11-1997

Tertiary Storage in Multimedia Systems: Staging or Direct Access?

Hwee Hwa PANG
Singapore Management University, hhpang@smu.edu.sg

DOI: https://doi.org/10.1007/s005300050070

Tertiary storage in multimedia systems: staging or direct access?

HweeHwa Pang

Institute of Systems Science, National University of Singapore, Heng Mui Keng Terrace, Kent Ridge, Singapore 119597, Republic of Singapore
e-mail: hhpang@iss.nus.sg

Abstract. Multimedia applications that are required to manipulate large collections of objects are becoming increasingly common. Moreover, the size of multimedia objects, which are already huge, are getting even bigger as the resolution of output devices improve. As a result, many multimedia storage systems are not likely to be able to keep all of their objects disk-resident. Instead, a majority of the less popular objects have to be off-loaded to tertiary storage to keep costs down. The speed at which objects can be accessed from tertiary storage is thus an important consideration. In this paper, we propose an adaptive data retrieval algorithm that employs a combination of staging and direct access in servicing tertiary storage retrieval requests. At retrieval time, an object that resides in tertiary storage can either be staged to and then played back from disks, or the object can be accessed directly from the tertiary drives. We show that a simplistic policy that adheres strictly to staging or direct access does not exploit the full retrieval capacity of both the tertiary library and the secondary storage. To overcome the problem, we propose a data retrieval algorithm that dynamically chooses between staging and direct access, based on the relative load on the tertiary versus secondary devices. A series of simulation experiments confirms that the algorithm achieves good access times over a wide range of workloads and resource configurations. Moreover, the algorithm is very responsive to changing load conditions.

Key words: Multimedia server – Storage hierarchy – Tertiary library – Data migration – Feedback control

1 Introduction

In recent years, demand for multimedia applications that are capable of manipulating both continuous media (CM) data, such as video and audio, and non-CM data, e.g. text and images, has been growing rapidly. Many of these applications, including interactive multimedia education [2] and news on demand [16], are expected to provide access to tens of thousands of objects. Moreover, the size of these multimedia objects are likely to be large. For example, a 2-h MPEG-1 movie can easily occupy 1.5 GB of storage [9]. Consequently, it would be very costly to maintain all of the objects on secondary storage devices like magnetic disks. A more economical alternative is to hold only the popular objects on disks, and to keep the less frequently accessed objects in a tertiary library that offers lower costs per byte of storage. Given that a small number of objects account for a majority of the accesses in most applications e.g., [15, 3], the bulk of the objects can reside in the tertiary library, thus significantly reducing the number of disks that are required.

While off-loading objects that are less popular to a tertiary library reduces the disk space requirement of a multimedia storage system, this practice also complicates its data management function. Instead of dealing only with disk drives, the storage system now has to contend with both secondary and tertiary storages, which have very different access characteristics. One issue that arises is data retrieval from the slower, tertiary library. For non-CM objects, this is straightforward, as they can be transmitted directly to the users at the maximum transfer rate of the tertiary library. CM objects, however, require their components to be played back at a controlled rate, e.g., 30 frames per second. Since the playback rate is not likely to correspond to the maximum transfer rate of the tertiary library, there is a choice between *staging* and *direct access*, which operate the tertiary drives at their maximum speed and the CM objects' playback rate, respectively. This paper focuses on the retrieval of CM objects.

In the staging mode, the cartridge containing a requested object is first loaded into a tertiary drive. The object is then copied from the cartridge to a set of staging disks as fast as possible. Finally, the staging disks feed the data pages of the object to the user at the object's playback rate. The staging mode has the advantage of minimizing turnaround time at the tertiary library, thus reducing its chances of becoming a system bottleneck. One drawback of staging is that it requires disk space for the requested object. Moreover, the disk scheduler must be capable of preventing the I/Os generated by a staging activity from disrupting any active CM streams that the staging disks may be supporting. Another drawback is that staging delays could prolong access times.

The direct access mode requires a tertiary drive to retrieve a requested object at its playback rate, so that the retrieved data can be forwarded directly to the user. Thus, direct access is much simpler to implement than staging. However, direct access may not lead to effective tertiary library utilization. This is because tertiary drives are typically capable of retrieving at higher rates than the playback rate[1]. Due to the time-consuming nature of the search-forward and rewind operations, it is usually impractical to multiplex several retrievals from a drive, even if the required objects happen to reside on the same cartridge. Moreover, the long cartridge loading time of a tertiary library precludes each drive from exploiting any excess bandwidth to retrieve from multiple cartridges concurrently. As a result of this ineffective resource utilization, the tertiary library is prone to develop into a system bottleneck that retards access times.

To date, most work on handling tertiary storage devices was done in the context of mass storage systems [6, 7]. These systems include Lawrence Livermore Labs' LSS [8, 12], NASA's MSS-II [21], Los Alamos National Labs' CFS [4], the National Center for Atmospheric Research's MSS [20], and Epoch's InfiniteStorage Architecture [13]. All of these systems require data to be staged to disk before the data can be used. Recently, there has also been some work that specifically addresses tertiary storage support for multimedia applications. In [10], Ghandeharizadeh and Shahabi proposed a pipelining mechanism to overlap the playing back of the front portion of a CM object from the disks with the staging of the object's remaining portion. Finally, Kienzle et al. [14] developed a cost model to compare the space and retrieval costs of direct access versus staging. Using this cost model, they concluded that an object should only be accessed directly from a tertiary drive if its retrieval rate is similar to the playback rate of the object; staging is more appropriate when there is a considerable disparity between the two rates.

While most of the reported studies have favored staging over direct access, there is no reason why a multimedia storage system has to operate only in the staging mode. Indeed, an interesting possibility that allows a storage system to enjoy the advantage of both modes is for it to perform staging for some requests, and direct access for other requests. In this paper, we propose an adaptive staging–direct access algorithm, called *Asdac*, that dynamically selects between staging and direct access in servicing a new object request. This choice is governed by feedback on the relative load on the staging disks and the tertiary library. Due to the feedback control mechanism, we could not model the performance of *Asdac* analytically. Instead, the algorithm is evaluated using a multimedia storage system simulator that we developed. The evaluation shows that *Asdac* consistently outperforms static staging and direct access over a wide range of system configurations. Moreover, *Asdac* is able to quickly detect and adapt to changing load conditions.

The remainder of this paper is organized as follows. Section 2 describes the performance characteristics of a tertiary storage library. In Sect. 3, we introduce a couple of algorithms to retrieve data from tertiary storage. A multimedia storage system simulator, intended for studying the performance of the data retrieval algorithms, is presented in Sect. 4. Section 5 gives the results of a series of experiments highlighting the gains that *Asdac* brings about. Finally, Sect. 6 concludes the paper.

[1] The case where the retrieval rate of the tertiary drives is lower than the playback rate is not interesting to this work, as staging becomes the only feasible solution.

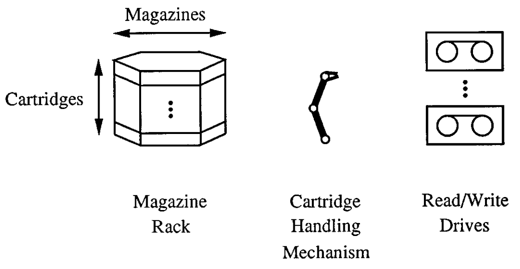


Fig. 1. Schematic representation of a tertiary storage library

2 Tertiary storage libraries

Figure 1 is a schematic representation of a typical tertiary storage library. It consists of a large set of cartridges, a number of read/write drives, and a mechanism to automatically load/unload a required cartridge into/from a drive. The cartridge-handling mechanism services only one cartridge movement request at a time, completing a load or unload operation before entertaining the next request. Each cartridge is stored in one of several magazines. The storage library services an object retrieval request in the following steps:

1. A free read/write drive is assigned for this object retrieval.
2. The magazine rack spins until the cartridge containing the requested object faces the cartridge-handling mechanism.
3. The cartridge-handling mechanism extracts the target cartridge from the magazine and slots the cartridge into the assigned drive.
4. The drive seeks to the starting location of the object.
5. The object is retrieved at a specified speed (the speed to use will be discussed shortly).
6. If the cartridge is a magnetic tape, it is rewound.
7. The magazine rack spins until the magazine that is supposed to hold the cartridge faces the cartridge-handling mechanism.
8. The cartridge-handling mechanism extracts the cartridge from the drive and slots the cartridge into the magazine.
9. The drive is freed.

As discussed in the introduction, an object retrieval (Step 5) can be carried out in one of two modes. In the first mode, the read/write drive transfers data directly to the user terminal at the playback rate of the object, which is frequently lower than the maximum transfer rate of the drive. This mode is called *direct access*. The second mode, *staging*, transfers the object from the tertiary drive to a set of staging disks as fast as the two devices allow, before playing back the object from the staging disks.

The performance of a tertiary storage library depends on the technologies used for the read/write drives and the cartridge autoloader, which comprises the magazine rack and

Table 1. Magnetic tape and magneto-optical disk technologies

Product	Capacity	Transfer rate	Cost US$	Source
EXB-8505XL	7–14 GB	0.5–1 MB/s	$3,000	[1]
DEC TZ87	20 GB	2.5 MB/s	$8,000	[1]
IBM 1/2″ 3490E	0.8 GB	3 MB/s	$20,000	[17]
Ampex DST 600	25 GB	15 MB/s	$150,000	[17]
1″ Metrum	12.5 GB	22 MB/s	$160,000	[17]
Sony SMO-F521	1.3 GB	1-2 MB/s	$2,200	[1]
EMO-1300	1-3 GB (2 Sides)	3 MB/s	$3,300	[1]
RICOH RS-5060K	1.3 GB	5 MB/s (Sync)	$4,290	[1]

cartridge-handling mechanism. Table 1 lists the characteristics of some magnetic tape and magneto-optical disk drives that are on the market. As shown in the table, many different types of tape drives are currently available, ranging from low-cost 8-mm drives that transfer at less than 1 MB/s to high-end drives boasting transfer rates of 22 MB/s at a much higher cost. The choices for magneto-optical drives are more limited, both in terms of transfer rate and cost. As for the cartridge autoloader, this could involve a high-speed carousel that takes less than a second to load/unload a cartridge, as in a StorageTek 9708 DataWheel. Alternatively, a magazine rack that holds a larger number of cartridges but incurs longer cartridge loading/unloading times could be used, as in a Box Hill Ice Box. These technologies enable a wide range of tertiary storage libraries representing different cost/performance trade-offs to be built.

The focus of this paper is not on assembling a set of specific read/write drives and autoloader technologies into a tertiary storage library that best meets a customer's storage capacity, access speed, and budget requirements. Instead, our objective is primarily to optimize the effectiveness of a given tertiary storage library. Specifically, we want to minimize the *average access time* of object retrievals from the tertiary library, defined as the elapsed time between the issuance of a retrieval request and the instant when the first page of data arrives at the user terminal. To achieve this objective, we propose in the next section a couple of algorithms to retrieve objects from a tertiary library. We then study the behaviors of the algorithms over a wide range of tertiary library, staging disks, and workload configurations, paying particular attention to the interplay between tertiary library and staging disks. Of course, a thorough understanding of the interplay between these two resources would help in selecting tertiary devices to complement the secondary storage at an installation, but that is beyond the scope of this paper.

3 Data retrieval algorithms

Having described the characteristics and performance objective of a tertiary library subsystem, we now introduce data retrieval algorithms for this subsystem. As explained in Sect. 1, staging a requested object from a tertiary drive minimizes the turnaround time of the drive, but adds a staging delay to the object's access time. In contrast, accessing the object directly from the tertiary drive eliminates the staging delay, but occupies the drive for a longer duration. This could cause new retrieval requests to experience long waiting times if all of the tertiary drives are engaged. Thus, direct access is expected to perform better under light load, where the number of busy drives is low most of the time, whereas staging is more appropriate when the tertiary library experiences heavy retrieval activities. This section introduces two data retrieval algorithms, *Staging-Occupancy*$_{Instant}$*%* and *Asdac*, both of which attempt to dynamically strike a balance between staging and direct access in order to minimize access time. The parameters of the two algorithms, which will be explained as they appear in the following description, are summarized in Table 2.

3.1 Staging-Occupancy$_{Instant}$%

The *Staging-Occupancy*$_{Instant}$*%* algorithm selects between staging and direct access according to the percentage of tertiary drives that are engaged. On one hand, if this percentage falls below *Occupancy*$_{Instant}$*%*, an input parameter, the algorithm concludes that the load is light and favors direct access in order to reduce access time. On the other hand, if the drive occupancy exceeds *Occupancy*$_{Instant}$*%*, *Staging-Occupancy*$_{Instant}$*%* switches to the staging mode in an effort to prevent the tertiary library from becoming a bottleneck.

The *Staging-Occupancy*$_{Instant}$*%* algorithm services an object request using the staging mode if at least *Occupancy*$_{Instant}$*%* of the tertiary drives are occupied (including the drive that is servicing the current request), and if the bandwidth that the staging disks can allocate for the staging activity exceeds the playback rate of the object[2]. If either of the two criteria is not satisfied, direct access mode is selected. Thus, *Staging-Occupancy*$_{Instant}$*%* essentially means to "attempt staging once the drive occupancy hits *Occupancy*$_{Instant}$*%*". To illustrate, suppose that the tertiary library is equipped with four drives. Under *Staging-25%*, staging is initiated as long as the staging disks can allocate the minimum bandwidth for the staging activity, as the second condition on drive occupancy is necessarily met with the current request occupying one drive. In contrast, *Staging-100%* favors direct access unless all of the drives become engaged.

Since the behavior of the *Staging-Occupancy*$_{Instant}$*%* algorithm depends on the number of requests in the tertiary library, we can model the algorithm by a Markov chain. The appendix gives the derivation and solution of the Markov chains for *Staging-25%*, *Staging-50%*, *Staging-75%*, and *Staging-100%* for a tertiary library with four tape drives. Using a playback time of 8000 s, a staging time of 1500 s, and an overhead time of 200 s (these timings are derived from the workload described in Sect. 4), the average waiting time and utilization produced by the four algorithms are captured in Figs. 2 and 3. We shall postpone the analysis of the algorithms till Sect. 5. For now, it suffices to observe that none of the algorithms consistently outperform the rest. The problem is that, while *Staging-Occupancy*$_{Instant}$*%* is designed to keep a balance between direct access and

[2] In this study, we assume that the disks have a reserved staging area, so we can focus on the retrieval operations without worrying about space availability. We hope to address the issues of storage space management and object migration between secondary and tertiary storages in a future paper.

Table 2. Algorithm parameters

Parameter	Meaning	Default
Occupancy$_{Instant}$*%*	Target instantaneous tertiary drive occupancy	–
Occupancy$_{Average}$*%*	Target average tertiary drive occupancy	50%
ObserveWindow	Observation window to base the adaptation of *Occupancy*$_{Instant}$*%* on	6
Conf%	Confidence level used in modifying *Occupancy*$_{Instant}$*%*	90%

staging, the decisions of the algorithm are governed by the *Occupancy*$_{Instant}$*%* parameter, the target *instantaneous* occupancy of the tertiary drives. However, the *average* occupancy is what determines whether a tertiary library can make good use of its drives without overloading them. Unfortunately, inherent workload variations cause the instantaneous occupancy to fluctuate, making it difficult to fix a value for *Occupancy*$_{Instant}$*%* that will lead to the desired average occupancy. Consequently, it is necessary to design an algorithm that is able to set *Occupancy*$_{Instant}$*%* dynamically.

3.2 Asdac

The *Asdac* algorithm enhances *Staging-Occupancy*$_{Instant}$*%* by automatically deriving an appropriate setting for *Occupancy*$_{Instant}$*%*. *Asdac* continually monitors the average occupancy of the tertiary library. Based on this feedback, the algorithm then adjusts *Occupancy*$_{Instant}$*%* to bring the average occupancy towards *Occupancy*$_{Average}$*%*. The setting of *Occupancy*$_{Average}$*%* is studied in Sect. 5.5. The feedback control process is described below.

At system start-up time, *Asdac* initializes the *Occupancy*$_{Instant}$*%* parameter to 100%. Thereafter, the tertiary drive occupancy observed by each new object request is recorded. If this causes the number of recorded observations to exceed *ObserveWindow*, an algorithm parameter that is discussed in Sect. 5.4, the oldest observation is discarded. The reason for keeping only the *ObserveWindow* most recent observations is to rid *Asdac* of the influence of old and possibly invalid observations, so that it remains sensitive to any changes in workload characteristics. Having recorded the most recent drive occupancy, *Asdac* next revises *Occupancy*$_{Instant}$*%* as necessary. In cases where less than *ObserveWindow* observations are available, *Asdac* is unable to judge whether the observed occupancy is satisfactory, so the current *Occupancy*$_{Instant}$*%* setting is retained. However, should there be *ObserveWindow* observations, *Asdac* will compute the *Conf%* confidence interval for the observed occupancy, assuming that it follows a t-distribution [5], to determine whether *Occupancy*$_{Average}$*%* falls within the interval. If so, the average observed occupancy satisfies *Occupancy*$_{Average}$*%*, so the current *Occupancy*$_{Instant}$*%* setting need not be changed. If *Occupancy*$_{Average}$*%* lies outside the computed confidence interval, *Occupancy*$_{Instant}$*%* is modified using the following formula:

$$\mathit{Occupancy}_{Instant}\%(new) = \mathit{Occupancy}_{Instant}\%(old) \times \frac{\mathit{Occupancy}_{Average}\%}{Average\ Observed\ Occupancy}\ .$$

Moreover, all of the existing occupancy observations are discarded in order for *Asdac* to track the average occupancy effected by the new *Occupancy*$_{Instant}$*%* setting. Having carried out any required adjustment to *Occupancy*$_{Instant}$*%*, *Asdac* then decides on a retrieval mode for the new object request according to the *Staging-Occupancy*$_{Instant}$*%* algorithm.

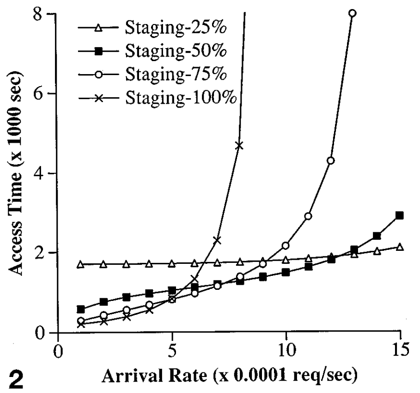
4 Simulation model

As for any algorithm that operates by feedback control, we need to study the efficacy of the *Asdac* algorithm by examining how it steers itself toward the optimal operating point, and whether the algorithm will remain there if the workload is stable. Equally importantly, we need to be sure that *Asdac* can respond quickly to changing workload compositions. Since *Asdac* bases its actions not only on the present load condition, but also on past system states, we could model *Asdac* by a Markov chain where each state is a vector that captures the last *ObserveWindow* system states. Unfortunately, this causes the state space to explode exponentially with *ObserveWindow*, rendering the solution intractable. The alternative of “approximate” modeling, rather than exact modeling, is also unsatisfactory, because we could not be certain about the validity of the approximations. These reasons, together with the need to study *Asdac* under transient workloads, led us to conclude that analytical modeling is not the most appropriate tool for our purpose. We therefore constructed a simulation model of a multimedia storage server to facilitate our evaluation, taking care to capture the detailed operations of the server as faithfully as we could. The simulation model is described below.

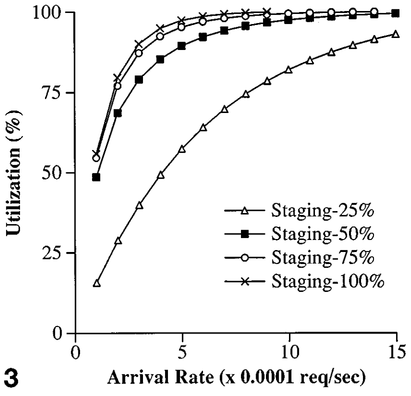
The simulator is constructed after the model in Fig. 4. There are four components: a *Source* that generates object retrieval requests and collects statistics on completed requests; a *Tertiary Library* component and a *Staging Disks* component that model the hardware resources; and a *Server Manager* that coordinates the execution of retrieval requests, including requisition for transfer bandwidths on the tertiary library and staging disks. In this section, we describe how the simulation model captures the details of the database, workload, and various physical resources of a multimedia storage system.

4.1 Database and workload model

Table 3 summarizes the database and workload model parameters that are relevant to this study. Our objective is to simulate a stream of retrievals from various cartridges in the tertiary library. To facilitate this, we populate each cartridge with as many objects as possible. The objects have an average playback rate of $Playback$. The size of each object is uniformly distributed between the range $ObjectSize$.



2



3

Fig. 2. Access time (analytical)

Fig. 3. Tape library utilization (analytical)

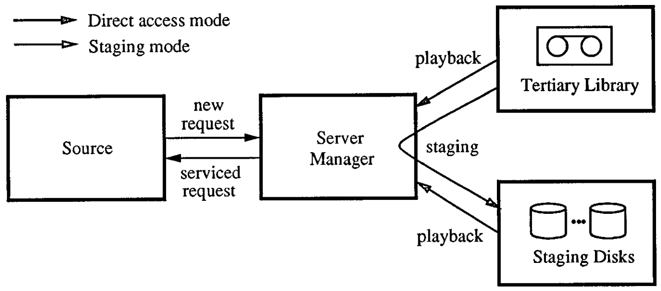


Fig. 4. Multimedia storage system model

Table 3. Database and workload model parameters

Parameter	Meaning	Default
$Playback$	Average playback rate	1.5 Mbits/s
$ObjectSize$	Range of object sizes	[1, 2] GB
$ArrivalRate$	Average request arrival rate	–

In this study, the workload is made up of a series of object requests. Since most of the requests in a multimedia storage system are expected to be targeted at popular objects that are disk-resident, retrievals from the tertiary library are likely to be irregular. For this reason, we adopt an open model, with request arrivals modeled after a Poisson distribution with a mean of $ArrivalRate$. We assume that the objects in the tertiary library are accessed equally rarely.

4.2 Physical resource model

The parameters that specify the physical resources of our simulation model, which consist of a tertiary library and a set of staging disks, are listed together with their default values in Table 4. The CPU and memory are not expected to be a bottleneck at the transfer speed that the tertiary library is capable of achieving, hence, they are left out to simplify the simulation model. Also, instead of treating the staging disks individually, we model them as a single resource with an aggregate bandwidth of $DiskBandwidth$. The rationale is as follows. On one hand, if the staging disks are managed as a RAID, then the individual disks are transparent to the multimedia storage system anyway; the storage system simply sees a high-capacity, high-speed logical disk. On the other hand, if the disks are managed directly by the multimedia storage system, we have a set of data placement and retrieval algorithms that allow the aggregate bandwidth of the staging disks to scale up linearly with the number of disks [18]. Therefore, there is no need to model the individual disks for the purpose of this study.

Table 4. Physical resource model parameters

Parameter	Meaning	Default
$DiskBandwidth$	Aggregate bandwidth of staging disks	18.75 MB/s
$NumCartridges$	Number of cartridges in tertiary library	80
$TapeCapacity$	Storage capacity of a cartridge	14 MB
$NumMagazines$	Number of cartridge magazines	8
$RotationTime$	Magazine rotation time	[0, 2] s
$CartTransfer$	Time to transfer cartridge between magazine and drive	[4, 14] s
$NumTapeDrives$	Number of tape drives	4
$TapeBandwidth$	Max. transfer rate of each tape drive	1 MB/s
$SearchTime$	Cartridge search time	[0, 190] s
$RewindTime$	Time to rewind cartridge	[0, 190] s

The tertiary library that we use in this study is modeled after an EXB-480 tape library. The tertiary library consists of a total of $NumCartridges$ tape cartridges that are evenly divided among $NumMagazines$ magazines, and $NumTapeDrives$ EXB-8505XL drives. Each cartridge has a storage capacity of $TapeCapacity$. The cartridge loading/unloading time is made up of two components: (a) a magazine rotation time that is in the range $RotationTime$, depending on the distance between the required magazine and the cartridge-handling mechanism; and (b) a delay in the interval $CartTransfer$ to transfer the target cartridge between the magazine and a tape drive, as determined by the cartridge's assigned location in the magazine and the position of the tape drive. Once the cartridge has been loaded, the drive takes $SearchTime$ to seek forward to the start of the target object. When the tape is in position, it is retrieved at a maximum rate of $TapeBandwidth$, though the actual retrieval speed may be lower if the object is being accessed directly from the tape, or if the staging disks have limited bandwidth. After the entire object has been retrieved, the cartridge has to be rewound, an operation lasting $RewindTime$. Finally, the cartridge is unloaded.

5 Experiments and results

In this section, our multimedia storage system simulator will be used to evaluate the performance of the $Staging-Occupancy_{Instant}$% and *Asdac* algorithms. The first three experiments are designed to profile the performance of *Asdac* under different kinds of workload and resource compositions. The tertiary library is the bottleneck resource in the first experiment; the second experiment uses shorter staging and playback times, which could result from employing faster storage devices or having objects with a shorter duration or a lower bit rate; in the third experiment, we shift the bottleneck from the tertiary library to the staging disks. The next three experiments provide a sensitivity analysis of the *Asdac* algorithm. We first examine the impact of different settings for *ObserveWindow* and *Occupancy*$_{Average}$*%*, the input parameters of *Asdac*, before investigating its responsiveness by subjecting it to a transient workload.

For comparison purposes, we shall also examine two static tertiary storage retrieval algorithms, *Staging* and *DirectAccess*. The *Staging* algorithm always stages a requested object to disks, even if the staging disks are overloaded and the tertiary library only needs to support sporadic retrievals. In contrast, the *DirectAccess* algorithm insists on direct access all the time, ignoring the existence of the staging disks. These two algorithms are included to highlight any performance gains that can be realized by algorithms which are cognizant of the relative loads on the tertiary library and the staging disks, such as *Staging-Occupancy*$_{Instant}$*%* and *Asdac*.

Since we adopt an open model, the steady-state throughput is the same for all algorithms, and, hence, is not a suitable performance metric. Instead, we choose as the primary performance metric the *access time*, defined as the elapsed time from the submission of an object request to the instant when the first data page of the object arrives at the user terminal. Each experiment was run for 3000 simulated hours, allowing a minimum of 2000 object retrievals. We also verified that the size of the 90% confidence intervals for access time (computed using the batch means approach [19]) was within a few percent of the mean in almost all cases.

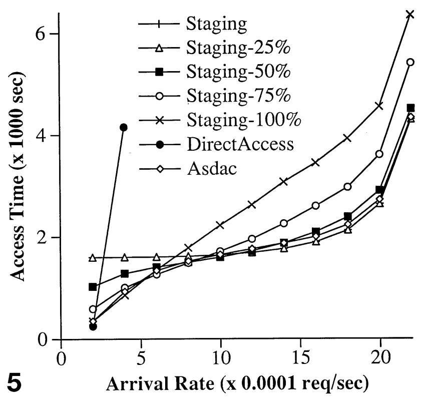
5.1 Baseline experiment

The objective of the baseline experiment is for us to establish an initial understanding of the trade-offs between the various proposed tertiary storage retrieval algorithms. We simulate a “typical” operating environment where the tertiary library plays the role of “cheap-but-slow” storage, and where the staging disks form “expensive-but-fast” storage. While it is true that tape drives like the Ampex DST 600 and 1$''$ Metrum are faster than most disk drives, installations that invest in such high-end tape drives are also likely to acquire high-performance disk arrays, so the relative cost/performance characteristics of the tertiary library and staging disks that we use are realistic [17]. To create the above operating environment, we model a storage system that comprises a tape library with four drives, each of which has a transfer rate of 1 MB/s. The storage system is also equipped with a set of staging disks offering an aggregate bandwidth of 18.75 MB/s. The rest of the resource parameter settings follow the default values in Table 4. As for the workload, the object sizes are uniformly distributed between 1 GB and 2 GB, and the average playback rate is 1.5 Mbits/s. These parameter settings are chosen to model the kind of workloads that can be expected in a video-on-demand system offering a library of MPEG-1 movies. Finally, the *Staging-Occupancy*$_{Instant}$*%* and *Asdac* algorithm parameters are set as in Table 2.

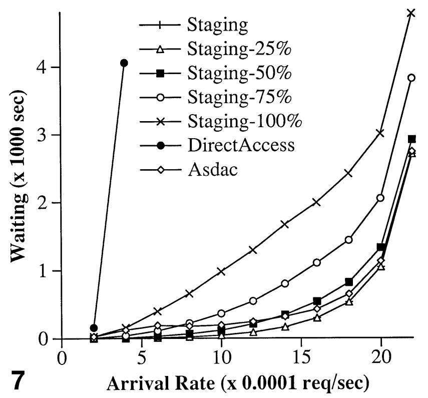
Figure 5 plots the access times produced by *Staging*, *DirectAccess*, *Staging-25%*, *Staging-50%*, *Staging-75%*, *Staging-100%*, and *Asdac*. As the figure shows, *Staging-100%*, *Staging-75%*, *Staging-50%* and *Staging-25%* perform best for request arrival rates of 0.0002–0.0004, 0.0005–0.0008, 0.0009–0.0012, and 0.0013–0.0022 requests/s, respectively. However, none of these algorithms consistently work well. These observations are exactly the same as those derived from our analytical model, described in Sect. 3.1, which confirm the validity of our simulator. *Asdac* manages to track the best performing *Staging-Occupancy*$_{Instant}$*%* algorithm for the entire range of load levels. *DirectAccess* behaves satisfactorily only for a very low arrival rate of 0.0002 requests/s; the performance of this algorithm becomes unacceptable as soon as the load mounts. Finally, *Staging* produces exactly the same access times as *Staging-25%*. These observations clearly show that the choice of tertiary storage retrieval algorithms can have a very significant impact on the access time. We shall explain the behavior of each algorithm in turn with the aid of Figs. 6–8, which give the utilization of the staging disks, the waiting time for tape drives, and the utilization of the tape library.

Let us first examine the *DirectAccess* algorithm. Since this algorithm always insists on the direct access mode, the staging disks are never used, hence, their 0% utilization. *DirectAccess* forces every tape drive to retrieve objects at a playback rate of 1.5 Mbits/s, instead of fully utilizing the transfer bandwidth of the tape drive which, at 1 MB/s, is more than five times faster than the playback rate. With an average object size of 1.5 GB, this means that the transfer time of each object takes 8000 s instead of the minimum of 1500 s. When the arrival rate is as low as 0.0002 requests/s, this produces a tape library utilization of only 40%, so no harmful consequences are observed. As the arrival rate increases, however, the tape library utilization immediately climbs very steeply. This makes the tape library a system bottleneck, and contention for tape drives causes object requests to experience long waiting times, as Fig. 7 indicates. In fact, at an arrival rate of 0.0004 requests/s, the access time is made up almost entirely of tape drive waiting time. Beyond 0.0004 requests/s, no stable access time measures can be obtained, because the workload overwhelms the tape library.

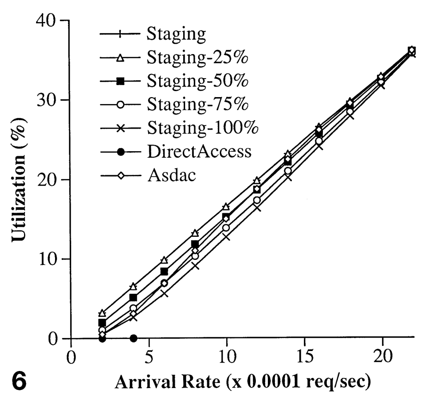
In contrast to *DirectAccess*, *Staging* always stages objects off the tape drives at their maximum transfer rate. This frees the tape drives for subsequent object requests as soon as possible, leading to *Staging*’s lowest tape library utilization rates and waiting times. Unfortunately, minimizing tape drive waiting does not automatically lead to low access times, since *Staging* has to invest in a staging delay that averages 1500 s, as explained above, for each object retrieval. At an arrival rate of 0.0002 requests/s, where the load is light and the tape drive waiting time is relatively low, the staging delay produces a net increase in access time, which explains why *Staging* performs worst. As the load mounts and the tape drive waiting time builds up, however, the staging delay gradually diminishes in significance until, at an arrival



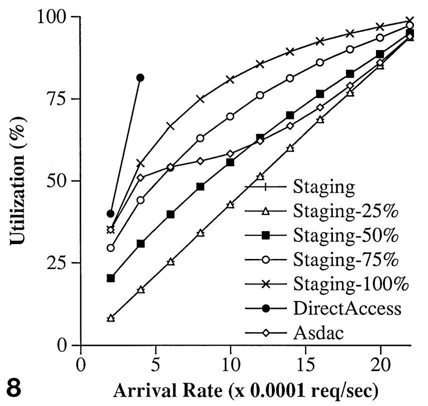
5



7



6



8

Fig. 5. Access time (baseline)

Fig. 6. Disk utilization (baseline)

Fig. 7. Tape drive waiting time (baseline)

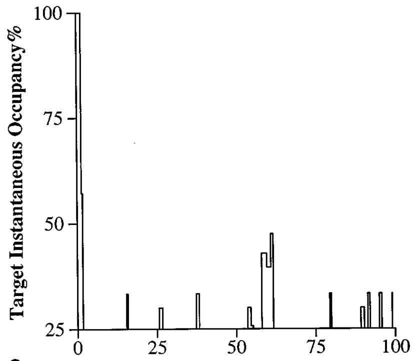
Fig. 8. Tape library utilization (baseline)

rate of 0.0014 requests/s, staging begins to pay off, helping the *Staging* algorithm to dominate all of the other algorithms.

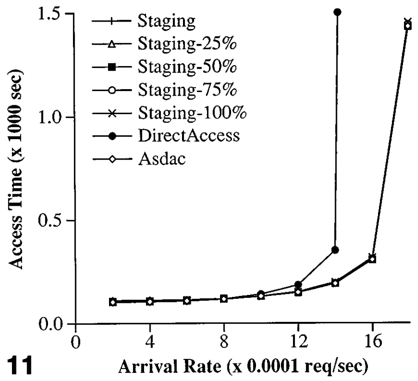
Next, we turn our attention to the *Staging-100%* algorithm. *Staging-100%* allows up to three of the four tape drives to be deployed in direct access mode. When the arrival rate is low, this enables almost all of the requested objects to be accessed directly, since the number of retrieval requests at the tape library is low anyway. Thus, *Staging-100%* effectively mimics *DirectAccess* under light loads. When the load is heavy and a backlog of requests begins to form, one of the tape drives will be staging at its maximum transfer rate in order to get to the next request in the queue in the shortest possible time. Moreover, upon completing the requests that are currently being served, the three drives that are in direct access mode will switch to staging mode to cope with the waiting requests. The tape drives will only revert to direct access mode when there are again less than four requests (including those being served) at the tape library, indicating that it has caught up with the backlog of object requests. This ability to dynamically switch tape drives between staging and direct access modes is the reason that *Staging-100%* degrades much more gracefully than *DirectAccess* as the arrival rate increases. However, since as many as three drives can be deployed in direct access mode, *Staging-100%* runs a high risk of allowing request backlogs to build up, thus degrading access times. This is why *Staging-100%* performs considerably worse than *Staging* under heavy loads, despite *Staging-100%*'s ability to switch retrieval mode dynamically.

Compared to *Staging-100%*, *Staging-75%*, *Staging-50%* and *Staging-25%* set increasingly stricter limits on the number of tape drives that can be in direct access mode. Consequently, the three algorithms behave less like *DirectAccess* and more like *Staging*. In fact, *Staging-25%* exhibits exactly the same behavior as *Staging*, because *Staging-25%* resorts to accessing an object directly from a tape drive only if the available bandwidth of the staging disks is below the object's playback rate. Since disk bandwidth is abundant in this experiment, *Staging-25%* is always able to perform staging, hence, *Staging-25%* is equivalent to *Staging* here. Therefore, *Staging-100%*, *Staging-75%*, *Staging-50%* and *Staging-25%* represent a range of trade-offs between staging delay and tape drive waiting. Since tape drive waiting time is jointly determined by the setting of $\mathit{Occupancy}_{Instant}\%$ and the system load, the most appropriate trade-off point is not fixed. This testifies to the need to automatically derive the best $\mathit{Occupancy}_{Instant}\%$ setting, which is exactly what *Asdac* aims to achieve.

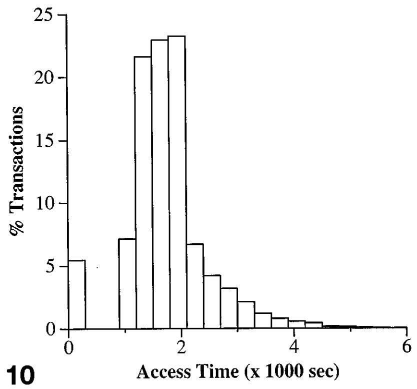
Finally, we examine the *Asdac* algorithm. In order to understand how *Asdac* adapts itself to the workload, we examine Fig. 9, which traces the $\mathit{Occupancy}_{Instant}\%$ setting of *Asdac* over the initial 100 h of operation at an arrival rate of 0.0012 requests/s. At system start-up time, *Asdac* initializes $\mathit{Occupancy}_{Instant}\%$ to a default value of 100%. However, after *ObserveWindow* = 6 requests, *Asdac* immediately detects that a 100% $\mathit{Occupancy}_{Instant}\%$ produces a backlog of requests at the tape library. This causes *Asdac* to reduce $\mathit{Occupancy}_{Instant}\%$ to 60%, then to 25% after another iteration. Thereafter, $\mathit{Occupancy}_{Instant}\%$ re-



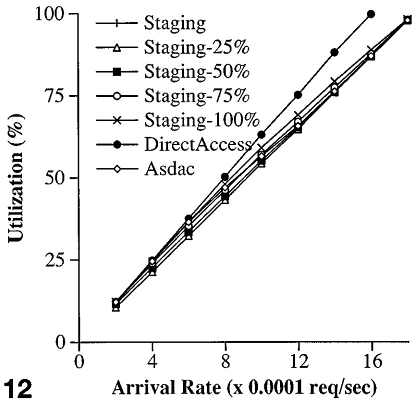
9



11



10



12

Fig. 9. *Occupancy*$_{Instant}$*%* (baseline)

Fig. 10. Access time distribution (baseline)

Fig. 11. Access time (small objects)

Fig. 12. Tape library utilization (small objects)

mains at 25% most of the time, occasionally rising up to 50% in response to workload fluctuations before settling back at 25%. This leads to an average *Occupancy*$_{Instant}$*%* setting of 35%, which enables *Asdac* to behave like the best *Staging-Occupancy*$_{Instant}$*%* algorithm for that arrival rate. In fact, *Asdac* manages to find the most appropriate *Occupancy*$_{Instant}$*%* setting over the entire range of arrival rates that we tested, gradually reducing its *Occupancy*$_{Instant}$*%* from 100% to 25% as the arrival rate increases. The access time distribution produced by *Asdac* is typified by Fig. 10, which shows the distribution at 0.0012 requests/s. The figure shows that the access times concentrate around the average of 1750 s, as 70% of the access times fall within the range of 1200–2100 s. Furthermore, about 90% of the access times are below 2400 s.

To summarize the results of this experiment, we can derive the following conclusions about situations where the workload comprises mainly large object retrievals and the tertiary library is the system bottleneck: First, *DirectAccess* performs well only if the tertiary library is used very infrequently; the access times produced by this algorithm are unacceptable even for moderate system loads. Second, *Staging* dominates the rest of the algorithms under heavy retrievals, but suffers from (relatively) long staging delays under light loads. Third, *Staging-Occupancy*$_{Instant}$*%* delivers good access times under light load conditions if a high *Occupancy*$_{Instant}$*%* is chosen, whereas a low *Occupancy*$_{Instant}$*%* helps the algorithm to cope with heavy loads better. However, no single *Occupancy*$_{Instant}$*%* setting works well for all load conditions. Finally, the *Asdac* algorithm is capable of finding the right *Occupancy*$_{Instant}$*%* setting very quickly, consistently achieving low access times by balancing between direct access and staging.

5.2 Small objects/high transfer speeds

In the baseline experiment, the workload was made up of retrieval requests for large objects that are typical of full-length movies. For the second experiment, we maintain the algorithm, resource, and workload parameters of the baseline experiment, but we reduce the object sizes to a range of only 4–12 MB. One objective of this experiment is to profile the relative performance of the proposed tertiary storage retrieval algorithms in systems that deal with a variety of multimedia objects, like video/audio clips and still images. An example would be a news-on-demand system. Another objective is to investigate the impact of a larger array of staging disks and a higher end tape library, such as a RAPID-tape array that stripes data across multiple drives like a RAID. These storage devices boast very fast transfer speeds that reduce object transfer times, which has the effect of making objects appear “smaller”.

Figures 11 and 12 plot the access time and the utilization of the tape library produced by the various algorithms. As Fig. 11 shows, the access time differences between *Staging, Staging-25%, Staging-50%, Staging-75%, Staging-100%,* and *Asdac* are hardly noticeable. Even the performance gap between *DirectAccess* and the rest of the algorithms is much narrower now. The reason is that, with an average object size of 8 MB, the staging time and playback time are only 8 s and 43 s, respectively. In comparison, the fixed overhead of the tape library averages 206 s (cartridge loading 8 s, tape search 95 s, tape rewind 95 s, and cartridge unloading 8 s). Consequently, the access time penalty that staging incurs is insignificant, which explains why all of the algorithms are almost equally good at low loads. At higher arrival rates, however, *DirectAccess*'s policy of accessing the tape drives directly again exacerbates the contention that object requests experience at the tape library (see Fig. 12), resulting in longer access times. As for *Asdac* and the four *Staging-Occupancy*$_{Instant}$*%* algorithms, their *Occupancy*$_{Instant}$*%* settings do not matter here because the short staging and playback times allow the algorithms to quickly switch to staging mode once there is a backlog of retrieval requests at the tape library.

In summary, we conclude that the static *Staging* algorithm is satisfactory if the workload comprises mainly small object retrievals and the tertiary library is the system bottleneck. However, *Asdac* is still the algorithm of choice.

5.3 Staging disk bottleneck

For the first two experiments, we have made the tertiary library the bottleneck resource of the system. As explained earlier, this is because the staging disks are normally faster than the tertiary drives. However, if the disks are not dedicated to staging operations, then the disks could conceivably become the bottleneck instead. An arrangement that will limit the bandwidth available for staging is where the disks that support staging operations are also used to cache the popular objects, so most of the disk bandwidth is consumed by the retrieval of these popular objects. To explore the implication of having a bottleneck at the disks, we lower the disk bandwidth to 1 MB/s without changing the rest of the resource parameters. The workload and algorithm parameters also remain at their settings in the baseline experiment.

Figures 13–16 plot the access time, staging disk utilization, tape drive waiting time, and tape library utilization values, respectively. The figures show that the various algorithms exhibit very different behaviors here than they did in the baseline experiment. Whereas *Staging* performed badly at low loads but dominated under heavy loads previously, it now produces much higher access times than the other algorithms at all arrival rates. The reason is that, with a small transfer bandwidth, the staging disks quickly become saturated (see Fig. 14) by staging operations. As a result, object requests have to wait for a long time before they get to use the staging disks. In the meantime, the object requests hold on to their assigned tape drives, causing the utilization of the tape library to increase. This, in turn, leads new object requests to experience long tape drive waiting times, as Fig. 15 shows. Consequently, *Staging* has the worst access times. In contrast, *DirectAccess* relies exclusively on the tape drives for its retrievals. Since the tape drives have a higher aggregate bandwidth than the staging disks, *DirectAccess* destabilizes later than *Staging*, as indicated in Fig. 13.

Having examined the two static algorithms, we now turn our focus to *Asdac* and the four *Staging-Occupancy*$_{Instant}$*%* algorithms. At low arrival rates, the relative performance between the four *Staging-Occupancy*$_{Instant}$*%* algorithms are the same as before, with *Staging-25%* and *Staging-50%* suffering more from staging delays than *Staging-75%* and *Staging-100%*. Under heavy loads where the staging disks and, in turn, the tape library approach saturation, the *Staging-Occupancy*$_{Instant}$*%* algorithms degenerate to a simple policy of staging whenever the staging disks have enough bandwidth, and accessing from the tape drives directly otherwise. In other words, the setting of *Occupancy*$_{Instant}$*%* has a negligible impact. This is why all four algorithms deliver the same access times. Due to their ability to exploit the staging disks, the *Staging-Occupancy*$_{Instant}$*%* algorithms outperform *DirectAccess*. Moreover, since they can bypass the staging disks when they get overloaded, the *Staging-Occupancy*$_{Instant}$*%* algorithms are superior to *Staging*. Finally, we observe that *Asdac* again tracks the best *Staging-Occupancy*$_{Instant}$*%* algorithm, producing the shortest access times.

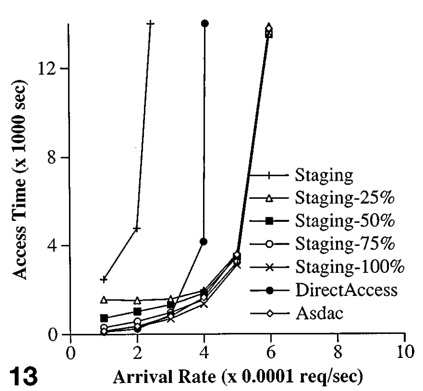
To summarize, we conclude that, when the disks have limited bandwidth for staging operations, *DirectAccess* is superior to *Staging*, but *Staging-100%* gives the best performance. Again, *Asdac* is able to set its *Occupancy*$_{Instant}$*%* appropriately to exploit both the staging disks and the tape library.

5.4 Impact of ObserveWindow

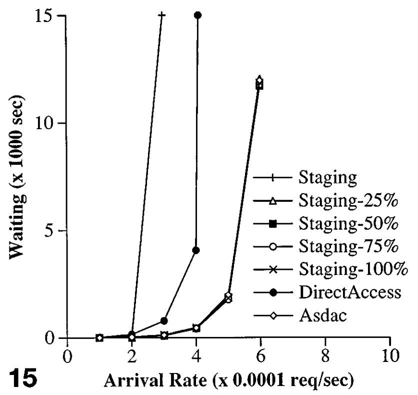
In the previous experiments, the *ObserveWindow* parameter of the *Asdac* algorithm has been set to 6. We now vary *ObserveWindow* in order to evaluate the sensitivity of the *Asdac* algorithm. The rest of the parameter settings are the same as for the baseline experiment. The resulting access times are given in Fig. 17. We also plot in Fig. 18 the standard deviation of *Asdac*'s target *Occupancy*$_{Instant}$*%* setting.

With *ObserveWindow* = 1, *Asdac* decides the transfer mode for a retrieval request based solely on the tape drive occupancy that the request observes when it first arrives. This allows the setting of *Occupancy*$_{Instant}$*%* to fluctuate with variations that are inherent in the workload, which is why the standard deviation of *Occupancy*$_{Instant}$*%* is highest at *ObserveWindow* = 1. We also observe in Fig. 18 that the standard deviation rises with the arrival rate initially, peaking at an arrival rate of 0.0006 requests/s before declining with further increases in the arrival rate. This is because *Occupancy*$_{Instant}$*%* is bounded from above by 100% at low arrival rates and bounded from below by 25% at high arrival rates, so *Occupancy*$_{Instant}$*%* fluctuates most freely at 0.0006 requests/s when the average *Occupancy*$_{Instant}$*%* (not shown) is right in between 25% and 100%. The high variations in *Occupancy*$_{Instant}$*%* means that it is very frequently set either too high or too low, although the average levels are about right. Consequently, *Asdac* produces the worst access times at this *ObserveWindow*.

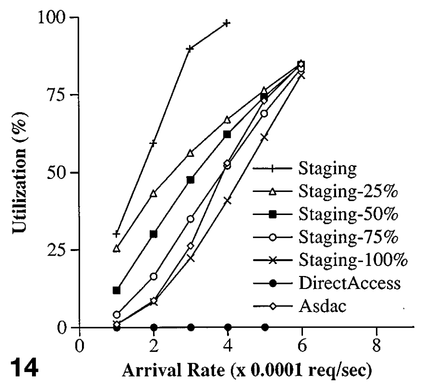
As *ObserveWindow* is raised, *Asdac* becomes less sensitive to temporary load variations. As a result, *Asdac* is better able to hold the *Occupancy*$_{Instant}$*%* parameter at its best level, thus achieving shorter access times. At $ObserveWin-dow$ = 6, *Asdac*'s *Occupancy*$_{Instant}$*%* setting becomes stable enough that further increases in *ObserveWindow* produces only inconsequential reductions in access time. Since



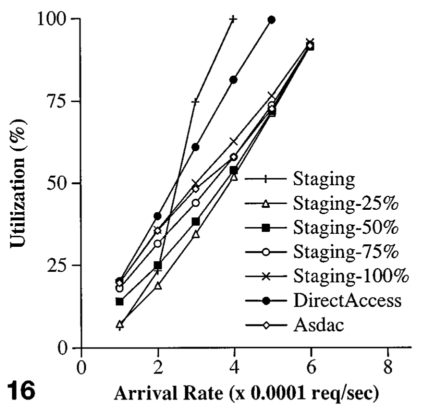
13



15



14



16

Fig. 13. Access time (staging)

Fig. 14. Disk utilization (staging)

Fig. 15. Tape drive waiting time (staging)

Fig. 16. Tape library utilization (staging)

too large an *ObserveWindow* impedes *Asdac*'s ability to quickly adapt to changes in workload characteristics, we conclude that a *ObserveWindow* of 6 suffices.

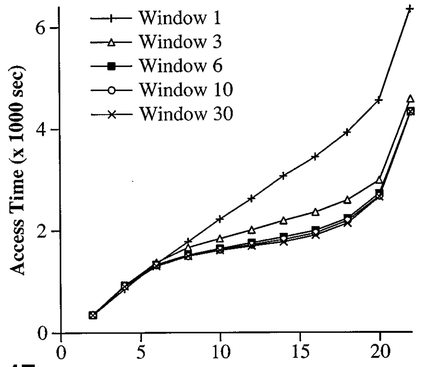
5.5 Impact of Occupancy$_{Average}$%

Besides *ObserveWindow*, *Occupancy*$_{Average}$*%* is the other input parameter that may affect the performance of *Asdac*. We now vary the value of this parameter, which has been kept at 50% in previous experiments. The rest of the parameters remain at their settings in the baseline experiment. The resulting access times are plotted in Fig. 19. The figure shows that an *Occupancy*$_{Average}$*%* of 50% is the best. Intuitively, this is because inherent workload variations cause the ideal (instantaneous) tape drive occupancy to fluctuate between 0% and 100%, so an *Occupancy*$_{Average}$*%* of 50% gives *Asdac* the most flexibility in adjusting to the variations. However, *Occupancy*$_{Average}$*%* values of 25% and 75% are not much worse. The reason is that *Asdac*'s feedback control mechanism can largely compensate for suboptimal *Occupancy*$_{Average}$*%* settings. To illustrate, let us refer to the formula for computing *Occupancy*$_{Instant}$*%*, given near the end of Sect. 3.2. If *Occupancy*$_{Average}$*%* is too low, *Occupancy*$_{Instant}$*%* will initially be low too. This will soon lead to a small average observed occupancy, which in turn will force *Occupancy*$_{Instant}$*%* back up. Adjustments to overly high *Occupancy*$_{Average}$*%* values are made in a similar fashion. Consequently, we recommend setting *Occupancy*$_{Average}$*%* to 50%, though that is not crucial.

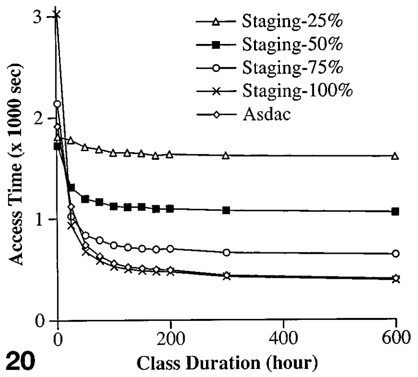
5.6 Transient workload

Our last experiment is designed to find out how quickly *Asdac* can adapt to workload changes. This is achieved by retaining all of the algorithm, resource, and workload parameter settings of the baseline experiment, except for the request arrival rate that now alternates periodically between 0.0002 and 0.0022 requests/s. The duration between workload changes, *ClassDuration*, is varied. Thus, there are two workload classes which we shall denote by Light and Heavy. As explained in Sect. 3 and demonstrated in the baseline experiment, *Staging-100%* produces lower access times for the Light class, whereas *Staging-25%* is more suitable for the Heavy class. Consequently, this fluctuating workload serves to test *Asdac*'s ability to steer itself towards *Staging-100%* or *Staging-25%* as the workload changes. For this experiment, we will only show the access times produced by *Staging-25%*, *Staging-50%*, *Staging-75%*, *Staging-100%* and *Asdac*. *Staging* is left out, since it has been shown to give the same performance as *Staging-25%* for the two workload classes, while *DirectAccess* is not able to handle the Heavy class.

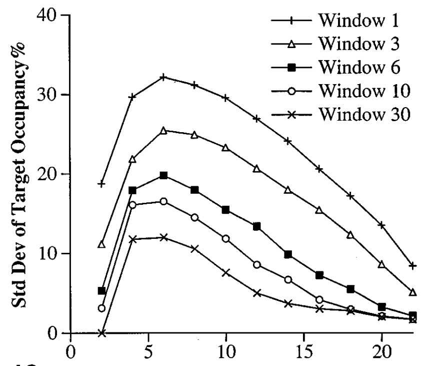
Figures 20 and 21 plot the access times for the Light class and Heavy class. At a ClassDuration of 2 h, an object request experiences at least one class change during its lifetime, as each request lasts more than 2 h on average (average object size/playback rate = 1.5 GB/1.5 Mbits per s = 8000 s). Therefore, no object request can be clearly identified as belonging to either class, although, for accounting purposes, we attribute the access time of a request to the class that is active when the request first arrives. Consequently, it is no



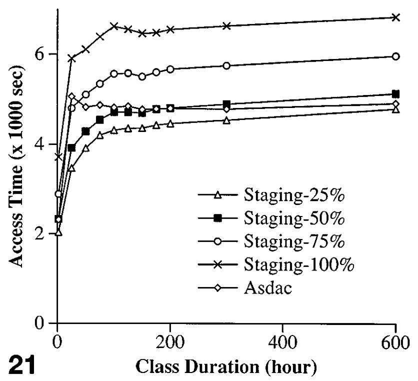
17



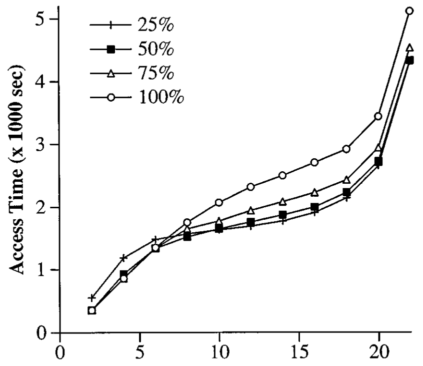
20



18



21



19

Fig. 17. Access time (*ObserveWindow*)

Fig. 18. Standard deviation of *Occupancy*$_{Instant}$% (*ObserveWindow*)

Fig. 19. Access time (*Occupancy*$_{Average}$%)

Fig. 20. Light access time (change)

Fig. 21. Heavy access time (change)

surprise that both Fig. 20 and Fig. 21 register similar access times at this ClassDuration.

As ClassDuration increases and class boundaries become more distinct, we expect the access time of the two classes to move gradually toward the levels observed in the baseline experiment. This is indeed the case for the Light class, as Fig. 20 shows. However, Fig. 21 reveals that, except for *Staging-25%*, the Heavy class access times produced by the algorithms grow faster than anticipated between ClassDurations of 25 and 150 h, peaking at ClassDuration = 100 h before dropping back down. We were surprised by this trend initially. After monitoring the behavior of the algorithms closely, we discovered that the unexpected trend in the access time of the Heavy class is due to performance deterioration induced by class transitions. To simplify our explanation, we shall first focus on *Staging-100%*. When the Light class is active, *Staging-100%* allows up to three of the four tape drives to be employed for direct accesses. This causes a backlog of object requests to form suddenly when the workload changes to the Heavy class. Since the tape library takes a while to complete the direct accesses at the tape drives before switching them to staging mode, access time naturally deteriorates when the workload switches from the Light class to the Heavy class. This deterioration is worst at a ClassDuration of 100 h, which is just about enough time for the tape library to catch up with the backlog of object requests. Beyond 100 h, the tape library begins to reap the benefits of having tuned to the Heavy class, and access time improves as a result. This explains the trend in *Staging-100%*'s access times between ClassDurations of 25 and 150 h. As for the rest of the algorithms, *Staging-75%*, *Staging-50%* and *Staging-25%* suffer less from class transitions, because these algorithms employ fewer of the tape drives in direct access mode. Finally, *Asdac* is also adversely affected by class transitions, as evident from the sharp rise in its access time in Fig. 21. However, beyond a ClassDuration of 25 h, *Asdac*'s automatic *Occupancy*$_{Instant}$*%* tuning mechanism starts to take effect, bringing the access times of both classes to satisfactory levels. This experiment shows that *Asdac* is able to adapt to a given workload, even if its characteristics remain stable for only 15–20 times the average object retrieval duration after each workload change.

6 Conclusions

In this paper, we have focused on the problem of retrieving data from a tertiary library. This problem will arise in storage systems that need to deal with very large volumes of data, making a purely disk-based implementation uneconomical. Systems that are likely to require tertiary storage include those that support multimedia applications like interactive multimedia education [2] and news on demand [16]. These applications not only need to maintain large collections of objects, but the objects themselves also tend to be huge in size. For example, a 2-h MPEG-1 movie can easily occupy 1.5 GB of storage [9]. Consequently, accessibility of objects that reside in the tertiary library plays an important role in determining the performance of a multimedia storage system. One measure of accessibility is the *access time*, defined as the elapsed time between the submission of a retrieval request and the instant when the first page of data arrives at the user terminal. At present, most existing storage systems support only *Staging*, which requires an object to be transferred to disks before it can be accessed. An alternative approach that has been studied is *DirectAccess*, where objects are read off the tertiary library directly. Our study demonstrated that neither *Staging* nor *DirectAccess* produce satisfactory access times consistently. Therefore, there is a need for dynamic approaches that combine *Staging* and *DirectAccess* according to the resource composition of a storage system.

To explore the efficacy of dynamically combining *Staging* and *DirectAccess*, we first propose a family of tertiary storage retrieval algorithms, called *Staging-Occupancy*$_{Instant}$*%*. These algorithms decide on a retrieval mode for an object request based on the occupancy of the read/write drives in the tertiary library at the time of the request's arrival. If the occupancy is above the *Occupancy*$_{Instant}$*%* parameter, then the requested object is staged; otherwise the object is accessed from a tertiary drive directly. Analytical modeling, as well as simulation experiments using different workloads and resource configurations reveal that there are always some *Occupancy*$_{Instant}$*%* settings that enable *Staging-Occupancy*$_{Instant}$*%* to perform at least as well as *Staging* and *DirectAccess*. Unfortunately, no single *Occupancy*$_{Instant}$*%* setting is optimal for all situations. On one hand, a low *Occupancy*$_{Instant}$*%* usually produces shorter access times at low loads, because staging delays are avoided by accessing objects directly from the tertiary drives, which are free most of the time. On the other hand, a high *Occupancy*$_{Instant}$*%* works better when the tertiary library is heavily accessed, as objects are read off the tertiary drives as fast as possible, thus speeding up turnaround time at the tertiary drives.

In order to provide efficient tertiary storage retrieval under different system configurations, we next introduce an *Asdac* algorithm. *Asdac* augments $Staging - Occupancy_{Instant}\%$ with a mechanism that automatically derives the most appropriate *Occupancy*$_{Instant}$*%* value based on feedback on the observed tertiary drive occupancy: If the observed occupancy is low, then *Asdac* raises $Occupancy_{Instant}\%$ to allow more direct accesses, so as to lower access time. Conversely, if the observed occupancy is high, then *Occupancy*$_{Instant}$*%* is lowered to favor staging, resulting in faster turnaround at the tertiary drives. Experiments show that *Asdac* is very effective for a wide range of workload and resource configurations, consistently reaching the optimal *Occupancy*$_{Instant}$*%* setting in all cases. Moreover, *Asdac* achieves this quickly enough, so that it works well even if the workload composition changes over time. Therefore, we conclude that *Asdac* should be very useful for managing data retrievals from the tertiary library of a multimedia storage system.

For future work, we intend to integrate storage space management with the data retrieval algorithm presented in this paper. Specifically, we will look into ways to determine which objects should reside on disks, and which objects can be held in tertiary library. We also plan to study issues relating to object migration across secondary and tertiary storages. Once these issues have been resolved, we will build a complete hierarchical storage subsystem for the multimedia storage server that we have implemented.

Appendix: Modeling *Staging-Occupancy*$_{Instant}$*%* analytically

In this appendix, we describe how we model the $Staging-Occupancy_{Instant}\%$ algorithm analytically. To simplify the model, we assume that the tertiary library is the only bottleneck resource in the multimedia storage system, and that all other resources have abundant capacities, and hence need not enter into the model. This means that the disks always have sufficient space and bandwidth for staging and the subsequent playback operations. With only the tertiary library to consider, the behavior of *Staging-Occupancy*$_{Instant}$*%* is entirely determined by the number of requests in the library. We can therefore model the algorithm by a Markov chain. Before we derive the Markov chain, let us first introduce some notation:

$play$	: playback time
$stage$	: staging time
$overhead$	: time for cartridge loading/unloading and search/rewind
λ	: request arrival rate/system throughput
$\mu_p = \frac{1}{overhead + play}$	: service rate for direct access
$\mu_s = \frac{1}{overhead + stage}$	: service rate for staging
P_i	: probability of having i requests in the tertiary library
W	: average waiting time for playback to begin

For a tertiary library with four tape drives, *Staging-100%* can be approximated by the following Markov chain:

$$\begin{tikzcd}
0 \arrow[r, bend left, "\lambda"] & 1 \arrow[l, bend left, "\mu_p"] \arrow[r, bend left, "\lambda"] & 2 \arrow[l, bend left, "2\mu_p"] \arrow[r, bend left, "\lambda"] & 3 \arrow[l, bend left, "3\mu_p"] \arrow[r, bend left, "\lambda"] & 4 \arrow[l, bend left, "3\mu_p+\mu_s"] \arrow[r, bend left, "\lambda"] & \bullet\bullet\bullet \arrow[l, bend left, "3\mu_p+\mu_s"]
\end{tikzcd}$$

where the states capture the number of requests in the tertiary library. From state 5 onwards, the system will always have four active requests – three being served in direct access mode and one in staging mode – after a request arrival or completion, hence, the number of requests in the system increases at a rate of λ and decreases at $3\mu_p + \mu_s$. At state 4, the system can either be left with three requests in direct access mode, or two requests in direct access mode plus one request in staging mode when a request departs. However, we aggregate the two possibilities with a single transition to state 3 with a rate of $3\mu_p + \mu_s$, in order to keep the Markov chain simple enough to derive from it a closed form solution. This approximation is reasonable, because the purpose of the Markov chain is to demonstrate the crossover between different *Staging-Occupancy*$_{Instant}$*%* algorithms, so the operating level of interest would push the system beyond state 3 most of the time. Moreover, there is a good chance that the departing request is the one in staging mode anyway, as μ_s is expected to be considerably higher than μ_p. The same approximation applies to the transitions between states 0 and 3. As shown in Figs. 2 and 5, the resulting Markov chain still captures the qualitative trade-offs between the *Staging-Occupancy*$_{Instant}$*%* algorithms.

The above Markov chain yields the following solution:

$$P_i = \begin{cases} \frac{1}{i!}(\frac{\lambda}{\mu_p})^i P_0 & \text{for } 1 \leq \text{i} \leq 2 \\ \frac{1}{3!}(\frac{\lambda}{\mu_p})^3(\frac{\lambda}{3\mu_p + \mu_s})^{i-3} P_0 & \text{for i} \geq 3 \end{cases} ,$$

$$P_0 = (1 + \frac{\lambda}{\mu_p} + \frac{1}{2!}(\frac{\lambda}{\mu_p})^2 + \frac{1}{3!}(\frac{\lambda}{\mu_p})^3(\frac{3\mu_p + \mu_s}{3\mu_p + \mu_s - \lambda}))^{-1} ,$$

$$W = \sum_{i=0}^{\infty} P_i \times w_i ,$$

where w_i is the waiting time that a new request can expect when there are already i requests in the tertiary library. When there are up to three requests in the library, a new request does not need to wait for a free tape drive. Since *Staging-100%* allows up to three of the four drives to be used for direct access, the delay experienced by a new request is simply $overhead$ with two or fewer existing requests in the library. With three existing requests, the waiting time could be either $overhead$ or $overhead$+$stage$, depending on whether one of the existing requests is already doing staging; on the average, the waiting time is $\frac{4\times overhead+stage}{4}$. When there are $i > 3$ existing requests, a new request has to wait for $(i - 3)$ requests to leave the library before it is assigned a tape drive. The expected duration for this wait is $\frac{i\ -\ 3}{3\mu_p+\mu_s}$. After seizing a tape drive, the new request goes through a further $\frac{4\times overhead+stage}{4}$ delay before playback begins. Therefore,

$$w_i = \begin{cases} overhead & \text{for } 0 \leq \text{i} \leq 2 \\ \frac{i\ -\ 3}{3\mu_p+\mu_s} + \frac{4\times overhead+stage}{4} & \text{for i} \geq 3 \end{cases} .$$

The behavior of the other *Staging-Occupancy*$_{Instant}$*%* algorithms can be modeled using the same method. We will simply present their solutions here:

Staging-75%:

$$P_i = \begin{cases} \frac{1}{i}(\frac{\lambda}{\mu_p})^i P_0 & \text{for } 1 \leq \text{i} \leq 2 \\ \frac{1}{2}(\frac{\lambda}{\mu_p})^2(\frac{\lambda}{2\mu_p+\mu_s})(\frac{\lambda}{2\mu_p+2\mu_s})^{i-3} P_0 & \text{for i} \geq 3 \end{cases}$$

$$P_0 = (1+\frac{\lambda}{\mu_p}+\frac{\lambda^2}{2\mu_p^2}+\frac{\lambda^3}{2\mu_p^2(2\mu_p+\mu_s)} \times \frac{2\mu_p+2\mu_s}{2\mu_p+2\mu_s\ -\ \lambda})^{-1}$$

$$w_i = \begin{cases} overhead & \text{for } 0 \leq \text{i} \leq 1 \\ \frac{3\times overhead+stage}{3} & \text{for i = 2} \\ \frac{i\ -\ 3}{2\mu_p+2\mu_s} + \frac{4\times overhead+2\times stage}{4} & \text{for i} \geq 3 \end{cases}$$

Staging-50%:

$$P_i = \begin{cases} \frac{\lambda}{\mu_p} P_0 & \text{for i = 1} \\ \frac{\lambda^2}{\mu_p(\mu_p+\mu_s)} P_0 & \text{for 1 = 2} \\ \frac{\lambda^3}{\mu_p(\mu_p+\mu_s)(\mu_p+2\mu_s)}(\frac{\lambda}{\mu_p+3\mu_s})^{i-3} P_0 & \text{for i} \geq 3 \end{cases}$$

$$P_0 = (1 + \frac{\lambda}{\mu_p} + \frac{\lambda^2}{\mu_p(\mu_p + \mu_s)} + \frac{\lambda^3}{\mu_p(\mu_p + \mu_s)(\mu_p + 2\mu_s)} \times \frac{\mu_p + 3\mu_s}{\mu_p + 3\mu_s\ -\ \lambda})^{-1}$$

$$w_i = \begin{cases} \frac{(i+1)\times overhead+i\times stage}{i+1} & \text{for } 0 \leq \text{i} \leq 2 \\ \frac{i\ -\ 3}{\mu_p+3\mu_s} + \frac{4\times overhead+3\times stage}{4} & \text{for i} \geq 3 \end{cases}$$

Staging-25%:

$$P_i = \begin{cases} \frac{1}{i!}(\frac{\lambda}{\mu_s})^i P_0 & \text{for } 1 \leq \text{i} \leq 2 \\ \frac{1}{3!}(\frac{\lambda}{\mu_s})^3(\frac{\lambda}{4\mu_s})^{i-3} P_0 & \text{for } 1 \geq 3 \end{cases}$$

$$P_0 = (1 + \frac{\lambda}{\mu_s} + \frac{1}{2!}(\frac{\lambda}{\mu_s})^2 + \frac{1}{3!}(\frac{\lambda}{\mu_s})^3(\frac{4\mu_s}{4\mu_s - \lambda}))^{-1}$$

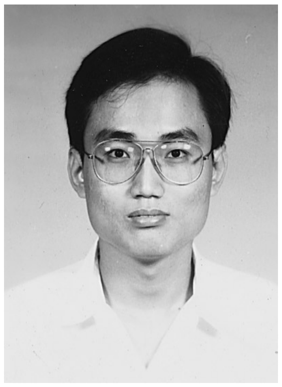
$$w_i = \begin{cases} overhead + stage & \text{for } 0 \leq \text{i} \leq 2 \\ \frac{i\ -\ 3}{4\mu_s} + overhead + stage & \text{for i} \geq 3 \end{cases}$$

Acknowledgement. The author would like to thank Desai Narasimhalu and the reviewers for their helpful comments on an earlier version of this paper.

References

1. Asia Computer Weekly, (1994) December 5–11
2. Adam JA (1993) Interactive multimedia: applications, implications, IEEE Spectrum. 30(3):24–31
3. Bestavros A (1995) Demand-based Document Dissemination for the World Wide Web. Technical Report TR-95-003, CS Dept., Boston University, Boston, Mass.
4. Collins B, Devaney M, Kitts D (1988) Profiles in mass storage: a tale of two systems. In: Digest of Papers, Ninth IEEE Symp. on Mass Storage Systems, October 1988, pp 61–67
5. Devore JL (1991) Probability and Statistics for Engineering and the Sciences. Brooks/Cole Pub. Co., Pacific Grove, Calif.
6. Digest of Papers, Ninth IEEE Symp. on Mass Storage Systems, October 1988. Ed: Friedman KD, Monterey, Calif.
7. Digest of Papers, Tenth IEEE Symp. on Mass Storage Systems, May 1990. Ed: Friedman KD, Monterey, Calif.

8. Foglesong J et al. (1990) The Livermore distributed storage system: implementation and experiences. In: Digest of Papers, Tenth IEEE Symp. on Mass Storage Systems, May 1990, pp 18–25
9. Gall D (1991) MPEG: A video compression standard for multimedia applications, Commun ACM, 34(4):46–58
10. Ghandeharizadeh S, Shahabi C (1994) Multimedia repositories, personal computers, and hierarchical storage systems. Proc. of the ACM Multimedia Conf., San Francisco, Calif., pp 407–416
11. Haas LM, Carey MJ, Livny M (1993) Tapes hold data, too: challenges of tuples on tertiary store. In: Proc. of the ACM SIGMOD Conf. Washington D.C. 413–417
12. Hogan C et al. (1990) The Livermore distributed storage system: requirements and overview. In: Digest of Papers, Tenth IEEE Symp. on Mass Storage Systems, May 1990, pp 6–17
13. Kenley GG (1990) An architecture for a transparent networked mass storage system. In: Digest of Papers, Tenth IEEE Symp. on Mass Storage Systems, May 1990, pp 160–167
14. Kienzle MG, Dan A, Sitaram D, Tetzlaff W (1995) Using tertiary storage in video-on-demand servers. In: Proc. of the IEEE COMPCON. San Francisco, Calif., pp 225–233
15. TDC. Little, Venkatesh D (1993) Popularity-based assignment of movies to storage devices in a video-on-demand system. In: Proc. of the 4th Int. Workshop on Network and Operating System Support for Digital Audio and Video. Lancaster, England, pp 213–224
16. Miller G, Baber G, Gilliland M (1993) News on demand for multimedia networks. In: Proc. of the ACM Multimedia Conf. Anaheim, Calif., pp 383–392
17. Myllymaki J, Livny M (1995) Disk-tape joins: synchronizing disk and tape access. In: Proc. of the ACM SIGMETRICS Conf. Ottawa, Ontario, pp 279–290
18. Pang H (1995) Data placement and retrieval in a distributed multimedia storage system. Technical Report, Institute of Systems Science, National University of Singapore, Singapore
19. Sargent R (1976) Statistical analysis of simulation output data. In: Proc. of the 1976 Symp. on Simulation of Computer Systems. Boulder, Colorado
20. Thanhardt E, Harano G (1988) File migration in the NCAR mass storage system. In: Digest of Papers, Tenth IEEE Symp. on Mass Storage Systems, May 1990, pp 114–121
21. Tweten D (1990) Hiding mass storage under UNIX: NASA's MSS-II architecture. In: Digest of Papers, Tenth IEEE Symp. on Mass Storage Systems, May 1990, pp 140–145



HweeHwa Pang received his BS - with first class honors – and MS degrees from the National University of Singapore in 1989 and 1991, respectively, and the PhD degree from the University of Wisconsin at Madison in 1994, all in Computer Science. He is now on the research staff of the Institute of System Science. Dr. Pang's research interests include database management systems, multimedia servers, and real-time systems. He is now heading the METEOR project that has the goal of building an operational distributed multimedia storage server.