a clear W^* at which $\hat{C}$ is minimized. As γ decreases, W^* decreases and the minimum becomes less marked. We show in Fig. 4.57 W^* versus γ, given λ. W^* decreases rather exponentially with γ at the beginning and more slowly at the end. There is a γ beyond which stream-through operation becomes more cost-effective.

We plot in Fig. 4.58 $\hat{C}^*$ versus γ, when $W = W^*$, given λ. $\hat{C}^*$ increases with γ, first quite

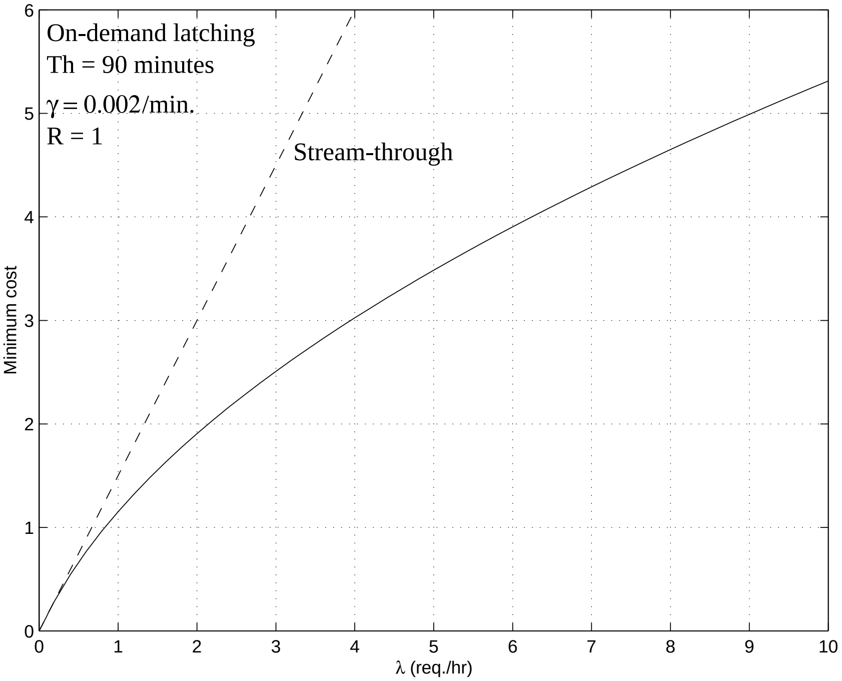


Figure 4.55: $\hat{C}^*$ versus λ

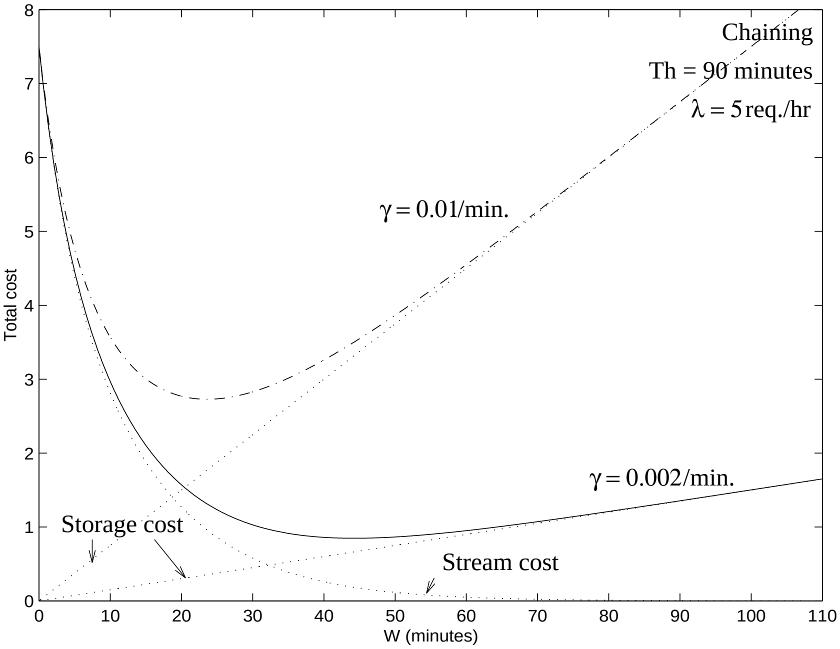


Figure 4.56: $\hat{C}$ versus W given γ

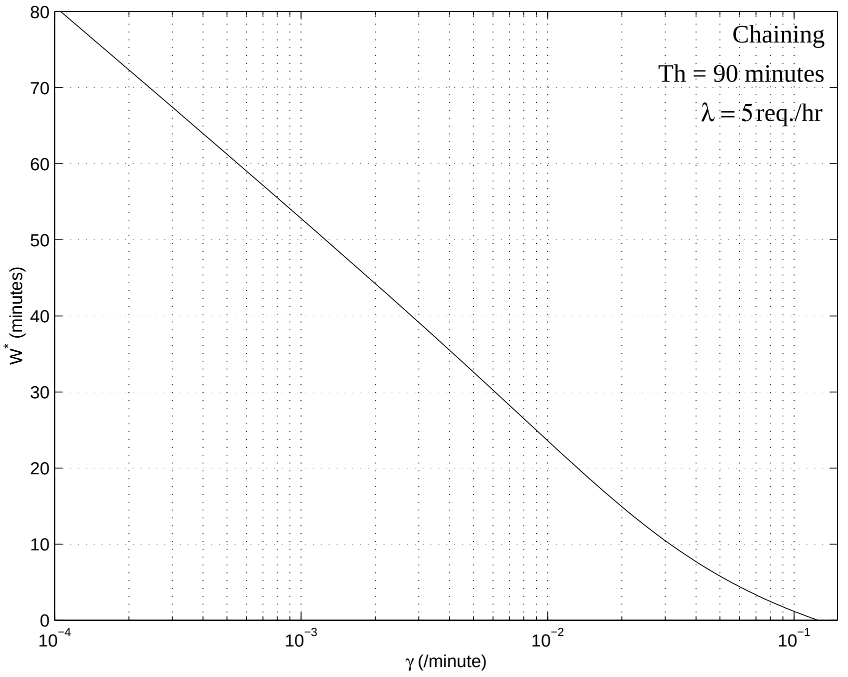


Figure 4.57: W^* as a function of γ given λ

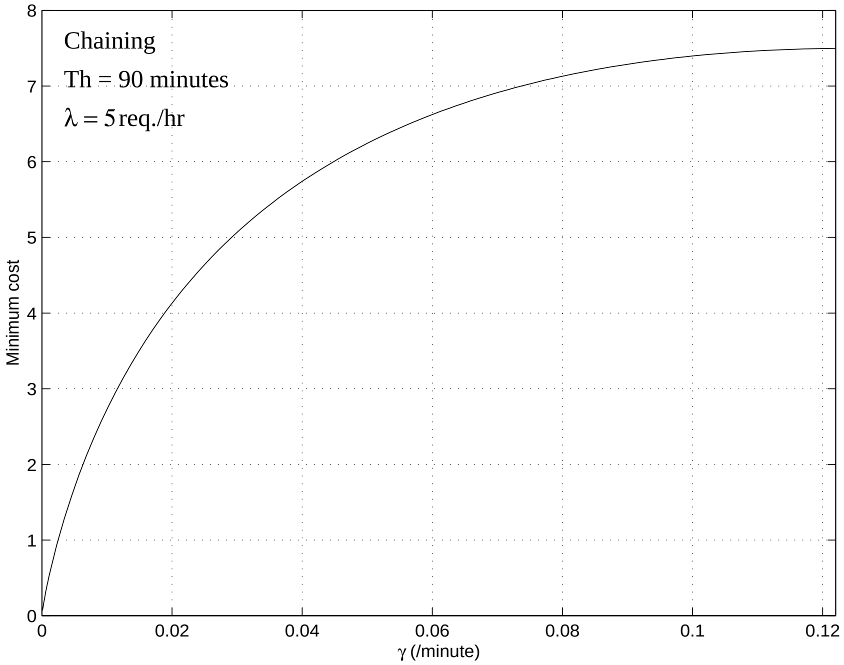


Figure 4.58: $\hat{C}^*$ versus γ given λ

quickly then more slowly, and reaches λT_h as $\gamma \to \infty$ (i.e., $W^* = 0$, the stream-through operation).

We plot in Fig. 4.59 W^* versus λ given γ. We see that there is a request rate below which stream-through from the repository should be used. W^* first increases very sharply with λ and then decreases more slowly. We show in Fig. 4.60 the corresponding $\overline{S}$ and total storage $\overline{B}$. We see that while storage keeps increasing, $\overline{S}$ remains quite constant.

We show in Fig. 4.61 $\hat{C}^*$ versus λ. As λ increases, $\hat{C}^*$ also increases, first quite fast and then more slowly. We see that chaining achieves substantial cost saving compared with the stream-through case. For finite N_s, $\hat{C}^*$ follows the same trend (since we would not expect much difference), but W^* takes on ∞ at λ for which the cost line intercept the horizontal line $N_s\gamma T_h$ (storing the movie completely).

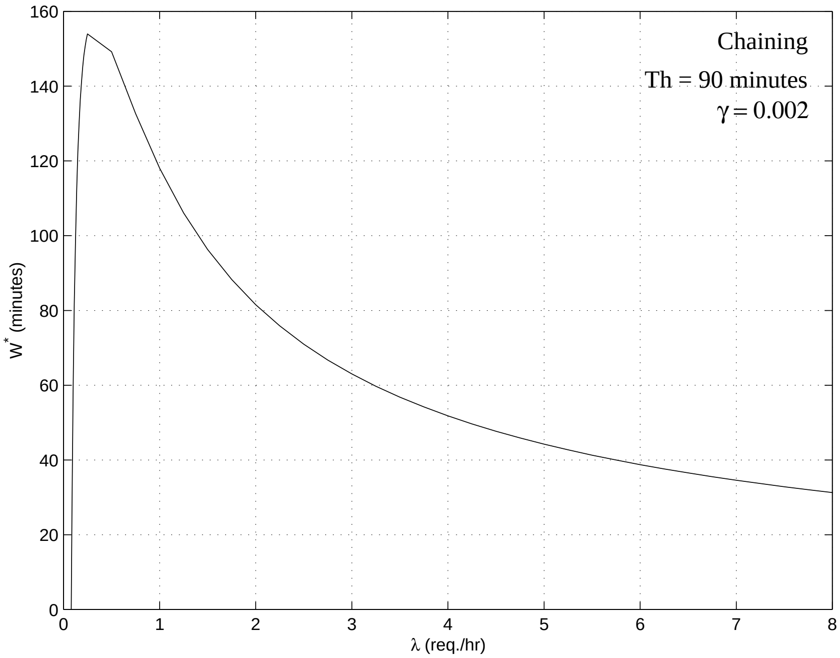


Figure 4.59: W^* versus λ for chaining

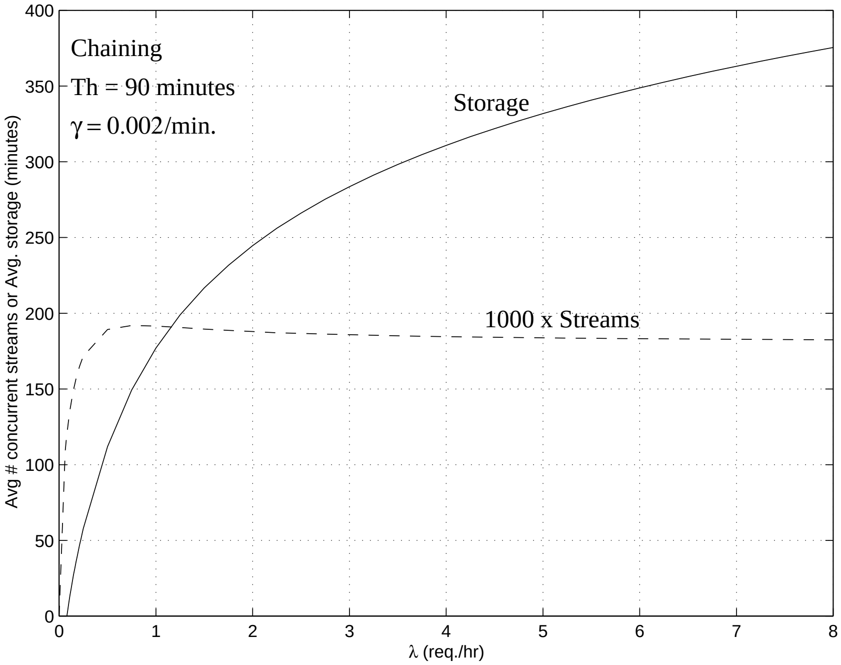


Figure 4.60: $\overline{B}$ and $\overline{S}$ versus λ when $W = W^*$

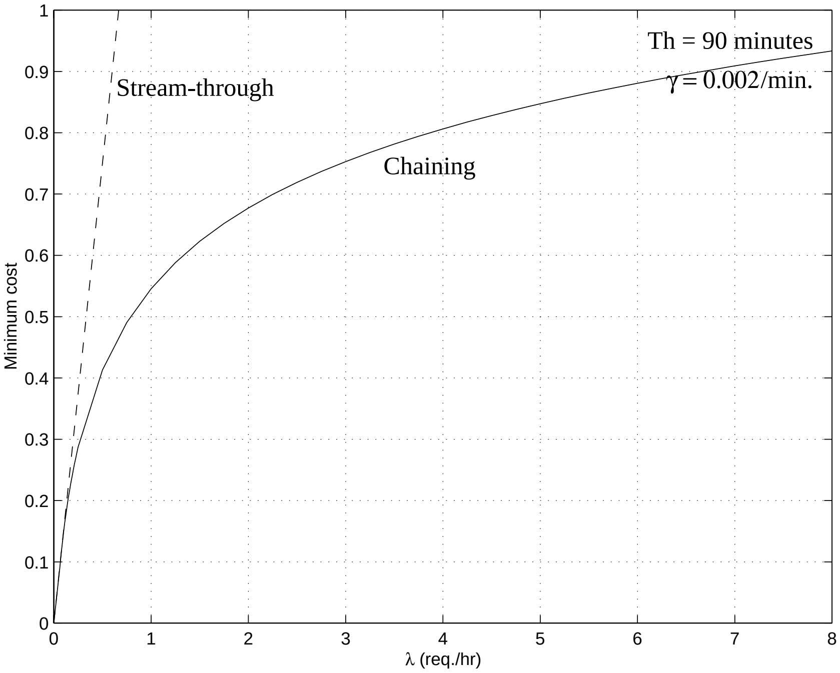


Figure 4.61: $\hat{C}^*$ versus λ

4.6 Cost Comparison Between A Batching System And A Distributed Servers Architecture

Consider a system in which a central repository is used to served all the requests. To increase the streaming capacity, request batching can be used. If users can tolerate a maximum of D_{max} minutes beyond which they very likely leave the system, the batching period should be no more than D_{max}. We may consider a window-based batching scheme as follows: the first user after a movie showing starts a batching window of size $W = D_{max}$, at the end of which the movie is multicast to all the requests arriving within the window. It can be easily shown that, given that the request rate for a movie is λ, the average number of streams required, and therefore the normalized total system cost, is,

$$\overline{S} = \frac{\lambda T_h}{1 + \lambda D_{max}}. \tag{4.26}$$

Let's now consider a distributed servers architecture with lifetime caching. Let N_s be the number of servers in the system, with the arrival rate of server i for a movie being λ_i $(\sum_{i=1}^{N_s} \lambda_i = \lambda)$. Then the normalized minimum cost is given by,

$$\hat{C}^* = \sum_{i=1}^{N_s} \left[\gamma T_h u(\lambda_i - \gamma) + (1 - u(\lambda_i - \gamma))\lambda_i T_h\right], \tag{4.27}$$

where $u(x)$ is the unit-step function of which $u(x) = 1$ for $x \geq 0$ and $u(x) = 0$ otherwise.

We show in Fig. 4.62 the minimum cost for the distributed servers system and the batching system versus λ given γ, with $D_{max} = 6$ minutes and $\lambda_i = \lambda/N_s$ (we have considered that the costs for unicast channels and multicast channels are the same). For values of $N_s\gamma$ higher than a certain value $(= 1/D_{max})$, distributed servers architecture definitely has a higher cost than a system with request batching (hence trading off system cost with low user delay). However, with low $N_s\gamma$, by taking advantage of the low storage cost, distributed servers architecture is able to achieve lower cost than request batching, while at the same time offer zero start-up delay to the users! Note that the two lines cross at λ', where

$$\lambda' = \frac{N_s\gamma}{1 - N_s\gamma D_{max}}. \tag{4.28}$$

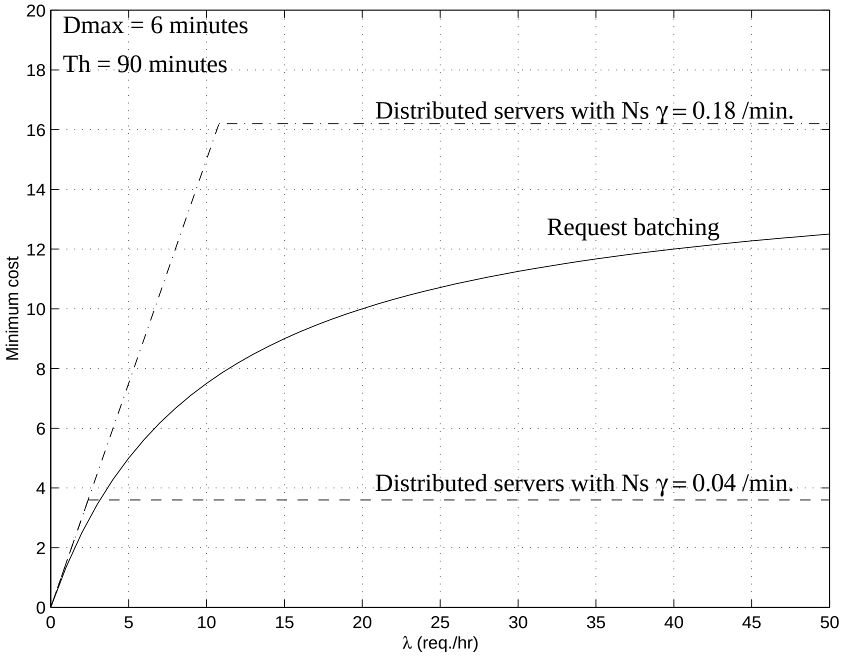


Figure 4.62: $\hat{C}^*$ versus λ, with request batching and distributed servers architecture

We see from the figure that to minimize system cost, when $\lambda < \lambda^{'}$, we should use request batching, and when $\lambda > \lambda^{'}$, we should store the movie locally.

We show in Fig. 4.63 $\lambda^{'}$ versus $N_s\gamma$ ($D_{max} = 6$ minutes). When $N_s\gamma$ is high, movies are less likely stored locally and request batching is generally used. However, when $N_s\gamma$ is low, many movies would be stored locally to take advantage of the low-cost storage.

As an illustrative example in how much cost a distributed system can save, let's consider a system with 80/20 movie popularity as shown in Fig. 4.1 (the total request rate in the system Λ is 4000 req./hr), with the column corresponding to $\beta = 0.03$/minute in Tab. 4.1. We consider six systems: 1) an on-demand video system in which the central server is solely used to serve all the requests (VOD); 2) a system in which $N_s = 20$ local servers storing all the movies are used to server all the request, with each server serving 200 req./hr; 3) a system with a central server using request batching to serve all the requests ($D_{max} = 6$ minutes); 4) a system with a central repository and $N_s = 20$ (uniformly loaded) local servers, with 20% of the most popular movies stored in a local server; 5) a system same in (4), but using lifetime caching with optimal caching strategies (i.e., either we completely

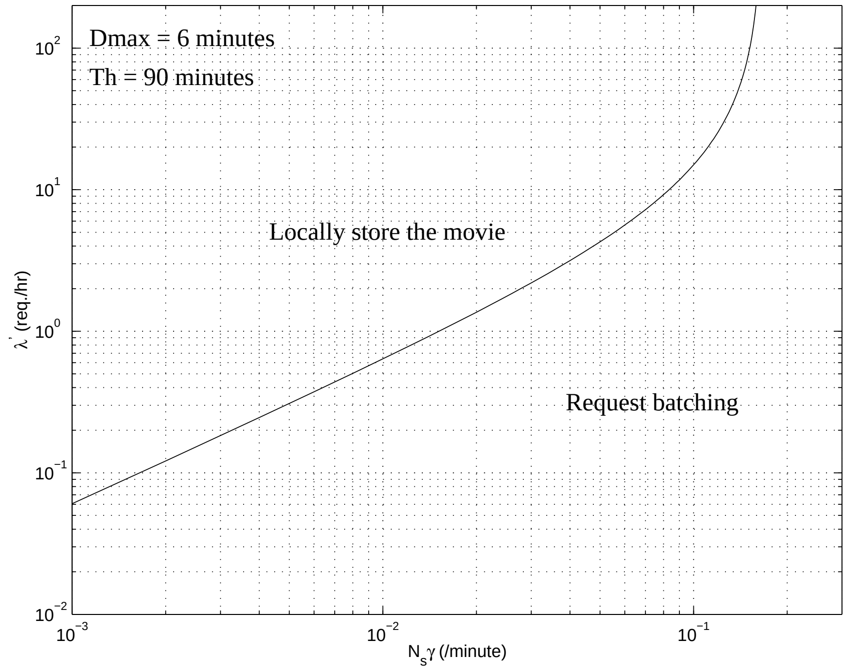


Figure 4.63: λ' versus $N_s\gamma$

Table 4.2: The total cost (in \$/minute), cost per user and the request rate in the central repository for different systems ($\gamma = 0.002$, $N_s = 20$, $\Lambda = 4000$ req./hr for 6 hours a day, number of movies = 500, 80/20 video popularity, $D_{max} = 6$ minutes, and $T_h = 90$ minutes)

	Cost (Cost per user)	Repository request rate
	\$/minute (\$)	req./hr
VOD	180 (2.7)	4000
All stored locally	54 (0.81)	0
Request batching	56.05 (0.84)	4000
20% local movies	46.75 (0.7)	800
Optimal caching[a]	28.73 (0.43)	148.79
Combined scheme[b]	27.85 (0.42)	196.02

[a] Each local server stores 204 most popular movies at optimum.

[b] Each local server stores 187 most popular movies at optimum.

store a movie or not at all depending on its request rate); 6) a system same in (4), but with caching and batching combined — movies are stored in the local servers if $\lambda \geq \lambda'$, and request batching is used in the central server if $\lambda < \lambda'$ ($D_{max} = 6$ minutes). We show in Tab. 4.2 the system cost in \$/minute (along with the cost per user, i.e., the minimum charge to a user in order to break even the system cost) and the request rate presented to the central repository.

We see from the table that by taking advantage of the nowadays low storage cost, optimal caching can achieve great saving compared with request batching. There is not much cost reduction if caching and request batching are combined (this is because the repository requests are for the not-so-popular movies, rendering request batching in that level not effective). The naive way of storing 20% most popular movies in a local servers does not achieve much cost saving compared with either request batching or storing all the movies locally, showing indeed there is an optimal number of movies to achieve minimal cost. More saving in cost is achieved when videos are more skewed in popularity.[4] If multicasting or server communication is possible, the cost can be further reduced. We also see from the table that the request rate in the central repository is greatly reduced through caching.

We show in Fig. 4.64 C versus the total request rate Λ, for the six systems considered above, and the multicasting (on-demand precaching) and chaining schemes. We have also shown the case $N_s = 2$ for optimal caching. By comparing the cases of VOD and storing all movies locally, we see that when Λ is low, we should use VOD, and when Λ is high, we should stored all the movies locally. Request batching can be used to reduce the cost substantially compared with the VOD case, but when Λ is high, its cost can still be higher than storing all the movies locally. Storing 20% of the movies locally, so that 80% of the requests can be directly served from the local servers, achieves good cost saving for a certain range of Λ, indicating that this "one size fits all" approach does not always lead to good decision. The cases of optimal caching and combined scheme achieves low cost, though they themselves do not differ much from each other. With multicasting, there is a slight decrease in cost. However, chaining is able to reduce system cost greatly, due to the fact that there is much less storage duplication and stream cost in the scheme (and hence lower storage and stream cost). When N_s is decreases, there can be more sharing in storage, and hence the cost is decreased.

We show in Fig. 4.65 the optimal number of movies that should be stored locally versus Λ, for the optimal caching and the combined scheme (We also show the case with $N_s = 2$ for optimal caching). We see that as Λ increases, the number of movies stored locally also increases (first quite sharply and then very slowly) and approaches the number of movies in the system (i.e., storing all movies locally). There is also not much difference between the optimal caching and the combined scheme. As N_s is decreased, the number of movies

[4] For 90/10 video popularity, cost per user for batching is \$0.435, for optimal caching is \$0.189 (94 movies in each local server with the repository request rate being 52.73 req./hr), and for the combined scheme is \$0.183 (88 movies in each local server with the repository request rate being 69.51 req./hr).

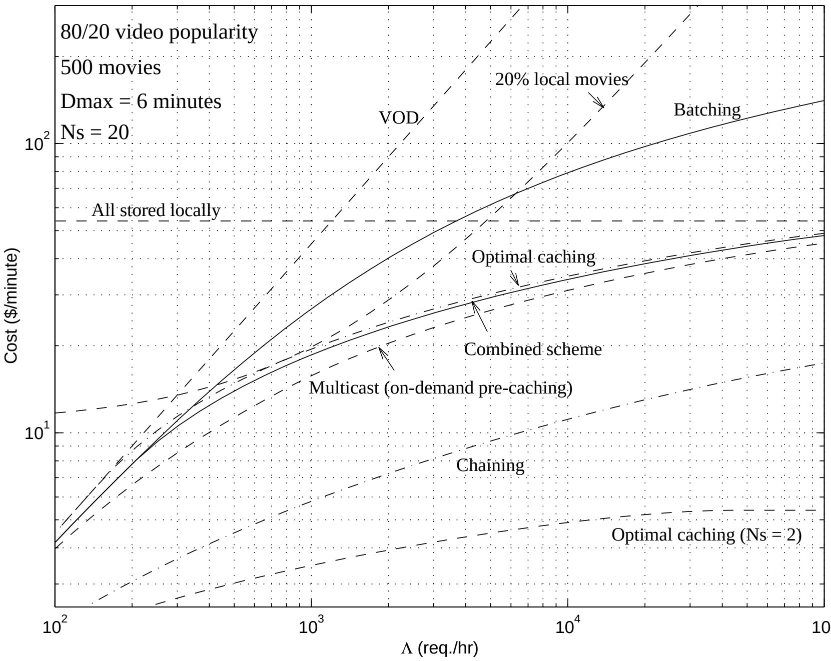


Figure 4.64: The system cost per minute versus Λ ($\gamma = 0.002$, $N_s = 20$, number of movies $= 500$, 80/20 video popularity, $D_{max} = 6$ minutes, $T_h = 90$ minutes, and system operated 6 hours a day)

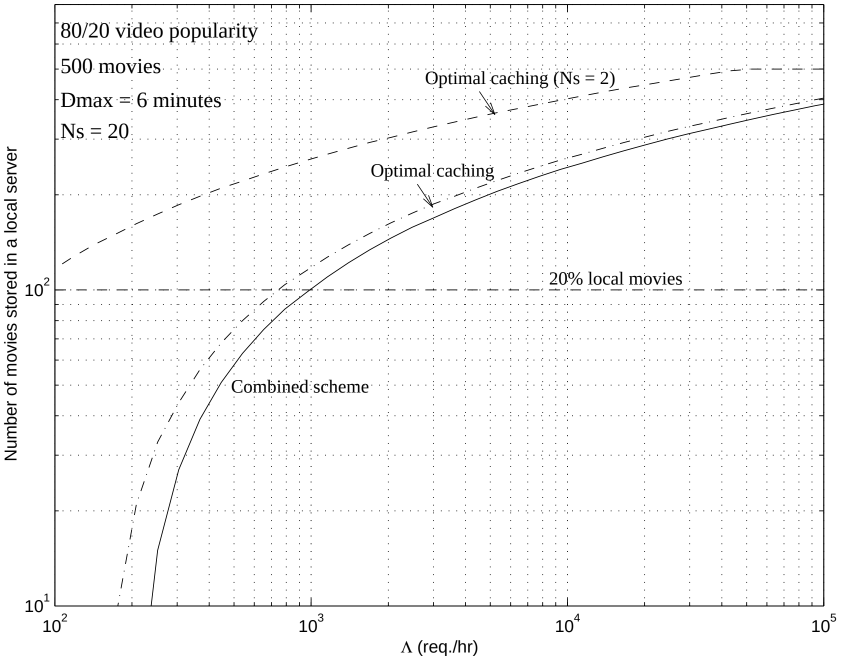


Figure 4.65: Number of movies stored versus Λ ($\gamma = 0.002$, $N_s = 20$, number of movies = 500, 80/20 video popularity, $D_{max} = 6$ minutes, $T_h = 90$ minutes and system operated 6 hours per day)

stored locally increases; and when Λ increases beyond a value, all the movies should be duplicated locally to take advantage of the low storage cost (since stream cost would be too high beyond such request rate).

4.7 Conclusions

A distributed servers architecture is able to achieve both storage and streaming scalability. In such a system, a number of repository servers store all the video contents of interest to a large pool of geographically distributed users. Due to the limited server bandwidth and the likelihood of high transmission cost from the repository to the users, using the repository to directly serve the users would not be effective. Using a number of local servers can increase the streaming capacity while lowering the network transmission cost. We have investigated such a video system for on-demand services, in which a number of local servers cache the video contents assessed from a repository server. We have studied a number of caching schemes, depending on whether the local servers are able to exchange their video contents among themselves or not, and whether the repository unicasts or multicasts the contents to the local servers. We have addressed the trade-off between the number of streams used and the storage requirement. Given a certain storage cost and streaming cost, we have studied when (i.e., under what conditions) and how much to cache a video file locally in order to minimize the system cost. We have also shown the cost advantage a distributed servers architecture can bring compared with a batching system.

Our caching schemes are effective in reducing the network bandwidth from the repository to the local servers. Generally, for unpopular movies, we should stream it from the central repository, and for the popular ones, we should store it locally. For intermediate popularity, we should partially store it in the local servers, with its storage requirement increases with its popularity. Multicasting is found to be able to reduce system cost; and in some cases it can be quite significantly. If the local servers can communicate with each other, the cost can be further reduced. We have shown that given that the current storage cost is so low, distributed servers architecture can trade-off storage cost with communication cost effectively. Compared with a request batching system, it can achieve both lower overall system cost and lower delay. The more skewed video popularity is, the more saving in cost a distributed servers architecture can achieve.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

Video-on-demand (VOD) refers to video services in which a user is able to request from a server any video content at any time. VOD encompasses many applications important in entertainment, education and advertising, such as movie-on-demand, news-on-demand, distance learning, home shopping, various interactive training programs, information kiosks, etc.

In order to provide VOD services accommodating thousands of video titles and thousands of concurrent users, a VOD system has to be scalable – scalable in storage and scalable in streaming capacity. We have considered in this thesis how such scalable services can be provided using hierarchical storage systems, request batching and multicasting, and distributed servers architecture. Our goal is to design such a system so as to offer high service quality with low cost and complexity.

Video servers based on hierarchical storage systems offer high-capacity, low-cost and scalable video storage. The system consists of a secondary level (characterized by fast file access and high throughput) and a tertiary level (characterized by cheap and large storage). Video files stored in the tertiary level are staged into the secondary level to be displayed. The design of such system includes its architectural parameters (bandwidth and storage in each level), and operational procedures (e.g., request scheduling and data replacement policies) for different level of user interactivities.

There are two types of users in a hierarchical system, hits and misses. We find that hits enjoy negligible delay; therefore the tertiary bandwidth should be designed to meet user delay requirement. An efficient tertiary level is very important in a high-performance hierarchical storage system, and hence if a video file is displayed after it is completely staged, we should parallelize the drive bandwidth. On the other hand, if a video can be displayed while it is being staged we should use multiple independent drives.

We have developed a simple model for a hierarchical storage system, from which we can specify the required bandwidth and storage in both secondary and tertiary levels to meet a certain user delay requirement, given specific application characteristics and a target request rate. We find that user delay increases rapidly when the arrival rate increases beyond the target arrival rate. Therefore, admission control should be used so that a hierarchical server should not be operated beyond the arrival rate under which it is designed.

We have seen that the number of files stored in the secondary level can be much lower than the total number of files in the system, and hence a hierarchical storage system is able to achieve much lower system cost compared to a system with secondary storage only. With large number of files, the storage and bandwidth requirements do not depend sensitively on the skewness of video popularity. The simple model we developed can also be used to specify bandwidth and storage requirements in a distributed storage system, in which multiple geographically-distributed servers get their video files from some remote repositories and streams the videos to their local users.

A near video-on-demand system achieves streaming scalability by batching requests for a particular video and serving them with a single multicast stream. We have considered providing such a service when there is a cost associated with using a multicast channel. Batching has then to be designed so as to amortize such channel cost.

We have studied a number of batching schemes. In the window-based batching schemes, user delay is bounded by a certain maximum value W; and in the batch-size based scheme, revenue can be maintained in each batch by allocating a stream whenever a certain number M of users are collected. We have analyzed and compared these schemes in terms of user delay experienced, the number of concurrent streams used, the number of users in a batch, etc. When the arrival rate is high, it is advantageous to use window-based scheme due to its bounded delay and high profit, but when the arrival rate is not high, it is advantageous to use batch-size based scheme to maintain profitability. We therefore proposed a combined adaptive scheme in which system profit and service quality (in terms of user delay experienced) can be balanced.

We have also studied the minimum number of channels required to satisfy a certain user delay requirement, and we find that when the delay requirement is low and the request rate is not high, the number of channels required can be quite low.

We have considered how profit can be maximized in providing near VOD services, given a certain user reneging behavior. We find that maximizing profit per-batch generally leads to long batching period and high user loss rate. Maximizing profit rate, on the other hand, encourages more frequent smaller returns. However, the loss rate may still be undesirably high, and hence a shorter batching period should be used in reality. Generally, the higher user delay tolerance is, the longer the batching period is. Our results also show that the optimal operating point also does not depend very much on the reneging function besides its mean. We find that the maximum profit rate for the window-based scheme and the batch-size based scheme are very similar, if both schemes are profitable. However, the batch-size based scheme is able to always maintain profit even when the arrival rate is low. We have therefore studied how the batch-size based scheme can be combined with the window-based scheme so that the window size can be dynamically adjusted to improve profit. We have shown that such a scheme is able to adapt to fluctuating request rete and still achieves high profit.

A distributed servers architecture is able to achieve both storage and streaming scalability. In such a system, a number of repository servers store all the video contents of interest to a large pool of geographically distributed users. Due to the limited server bandwidth and the likelihood of high transmission cost from the repository to the users, using the repository to directly serve the users would not be effective. Using a number of local servers can increase the streaming capacity while lowering the network transmission cost. We have investigated such a video system for on-demand services, in which a number of local servers cache the video contents assessed from a repository server. We have studied a number of caching schemes, depending on whether the local servers are able to exchange their video contents among themselves or not, and whether the repository unicasts or multicasts the contents to the local servers. We have addressed the trade-off between the number of streams used and the storage requirement. Given a certain storage cost and streaming cost, we have studied when (i.e., under what conditions) and how much to cache a video file locally in order to minimize the system cost. We have also shown the cost advantage a distributed servers architecture can bring compared with a batching system.

Our caching schemes are effective in reducing the network bandwidth from the repository to the local servers. Generally, for unpopular movies, we should stream it from the central repository, and for the popular ones, we should store it locally. For intermediate popularity, we should partially store it in the local servers, with its storage requirement increases with its popularity. Multicasting is found to be able to reduce system cost; and in some cases it can be quite significantly. If the local servers can communicate with each other, the cost can be further reduced. We have shown that given that the current storage cost is so low, distributed servers architecture can trade-off storage cost with communication cost effectively. Compared with a request batching system, it can achieve both lower overall system cost and lower delay. The more skewed video popularity is, the more saving in cost a distributed servers architecture can achieve.

5.2 Future Work

The work done in this dissertation can be extended in the following directions:

- Influence of user arrival processes — Most of our results are derived based on stationary, quasi-stationary, or in some cases, non-stationary Poisson arrival. While Poisson arrival is generally assumed in literature on server design, the impact of non-Poisson arrival processes on system design should be studied. Such processes may include periodic arrivals, batch arrivals, Markov-modulated processes, etc.

- Estimation of video popularity — Video popularity may be time-dependent or event-driven. A good video system should be able to adapt to both short-term and long-term variation of the popularity of a file. We have shown that knowing video popularity can bear great effects on system decision and hence profits. Estimating such popularity becomes important in system performance.

- Round-robin staging in a hierarchical storage system — So far, in the hierarchical storage system, we have considered file-by-file staging mechanism, in which a file has to be completely staged from the tertiary level before another file can replace it in a tertiary drive. We may also consider round-robin staging, in which a video file is partitioned into several blocks and staged on the block-by-block basis. Once a video file starts to be staged, it can then be displayed (continuity hence has to be satisfied). Since files can now be staged in a multiplexed manner, this scheme potentially offers lower start-up delay. Since more exchanges are involved in staging a file, it incurs more overheads. For example, if exchange time is negligible, round-robin staging would have lower start-up delay; however, if the library exchange time is high, file-by-file staging would be better. Therefore conditions under which round-robin staging achieves better delay performance can be established.

- Channel allocation and assignment in nVOD — So far we have considered that network channels can be acquired on a demand basis. Network channels may also be leased to a server, hence putting an upper limit on how many channels a server may use at any one time. The limited number of channels then have to be shared among all the movies. We have studied the planning issue for a single movie, and here we are interested in multiple movies and address the minimum number of channels necessary to meet a certain user delay or loss requirement.

Given a certain limited number of channels, we would also like to consider assignment policies when channels are about to run out. Since different movies may have different lengths, pay-per-view, and popularities, for limited channels, the decision on which movie requests to be served would bear influences in system performance (e.g., system profit or user loss rate).

- Comparison of batching schemes in nVOD — There have been a number of batching schemes proposed and studied in literature (see previous work in Chapter 3). Various batching schemes are studied under different objectives or assumptions. There has not been a uniform way to compare them, in terms of their batch sizes, stream consumptions, user delay and fairness issues.

- Web servers — So far our study has focused on video applications. Our research results and efforts can be extended to the web servers environment, in which multiple sites serving various kinds of contents (data, graphics or continuous media) with users accessing these contents/pages through a network. As different from video applications, web data is smaller and tend to be more heterogeneous in size and lifetime than video data. Therefore sophisticated batching schemes, caching schemes and hierarchy would have to be studied.

  Some contents in a web servers environment have to be delivered reliably and updated very frequently (e.g., stock information). The issues then become what batching, caching and delivery schemes should be used so as to minimize network or server bandwidths and to guarantee timely delivery, and how to measure the popularity of a page and incrementally update local pages instead of downloading the whole pages again each time a piece of information is needed.

- Limited storage in distributed server architecture — In the caching schemes we studied, we have considered that the storage comes with a cost depending on the size and length of storage. Another interesting problem is that under the condition of limited storage, how should the storage be shared among the files through caching so as to minimize the network cost while meeting the performance requirement? The caching schemes would depend on the file size among others. For example, a large file may be less likely to be cache unless it is extremely popular; and a small file would be more likely to be locally cached. We will study caching and replacement schemes here. Both static policies (policies independent of the system states) and dynamic policies (policies dependent on the current system states) can be investigated. dependent on the current system states) can be derived.

Bibliography

[1] T. Little and D. Venkatesh, "Prospects for interactive video-on-demand," *IEEE Multimedia Magazine*, pp. 14–24, Fall 1994.

[2] V. O. Li and W. Liao, "Distributed multimedia systems," *Proceedings of the IEEE*, vol. 85, pp. 1063–1108, July 1997.

[3] F. A. Tobagi, "Distance learning with digital video," *IEEE Multimedia Magazine*, pp. 90–94, Spring 1995.

[4] K. Cleary, "Video on demand — competing technologies and services," in *Proceedings of International Broadcasting Convention*, (Amsterdam, Netherlands), pp. 432 – 437, 14–18 Sept. 1995.

[5] B. Furht, D. Kalra, F. L. Kitson, A. A. Rodriguez, and W. E. Wall, "Design issues for interactive television systems," *IEEE Computer Magazine*, pp. 25–38, May 1995.

[6] A. D. Gelman, H. Kobrinski, L. S. Smoot, S. B. Weinstein, M. Fortier, and D. Lemay, "A store-and-forward architecture for video-on-demand service," *Canadian Journal of Electrical and Computer Engineering*, vol. 18, pp. 37–40, January 1993.

[7] D. D. Harman, G. Huang, G.-H. Im, M.-H. Nguyen, J.-J. Werner, and M. K. Wong, "Local distribution for IMTV," *IEEE Multimedia Magazine*, pp. 14–23, Fall 1995.

[8] H. J. Stüttgen, "Network evolution and mutlimedia communication," *IEEE Multimedia Magazine*, pp. 42–59, Fall 1995.

[9] H. Kobayashi, *Modeling and Analysis: An Introduction to System Performance Evaluation Methodology.* Addison Wesley, 1981.

[10] L. Kleinrock, *Queueing Systems: Theory*, vol. 1. Wiley Interscience, 1975.

[11] C. C. Bisdikian and B. V. Patel, "Issues on movie allocation in distributed video-on-demand systems," in *Proceedings of ICC'95*, pp. 250–255, 1995.

[12] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," in *Proceedings of ACM Multimedia*, pp. 15–23, 1994.

[13] F. Schaffa and J.-P. Nussbaumer, "On bandwidth and storage tradeoffs in multimedia distribution networks," in *Proceedings of Infocom'95*, pp. 1020–1026, 1995.

[14] L. D. Giovanni, A. M. Langellotti, L. M. Patitucci, and L. Petrini, "Dimensioning of hierarchical storage for video on demand services," in *Proceedings of ICC'94*, pp. 1739–1743, 1994.

[15] Y. N. Doğanata and A. N. Tantawi, "Making a cost-effective video server," *IEEE Multimedia Magazine*, pp. 22–30, Winter 1994.

[16] D. Patterson, G. Gibson, and R. Katz, "A case for redundant arrays of inexpensive disks (RAID)," in *Proceedings of ACM SIGMOD Conference on the Management of Data*, pp. 109–116, 1988.

[17] D. Patterson, P. Chen, G. Gibson, and R. Katz, "Introduction to redundant arrays of inexpensive disks (RAID)," in *Digest of Papers: Spring Compcon*, pp. 112–117, Feb 27 - Mar 3 1989.

[18] G. Ganger, B. Worthington, R. Hou, and Y. Patt, "Disk arrays – high-performance, high reliability storage subsystems," *Computer Magazine*, pp. 30–36, March 1994.

[19] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "RAID: High-performance, reliable secondary storage," *ACM Computing Surveys*, vol. 26, pp. 145–185, June 1994.

[20] P. Chen and D. Patterson, "Maximizing performance in a striped disk array," in *The 17th IEEE Annual International Symposium on Computer Architecture*, pp. 322–331, May 28-31 1990.

[21] S. Ng, "Some design issues of disk arrays," in *Digest of Papers: Spring Compcon*, pp. 137–142, Feb 27 - Mar 3 1989.

[22] W. Tetzlaff and R. Flynn, "Elements of scalable video servers," in *Compcon: Digest of Papers*, pp. 239–248, IEEE, 1995.

[23] W. J. Bolosky, J. S. Barrera, R. P. Draves, R. P. Fitzgerald, G. A. Gibson, M. B. Jones, S. P. Levi, N. P. Myhrvold, and R. F. Rashid, "The tiger video fileserver," in *Proceedings of Sixth International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 96)*, April 1996.

[24] J. Y. B. Lee, "Parallel video servers: A tutorial," *IEEE Multimedia*, vol. 5, pp. 20–28, April - June 1998.

[25] Y. B. Lee and P. C. Wong, "Viola — video on local-area network," in *Proceeding of the 2nd IASTED/ISMM International Conference on Distributed Multimedia Systems*, (Stanford, CA), 1995.

[26] Y. B. Lee and P. C. Wong, "A server array approach for video-on-demand service on local area networks," in *Proceedings of Infocom'96*, (San Franscisco, CA), pp. 27–34, IEEE, March 1996.

[27] B. Duzett and R. Buck, "An overview of the ncube 3 supercomputer," in *Proceedings of 4th Symposium on the Frontiers of Massively Parallel Computation*, pp. 458–464, IEEE, 1992.

[28] A. Laursen, J. Olkin, and M. Porter, "Oracle media server: Providing consumer based interactive access to multimedia data," in *Proceedings of ACM SIGMOD*, (Minneapolis, Minnesota), pp. 470–477, ACM, 1994.

[29] R. Buck, "The oracle media server for ncube massively parallel systems," in *Proceedings of the 8th International Parallel Processing Symposium*, pp. 670–673, IEEE, 1994.

[30] G. Arnaiz, "Tuning oracle7 for ncube," in *Proceedings of the 28th Annual Hawaii International Conference on System Sciences*, pp. 137–139, IEEE, 1995.

[31] A. Laursen, J. Olkin, and M. Porter, "Oracle media server framework," in *Compcon: Digest of Papers*, pp. 203–208, IEEE, 1995.

[32] R. L. Haskin and F. L. Stein, "A system for the delivery of interactive television programming," in *Compcon: Digest of papers*, pp. 209–215, 1995.

[33] R. L. Haskin and F. B. Schmuck, "The tiger shark file system," in *Proceedings of COMPCON'96*, pp. 226–231, IEEE, 1996.

[34] F. A. Tobagi and J. Pang, "Starworks^{TM} – a video applications server," in *Proceedings of Compcon*, (San Fransisco, CA), February 22-25 1993.

[35] F. A. Tobagi, J. Pang, R. Baird, and M. Gang, "Streaming RAID^{TM} – a disk array management system for video files," in *Proceedings of First ACM International Conference on Multimedia*, (Anaheim, CA), August 1-6 1993.

[36] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan, "The HP AutoRAID hierarchical storage system," *ACM Transactions on Computer Systems*, vol. 14, pp. 108–36, February 1996.

[37] D. J. Gemmell, H. M. Vin, D. D. Kandlur, P. V. Rangan, and L. A. Rowe, "Multimedia storage servers: A tutorial," *IEEE Computer Magazine*, pp. 40–49, May 1995.

[38] A. N. Mourad, "Issues in the design of a storage server for video-on-demand," *ACM Multimedia Systems Journal*, vol. 4, pp. 70–86, April 1996.

[39] A. Mourad, "Reliable disk striping in video-on-demand servers," in *Proceedings of Int. Conf. on Distr. Multimedia Syst. & Appl.*, August 1995.

[40] A. Mourad and Y. T. Wang, "Architecture and performance issues of media server and content delivery," in *Proceedings of Digital Image Storage and Archiving Systems, Vol. 2606*, SPIE, October 1995.

[41] D. J. Gemmell, J. Han, R. J. Beaton, and S. Christodoulakis, "Delay-sensitive multimedia on disks," *IEEE Multimedia Magazine*, pp. 56–67, Fall 1994.

[42] J. Gemmell and S. Christodoulakis, "Principles of delay-sensitive multimedia data storage and retrieval," *ACM Transactions on Information Systems*, vol. 10, pp. 51–90, January 1992.

[43] R. Steinmetz, "Analyzing the multimedia operating system," *IEEE Multimedia Magazine*, pp. 68–84, Spring 1995.

[44] K. K. Ramakrishnan, L. Vaitzblit, C. Gray, U. Vahalia, D. Ting, P. Tzelnic, S. Glaser, and W. Duso, "Operating system support for a video-on-demand file service," *Multimedia Systems Journal*, vol. 3, no. 2, pp. 53–65, 1995.

[45] S. A. Barnett and G. J. Anido, "Performability of disk-array-based video servers," *Multimedia Systems*, no. 6, pp. 60–74, 1998.

[46] H. M. Vin, A. Goyal, and P. Goyal, "Algorithms for designing multimedia servers," *Computer Communications*, vol. 18, pp. 192–203, March 1995.

[47] A. Mourad, "Evaluation of paging policies in a hierarchical multimedia server," in *Proceedings of Int. Conf. on Computer Simulations*, September 1995.

[48] A. Dan, D. M. Dias, R. Mukherjee, D. Sitaram, and R. Tewari, "Buffering and caching in large-scale video servers," in *Compcon'95*, pp. 217–224, 1995.

[49] T. L. Kunii, Y. Shinagawa, R. M. Paul, M. F. Khan, and A. A. Khokhar, "Issues in storage and retrieval of multimedia data," *Multimedia Systems*, vol. 3, no. 5–6, pp. 298–304, 1995.

[50] H. M. Vin and V. Rangan, "Designing a multiuser HDTV storage server," *IEEE Journal on Selected Areas in Communications*, vol. 11, pp. 153–164, January 1993.

[51] W. D. Sincoskie, "System architecture for a large scale video on demand service," *Computer Networks and ISDN Systems*, vol. 22, pp. 155–162, 1991.

[52] P. N. Misra, "Capacity analysis of the mass storage system," *IBM System Journal*, vol. 20, no. 3, pp. 346–364, 1981.

[53] M. G. Kienzle, A. Dan, D. Sitaram, and W. Tetzlaff, "Using tertiary storage in video-on-demand servers," in *Compcon: Digest of Papers*, pp. 225–232, IEEE, 1995.

[54] C. Schűnemann and W. G. Spruth, "Storage hierarchy technology and organization," in *Digital Memory and Storage* (W. E. Proebster, ed.), pp. 377–389, Braunschweig: Vieweg, 1978.

[55] A. L. Chervenak, D. A. Patterson, and R. H. Katz, "Choosing the best storage system for video service," in *Proceedings of ACM Multimedia*, pp. 109–119, 1995.

[56] Y. Won and J. Srivastava, "Minimizing blocking probability in a hierarchical storage based vod server," in *Proceedings of International Workshop on Multimedia Database Management Systems*, (Blue Mountain Lake, NY, USA), pp. 12–19, 14-16 Auguest 1996.

[57] G. Bianchi, R. Melen, and A. Rainoni, "Performance of video storage hierarchy in interactive video services networks," in *Proceedings of IEEE Globecom'95*, (Singapore), pp. 815–810, 13-17 November 1995.

[58] S. Ghandeharizadeh and C. Shahabi, “On multimedia repositories, personal computers, and hierarchical storage systems,” in *Proceedings of ACM Multimedia*, pp. 407–416, 1994.

[59] S. Ghandeharizadeh, A. Dashti, and C. Shahabi, “A pipelining mechanism to minimize the latency time in hierarchical multimedia storage managers,” *Computer Communication*, vol. 18, pp. 170–184, March 1995.

[60] J. Z. Wang and K. A. Hua, “A bandwidth management technique for hierarchical storage in large-scale multimedia servers,” in *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, (Otta, Ont., Canada), pp. 261–268, IEEE, 3–6 June 1997.

[61] T. Nemoto, Y. Sato, K. Mogi, K. ichiro Ayukawa, M. Kitsuregawa, and M. Takagi, “Performance evaluation of cassatte migration mechanism for scalable tape archiver,” in *Proceedings of SPIE High-Density Data Recording and Retrieval Technologies*, (Philadelphia, PA), Oct 1995.

[62] A. Drapeau and R. Katz, “Striped tape arrays,” in *Proceedings of IEEE Symposium on Mass Storage Systems*, pp. 257–265, 1993.

[63] D. A. Ford, R. J. T. Morris, and A. E. Bell, “Redundant arrays of independent libraries (rail): A tertiary storage system,” in *Proceedings of COMPCON’96*, pp. 280–285, 1996.

[64] S.-W. Lau and J. C. S. Lui, “Scheduling and data layout policies for a near-line multimedia storage architecture,” *Multimedia Systems*, vol. 5, pp. 310–323, September 1997.

[65] O. I. Pentakalos, D. A. Menascé, M. Halem, and Y. Yesha, “Analytic performance modelling of hierarchical mass storage systems,” *IEEE Transactions on Computers*, vol. 46, pp. 1103–1118, October 1997.

[66] H. Suzuki and K. Nishimura, “Performance analysis of a storage hierarchy for video servers,” *Systems and Computers in Japan*, vol. 28, pp. 300–308, March 1997.

[67] A. Merchant, Q. Ren, and B. Sengupta, “Hierarchical storage servers for video on demand: Feasibility, design and sizing,” in *Proceedings of IEEE Globecom’96*, (London, UK), pp. 272–278, 18-22 November 1996.

[68] S.-H. G. Chan and F. A. Tobagi, “Hierarchical storage systems for on-demand video servers,” in *Proceedings of the SPIE (the International Society for Optical Engineering) High-Density Data Recording and Retrival Technologies*, (Philadelphia, PA), pp. 103–120, SPIE, October 1995.

[69] S.-H. G. Chan and F. A. Tobagi, “Hierarchical storage systems for interactive video-on-demand.” CSL-TR-97-723, Computer System Laboratory Technical Report, Stanford University, April, 1997.

[70] C. Federighi and L. A. Rowe, “A distributed hierarchical storage manager for a video-on-demand system,” in *Proceedings of Storage and Retrieval for Image and Video Databases II, IS & T/SPIE Symp. on Elect. Imaging Sci. & Tech.*, (San Jose, CA), Feb. 1994.

[71] R. Syski, *Introduction to Congestion Theory in Telephone Systems.* New York: Elsevier Science Pub. Co., 2nd ed., 1986.

[72] N. U. Prabhu, *Queues and Inventories, a Study of Their Basic Stochastic Processes.* New York: Wiley, 1965.

[73] S. S. Lavenberg, *Computer Performance Modeling Handbook.* New York: Academic Press, 1983.

[74] S. A. Nozaki and S. M. Ross, “Approximations in finite-capacity multi-server queues with poisson arrivals,” *J. of Appl. Prob.*, no. 15, pp. 826–834, 1978.

[75] A. D. Gelman and S. Halfin, "Analysis of resource sharing in information providing services," in *Proceedings of IEEE Globecom'98*, pp. 312–316, 1990.

[76] A. Dan, D. Sitaram, and P. Shahabuddin, "Dynamic batching policies for an on-demand video server," *Multimedia Systems*, vol. 4, pp. 112 – 121, June 1996.

[77] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "On optimal batching policies for video-on-demand storage servers," in *Proceedings of the 3rd International Conference on Multimedia Computing and Systems*, (Hiroshima, Japan), pp. 253–258, IEEE, 17-23 June 1996.

[78] K. C. Almeroth, A. Dan, D. Sitaram, and W. H. Tetzlaff, "Long term resource allocation in video delivery systems," in *Proceedings of IEEE Infocom'97*, pp. 1333–1340, 1997.

[79] E. L. Abram-Profeta and K. Shin, "Scheduling video programs in near video-on-demand systems," in *Proceedings of ACM International Multimedia Conference*, pp. 359–371, Nov. 9-13 1997.

[80] L. Golubchik, J. C. S. Lui, and R. Muntz, "Reducing I/O demand in video-on-demand storage servers," in *Proceedings of the ACM SIGMETRICS'95*, (Ottawa, Ontario, Canada), pp. 25–36, ACM, 1995.

[81] C. Aggarwal, J. Wolf, and P. S. Yu, "On optimal piggybacking merging policies for video-on-demand systems," *Performance Evaluation Review*, vol. 24, pp. 200–209, May 1996.

[82] S. Sheu, K. A. Hua, and W. Tavanapong, "Chaining: A generalized batching technique for video-on-demand systems," in *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, (Ottawa, Ont., Canada), pp. 110–117, IEEE, 3-6 June 1997.

[83] W. Liao and V. O. Li, "The split and merge protocol for interactive video-on-demand," *IEEE Multimedia*, pp. 51–62, October-December 1997.

[84] S.-H. G. Chan, F. Tobagi, and T.-M. Ko, "Providing on-demand video services using request batching," in *Proceedings of the 1998 IEEE International Conference on Communications (ICC'98)*, (Atlanta, Georgia), pp. 1716–1722, June 7-11 1998.

[85] S.-H. G. Chan, F. Tobagi, and T.-M. Ko, "Bandwidth planning in near video-on-demand," in *Proceedings of the 1998 IEEE International Symposium on Circuits and Systems (ISCAS'98)*, (Monterey, CA), pp. 61–64, May 31 - June 3 1998.

[86] S. Chen and M. Thapar, "A novel video layout strategy for near-video-on-demand servers," in *Proceedings of IEEE International Conference on Multimedia Computing and Systems'97*, (Ottawa, Ont., Canada), pp. 37–45, 3–6 June 1997.

[87] M. N. Garofalakis, B. Özden, and A. Silberschatz, "On periodic resource scheduling for continuous media databases," in *IEEE RIDE (Research in data engineering)*, (Orlando, Florida), pp. 111–120, Feb 1998.

[88] P. S. Yu, J. L. Wolf, and H. Shachnai, "Design and analysis of a look-ahead scheduling scheme to support pause-resume for video-on-demand applications," *Multimedia Systems*, vol. 3, no. 4, pp. 137–149, 1995.

[89] K. C. Almeroth and M. H. Ammar, "The use of multicast delivery to provide a scalable and interactive video-on-demand service," *IEEE Journal on Selected Areas in Communications*, vol. 14, pp. 1110–1122, August 1996.

[90] A. Dan, P. Shahabuddin, D. Sitaram, and D. Towsley, "Channel allocation under batching and vcr control in video-on-demand systems," *Journal of Parallel and Distributed Computing*, vol. 30, no. 2, pp. 168–179, 1995.

[91] V. O. K. Li, W. Liao, X. Qiu, and E. W. M. Wong, “Performance model of interactive video-on-demand systems,” *IEEE Journal on Selected Areas in Communications*, vol. 14, pp. 1099–1109, August 1996.

[92] E. Gelenbe and H. Shachnai, “On g-networks and resource allocation in multimedia systems,” in *IEEE RIDE (Research in Data Engineering)*, (Orlando, Florida), pp. 104–110, Feb. 1998.

[93] A. K. Tsiolis and M. K. Vernon, “Group-guaranteed channel capacity in multimedia storage servers,” *Performance Evaluation Review*, vol. 25, no. 1, pp. 285–297, 1997.

[94] S. A. Barnett and G. J. Anido, “A cost comparison of distributed and centralized approaches to video-on-demand,” *IEEE Journal on Selected Areas in Communications*, vol. 14, pp. 1173–1183, August 1996.

[95] C. Papadimitriou, S. Ramanathan, P. V. Rangan, and S. SampathKumar, “Multimedia information caching for personalized video-on-demand,” *Computer Communications*, vol. 18, pp. 204–216, March 1995.

[96] T. D. C. Little and D. Venkatesh, “Popularity-based assignment of movies to storage devices in a video-on-demand system,” *Multimedia Systems*, no. 2, pp. 280–287, 1995.

[97] H. Ghafir and H. Chadwick, “Multimedia servers – design and performance,” in *Proceedings of Globecom*, pp. 886–889, 1994.

[98] S.-W. Lau, J. C. S. Lui, and L. Golubchik, “Merging video streams in a multimedia storage server: Complexity and heuristics,” *ACM Multimedia Systems*, vol. 6, pp. 29–42, Jan. 1998. NR.

[99] Y. Wang, Z.-L. Zhang, D. H. C. Du, and D. Su, “A network-conscious approach to end-to-end video delivery over wide area networks using proxy servers,” in *Proceedings of the IEEE Infocom’98*, (San Francisco, USA), pp. 660–667, March 29 - April 2 1998.

[100] A. Dan, M. Kienzle, and D. Sitaram, “A dynamic policy of segment replication for load-balancing in video-on-demand servers,” *Multimedia Systems*, vol. 3, pp. 93–103, July 1995.

[101] G. H. Petit, D. Deloddere, and W. Verbiest, “Bandwidth resource optimization in video-on-demand network architectures,” in *Proceedings of the 1st IEEE International Workshop on Community Networking*, (NY, New York), pp. 91–97, 13-14 July 1994.