Vertical Data Migration in Large Near-Line Document Archives Based on Markov-Chain Predictions*

Achim Kraiss Gerhard Weikum

Department of Computer Science, University of the Saarland
P.O. Box 151150, D-66041 Saarbrücken, Germany
E-Mail: {kraiss,weikum}@cs.uni-sb.de http://www-dbs.cs.uni-sb.de

Abstract

Large multimedia document archives hold most of their data in near-line tertiary storage libraries for cost reasons. This paper develops an integrated approach to the vertical data migration between the tertiary and secondary storage in that it reconciles speculative preloading, to mask the high latency of the tertiary storage, with the replacement policy of the secondary storage. In addition, it considers the interaction of these policies with the tertiary storage scheduling and controls preloading aggressiveness by taking contention for tertiary storage drives into account. The integrated migration policy is based on a continuous-time Markov-chain (CTMC) model for predicting the expected number of accesses to a document within a specified time horizon. The parameters of the CTMC model, the probabilities of co-accessing certain documents and the interaction times between successive accesses, are dynamically estimated and adjusted to evolving workload patterns by keeping online statistics. The integrated policy for vertical data migration has been implemented in a prototype system. Detailed simulation studies with Web-server-like synthetic workloads indicate significant gains in terms of client response time. The studies also show that the overhead of the statistical bookkeeping and the computations for the access predictions is affordable.

1 Introduction

1.1 Problem Statement

Internet/WWW and Web-like intranet infrastructures gain increasing importance as a medium for convenient information access within large enterprises and across the world. While the narrowly restricted bandwidth of the Internet currently limits the amount and type of data that is offered on the Web (e.g., in electronic product catalogs), a tremendous growth of multimedia data (images, videos, animations, etc.) is expected in the near future with rapidly increasing network bandwidth. We may soon see Web servers (probably with a full-fledged DBMS behind them) that have to manage Terabytes or even Petabytes of data and provide efficient access to millions of clients. In the following we will refer to the data objects of such a server generically as *documents*.

Among the multitude of documents that are held by a server, typically only a small fraction is "hot", that is, frequently accessed. Furthermore, the hot fraction will evolve over time; previously hot documents become "cold" (i.e., requested infrequently) but still need to be archived for occasional access. For cost/performance reasons (cf. [GP87]), cold documents, which may be accessed only once every so many hours, should reside in tertiary storage libraries. Such libraries provide "near-line" access by keeping data on magneto-optical platters or tapes, generically referred to as *volumes*, that reside in a robot-served jukebox with a certain number of drives, typically at least one order of magnitude fewer drives than volumes. So, in principle, all documents are available online, but the high latency of possible volume exchanges in the drives may incur response times of more than 10 seconds or even minutes. Thus, it is crucial that the currently hot documents are indeed held on the secondary storage level (i.e., the disks) of the storage hierarchy.

In the presence of evolving document popularities and access patterns, the disks then essentially serve as a cache with regard to the tertiary storage, and good cache *replacement* policies for variable-length documents are extremely important for the overall server performance. In addition, the vertical data migration between the tertiary and the secondary storage and thus the cache hit rate can be further improved by employing "intelligent" *preloading* policies, so that the high latency of the tertiary storage can be masked from the client in many cases.

Designing good replacement and, especially, preloading policies for the vertical data migration between secondary and tertiary storage is substantially more difficult than standard DBMS buffer management for a number of reasons:

1. The units of data migration, the documents, have a very high variance in their size. DBMS buffer management is well understood for page granularity, but practical work on variable-size granule caching policies has been limited to outdated operating system architectures with non-paged memory and would at least have to be re-assessed for the new application setting.
2. Prefetching into DBMS page buffers works excellent for sequential scans [TG84]. Preloading from tertiary storage, on the other hand, can be beneficial for more complex access patterns, and thus requires a much richer access-prediction model.
3. Overly aggressive prefetching may have a detrimental effect on the cache replacement in that it possibly reduces the effectively exploited cache size by prefetching data that may turn out not being accessed at all or only in the far future.
4. Both cache replacement and preloading interfere with the scheduling policy of the tertiary storage library, particularly, the policy for exchanging volumes. Throughput considerations suggest minimizing volume exchanges and using SCAN-like service pol-

Proceedings of the 23rd VLDB Conference
Athens, Greece, 1997

* This work has been supported by the ESPRIT Long Term Research Project No. 9141, HERMES (Foundations of High Performance Multimedia Information Management)

icies on an individual volume, but this may adversely affect response time. Both aspects have to be taken into account by the vertical data migration policies.

5. Under high load, drive contention leads to queueing of document requests at the tertiary storage. Speculative preloading would aggravate the queueing delays of pending document requests. Therefore, a good preloading policy needs to take into consideration the utilization of the tertiary storage drives.

1.2 Contribution and Outline

This paper develops a unified approach to cache replacement and speculative preloading, based on a stochastic model for predicting document accesses, and integrates this vertical migration policy with the scheduling policy of the tertiary storage library. In doing so, we aim to minimize the response time of client requests. We are not aware of any similarly comprehensive work on managing large near-line document archives.

Technically, the major novelty of the paper lies in using a continuous-time Markov-chain model and its underlying theory [Nel95, Tij94] for predicting future document accesses. This model involves estimating, through access monitoring, the transition probabilities between documents that are successively requested within a client session, that is, the probability that a client requests document j given that its previous request accessed document i. We further monitor the interaction times between successive session requests, and also the arrival rate of new client sessions. From these parameters, we utilize mathematical results on Markov chains to predict the expected number of accesses to certain documents within a specified time horizon.

Obviously, a Markov-chain model fits well with navigational accesses, where a client would start a new session by accessing some "entry document" and then proceed along a hyperlink structure. Navigational access seems to be typical for applications like teleteaching, virtual museums, and the like. However, the Markov-chain model does in no way rely on this type of access mode. What it captures are the patterns of co-accesses: access to a certain document affects the probability of accessing a certain other document in the near future. Thus, the Markov-chain model is applicable equally well to a descriptive access model with high-level queries; for example, the transition probabilities between documents would reflect if two documents contain semantically related information and consequently both appear in the result set of many queries. Furthermore, client caching of documents is automatically factored out, as requests served by the client cache are not known to the server's bookkeeping and are thus not considered in the parameter estimations, which is perfectly adequate.

The Markov-chain model pursued here is substantially richer (in terms of capturing more workload information) than a class of models that merely aim to estimate the stationary access probabilities of the various documents, often referred to as the "heat" of a document [Co88]. Taking into consideration the current state of an active client session, i.e., the last requested document, leads to much better predictions than the simpler stationary-probability models. On the other hand, it is evident that the parameter estimation of a Markov-chain model incurs much more bookkeeping overhead. We believe that this is one of the reasons why Markov-chain models have not received more attention for cache management between memory and secondary storage. With the high latency of tertiary storage, it is worthwhile to employ a richer decision-making model even if its overhead may not be negligible.

Whereas discrete-time Markov chains have been used in the literature for characterizing the access patterns of a single client [TN91, CKV93, Be96], our approach proceeds substantially further in that we

1. incorporate document-specific client interaction times between successive document requests by using a continuous-time rather than a discrete-time Markov chain,
2. reconcile the Markov-chain-induced access patterns of all simultaneously active client sessions into a global prediction, and
3. take into account, within the mathematical framework, the dynamic "out-of-the-blue" arrivals of new client sessions, whose initial state is unknown so that accesses cannot be predicted based on the last requested document, and also the termination ("departure") of sessions.

Incorporation of time into the model is crucial in order to capture the very high variance of client interaction times among documents. A user typically spends much less time on overview-like HTML documents that merely contain graphically enriched anchors than on long text and image documents with complex and interesting contents. Furthermore some browsing tools support the automatic following of embedded links which leads to very short interaction times.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 develops a continuous-time Markov-chain model for predicting near-future document accesses. Section 4 presents the integrated vertical migration policy that incorporates preloading, replacement, and the scheduling of volume exchanges. In Section 5 we discuss the bookkeeping overhead of our policy in terms of CPU and memory consumption. Section 6 gives an overview of our prototype implementation. Section 7 presents experimental performance results based on simulation and Section 8 concludes the paper.

2 Related Work

Tertiary storage management for long-term file archival has been an important issue for supercomputing centres; policies for the replacement of files on the secondary storage have been limited to simple heuristics, however, based on file age or estimates of the stationary access probabilities [Smi81]. More recent work has focused on data placement on tertiary storage volumes [FC91, CR94, TCG96] and request scheduling [HS96, NKT97]; this includes work with special considerations on the real-time requirements of video data [GMW94, LLW95]. Motivated by the large data volume in data warehouses, tertiary storage management has also received attention in the context of relational DBMS queries [Sto91, ML97, Sa95].

Prefetching in database systems has been studied mostly for applications where future access patterns are largely predictable due to specific structures of the underlying databases and the programs accessing them, especially in object-oriented database systems [CK89, CH91, GK94], but also in real-time and multimedia applications [WZ86, MKK95]. The effect of object preloading is implicitly achieved (on a per page basis) also by intelligently clustering objects into pages [CK89, TN91, TN92, GKKM93].

There is only little work on prefetching based on probabilistic models. Fundamental properties of Markov-chain-based paging have been investigated in [KPR92] with the focus on the asymptotic worst-case competitiveness of online algorithms. On the practical side, [PZ91] has proposed an associative memory approach for predicting object requests and initiating prefetching. A major disadvantage of this approach is that the associative memory needs offline training, which renders it infeasible for document archives with evolving workload patterns. [MKK95] has proposed a relevance ranking scheme for the buffering of video frames in multimedia applications that aims to capture access probabilities, but relies on external input for determining the relevance measures. In [CKV93] compression schemes based on k^{th}-order Markov chains have been

applied to the problem of prefetching pages, and [Be96] uses a first-order Markov chain for speculative prefetching in a distributed system. All these strategies are tailored to supporting a single access sequence running with dedicated client memory, which is not applicable in our scenario where multiple, dynamically arriving and departing sessions compete for cache space (i.e., the secondary storage). Also, object-specific interaction times have been disregarded, and prefetching has been studied in isolation in the above mentioned work, without considering the interdependencies with cache replacement and storage device scheduling.

The only approaches that aim to reconcile the replacement and prefetching policies for an in-memory page cache are [GK94], [CFKL95a, CFKL95b], and [PGG+95], but all of these assume perfect knowledge of future page accesses through application hints. [GK94] explicitly maintains, in a special data structure, the page access history of method invocations within object-oriented databases, and uses this information for prefetching pages into the method's working space in memory and also for selecting replacement victims based on the remaining number of accesses within the method execution. [CFKL95a, CFKL95b] develops rules for when aggressive prefetching needs to be throttled in order to avoid adverse effects on the page replacement (e.g., prefetching a page that causes the replacement of a previously cached page that will be re-used earlier than the prefetched page) and analyzes bounds on the suboptimality of two heuristics. Finally, [PGG+95] develops a simple cost model with constant CPU and disk access time per page request to heuristically control the dynamic subdivision of cache space into an LRU-managed cache and a separate prefetching cache. All of these approaches are geared for cases in which the application's accesses are perfectly predictable (e.g., a Unix grep command running on a directory tree of files), and cannot be used in our problem setting.

3 Stochastic Model

In this section we describe the stochastic model for the prediction of future document accesses. We assume that clients open sessions with the server and then proceed through a number of document accesses before terminating a session, which models an interactive multimedia information system. Let D denote the document set stored on a server consisting of N documents $d_i \in D$, i=1..N. Furthermore let S denote the set of currently active user sessions $s_j \in S$, j=1..|S|, and let $d(s_j)$ denote the last (i.e., most recent) document that the session s_j has requested from the server. We model the request patterns of a single session as a continuous-time Markov chain [Nel95, Tij94], as developed in Section 3.1. We will then show in Section 3.2 how multiple sessions and, particularly, the dynamic arrivals of new sessions can be incorporated into the model.

3.1 A Continuous-Time Markov-Chain Model for a Single Session

A continuous-time Markov chain (CTMC) is a stochastic process that proceeds through different states in certain time epochs. Its basic property is that the probability of entering a state depends only on the current state, not on the previous history (this is a first-order Markov chain; higher-order Markov chains are not relevant to this paper). This property has the mathematical implication that the time for which the process resides in a given state must be an exponentially distributed random variable; different states may have different mean residence times, however. Thus, a CTMC with states denoted 1, 2, ... N is uniquely described by a matrix $P=(p_{ij})$ of transition probabilities between states, and the mean residence times (or "state holding times") H_i of the states. Equivalently, one can specify the transition rates v_{ij} between states i and j, where $v_{ij} = \frac{1}{H_i} * p_{ij}$; the term $1/H_i$ is also known as the state departure rate and denoted as v_i.

In our application setting, the state of the CTMC corresponds to a session (i.e., the stochastic process) accessing a certain document. For each document d_i, p_{ij} denotes the probability that when a session has requested document d_i, it will next request document d_j from the server. The state residence time corresponds to the time that the session resides at a document; this captures the actual interaction time, i.e., the time that a human user needs to "digest" a document's contents or a browser needs to process the document before requesting the next one.

We are interested in predicting the future accesses of a session. In this prediction, we can exploit the knowledge of a session's current state. Thus, the first relevant measure that we are interested in are the probabilities $p_{ij}(t)$ that a session will be in state j (i.e., will access document d_j) at time t from now, given that it currently resides in state i (i.e., document d_i). There are well-known methods for performing this type of transient analysis of a CTMC. However, a first difficulty in applying these methods is the fact that the mean residence times are not uniform across all states. To overcome this problem, we apply a method that is known as *uniformization* [Tij94] to transform the CTMC into an equivalent CTMC with uniform mean residence times. Here equivalence means that both processes will be in the same state with the same probability for all times t; so we have $p_{ij}(t) = \bar{p}_{ij}(t)$ where $p_{ij}(t)$ refers to the original CTMC and $\bar{p}_{ij}(t)$ to the uniformized CTMC. The uniformization method essentially adjusts the state transition probabilities so as to factor out the different mean residence times; this involves introducing transitions back into the left state and is described mathematically as follows:

$$\bar{p}_{ij} = \begin{cases} \frac{v_i}{v} * p_{ij}, & j \neq i \\ 1 - \frac{v_i}{v}, & j = i \end{cases} \quad \text{where } v = \max\{v_i \mid i = 1..N\} \qquad (3.1)$$

The formal proof for this uniformization can be found in [Tij94]. The central property that is exploited here is that the state-transition epochs of the uniformized CTMC can be generated by a Poisson process with rate v, the maximum state departure rate of the original CTMC.

Next we consider the m-step transition probabilities $\bar{p}_{ij}^{(m)}$ of the uniformized CTMC, i.e., the probabilities that the session will be in state j after m transitions given that it currently is in state i. These can be inductively computed from the Chapman-Kolmogorov equations [Nel95, Tij94]:

$$\bar{p}_{ij}^{(m)} = \sum_{k=1}^{N} \bar{p}_{ik}^{(m-1)} \bar{p}_{kj} \quad \text{with } \bar{p}_{ij}^{(0)} = \begin{cases} 1 \text{ if } i=j \\ 0 \text{ otherwise} \end{cases} \qquad (3.2)$$

Finally we obtain the time-dependent transition probabilities $p_{ij}(t)$ by taking the product of the probability that m steps are performed in time t with the m-step transition probability, and summing up these products for all possible values of m. This is exactly the part of the derivation that is greatly simplified by the previous uniformization, and we obtain:

$$p_{ij}(t) = \sum_{m=0}^{\infty} e^{-vt} \frac{(vt)^m}{m!} * \bar{p}_{ij}^{(m)} \quad \text{for all i,j and } t > 0 \qquad (3.3)$$

We will show in Section 5 that these probabilities can be computed efficiently in an incremental manner, i.e., without actually having to approach the infinite sum. The $p_{ij}(t)$ values denote the probability

that a session resides on document d_j at time t (from now on) under the condition that the session currently resides on document d_i. For the decision on whether it is beneficial to preload a certain document from tertiary storage onto disk and possibly drop another document from the secondary storage as a replacement victim, we are interested in the expected number of requests to a document within a certain lookahead time horizon t. We postpone the discussion on how to set and possibly fine-tune the value of the lookahead time until Section 4.3. Note that we are still focusing on a single session only, but estimating the expectation of the number of requests to a document will later allow us to reconcile multiple, concurrently active sessions by essentially summing up these expectation values.

The expected amount of time that a session that currently resides in state i will spend in state j within a time horizon of duration t is obtained by the product of the mean residence time per visit of state j, which is 1/v, and the expected number of visits to j or, actually, departures from state j within time t. We consider departures from j rather than arrivals at j so that we count only complete visits within the time horizon t (i.e., complete residence times), where the difference matters in the transient analysis, as opposed to steady-state analyses, for the time horizon t may be relatively short. The expected number of departures from j is in turn obtained by summing up, for all possible values *n* of the total number of transitions within time t, the product of the probability that n transitions are performed within time t and the probability that state j is reached from state i in less than n steps. So we arrive at the following formula [Tij94]:

$$E_{ij}(t) = \frac{1}{v} * \sum_{n=1}^{\infty}\left(e^{-vt}\frac{(vt)^n}{n!} * \sum_{m=0}^{n-1} \overline{p}_{ij}^{(m)} \right) \qquad (3.4)$$

Finally, to derive the expected number of arrivals at state k, or, equivalently, accesses to document d_k, we consider all possible predecessor states j that have transitions into k (with non-zero probability). $E_{ij}(t) / (1/v)$, the ratio of the total time spent in j (during complete visits) to the mean time per visit, is the expected number of complete visits to and thus departures from j, and we finally obtain the expected number of transitions into k by multiplying the expected number of departures from the predecessor state j with the transition probability p_{jk} and summing up these values over all predecessor states j. This yields the following formula:

$$E[\text{number of accesses to } d_k \text{ in time } t] = \sum_{j=1}^{N} v * E_{ij}(t) * \overline{p}_{jk} \qquad (3.5)$$

So we finally have a mathematically founded predictor for the near-future number of accesses to a document and, thus, a basis for assessing the "worthiness" of a document, i.e., the benefit of preloading the document from tertiary storage and/or keeping it in the secondary-storage cache.

3.2 Incorporating Multiple Sessions with Dynamic Arrival and Termination

The prediction formula derived in the preceding subsection holds only for a single session for which we know its current state (i.e., its last requested document). For the overall optimization of the server, we still need to reconcile the predictors of multiple ongoing sessions, and we also have to take into consideration that new sessions arrive dynamically and we do not know in advance their initial state (i.e., the first requested document of a session). The first problem can be easily solved by summing up, over all ongoing sessions, the expected values of the number of accesses to a document within a session:

$$E[\text{total number of accesses to } d_k \text{ in time } t] = \sum_{s \in S}\sum_{j=1}^{N} v * E_{d(s),j}(t) * \overline{p}_{jk} \qquad (3.6)$$

where d(s) is the document on which session s currently resides (i.e., the current session state).

Now consider the issue of newly arriving sessions. Disregarding these and focusing only on the ongoing sessions would underestimate the number of near-future accesses to certain documents, in particular, those documents that are the first ones to be accessed by new sessions. Accesses to these "entry" documents arrive "out of the blue" so-to-speak. There is an elegant way of incorporating these accesses into the CTMC framework. We simply add to the CTMC model additional, fictitious states N+1, ..., N+c that represent all currently inactive clients (which do not have a session in progress) among which we expect c new sessions to be started within the time horizon t. The value of c is derived from the session arrival rate λ, which can easily be monitored, by setting $c = t*\lambda$, and the mean residence time of state N+i is set to $i*(1/\lambda)$, the expected time until the i^{th} session starts. The transition probabilities $p_{N+i,j}$ (i=1..c) are the stationary access probabilities for the entry documents of new sessions. Once the CTMC is extended in this way, we can directly apply the derivation of Section 3.1 with states N+1 through N+c added to the various formulas, and the only thing to do in addition is to add c fictitious sessions, one session residing on each of the states N+1, ..., N+c, to the set S of sessions over which the per-session expectations are summed up (see formula 3.6).

In contrast to session arrivals, there is no explicit notion of a session termination. We simply consider a session as terminated if it does not issue any further requests for a certain timeout period. In terms of modeling the impact of terminations, however, the termination of ongoing sessions can be taken into account, similarly to the above considerations on arrivals, by adding transitions between each state i and an additional, fictitious (absorbing) state 0, where the transition probability $p_{i,0}$ denotes the probability that a session terminates (i.e., remains inactive for the timeout period) after having accessed document d_i. These probabilities can be estimated through continuous monitoring in the same way as all other transition probabilities.

4 Integrated Migration Policy

This section presents our vertical migration policy that aims to reconcile the preloading from tertiary storage, the replacement on secondary storage, and the scheduling of volume exchanges. The algorithms are based on the expected number of near-future accesses to a document as derived by equation (3.6). We will first present the preloading and replacement algorithms in Subsection 4.1, and then elaborate on the tertiary storage scheduling in Subsection 4.2. Subsection 4.3 discusses the fine-tuning of the lookahead time horizon used by the CTMC predictions.

4.1 Preloading and Replacement of Documents

As the worst-case access time to near-line tertiary storage is orders of magnitude higher than a disk access (i.e., access to secondary storage), the general objective of our migration policy is to maximize the number of document requests that can be served from disk. The number of requests within a lookahead time t is constituted by two classes of requests. First, a document may have been explicitly requested so that sessions are waiting for it to be delivered; we will refer to such requests as *pending requests*. Second, documents have a

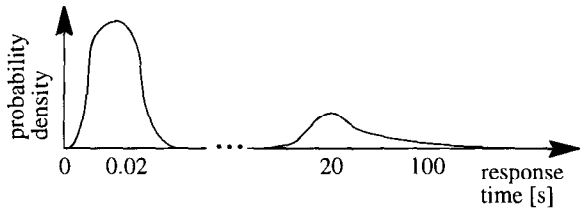

Figure 1:
Typical response time distribution of document accesses

probabilistically expected number of requests as given by the formula (3.6); we will refer to such requests as *speculative requests*. The latter class are the candidates for preloading, whereas the pending requests definitely need to be served. We denote the current number of pending and speculative requests for document d as $N_{pend}(d)$ and $N_{spec}(d,t)$ where t is the lookahead time horizon. Obviously, the ranking of documents in terms of migrating them to disk or dropping them from disk should be based on the sum of $N_{pend}(d)$ and $N_{spec}(d,t)$. An important additional consideration, however, is that the size of a document and thus its cache space occupation affects its worthiness of being cached. Normalizing the caching worthiness on a per byte basis leads to the following definition of *document weights* (see also [Co88, SSV96] for similar considerations):

$$weight(d) := (N_{pend}(d)+N_{spec}(d,t)) / size(d) \qquad (4.1)$$

Clearly, the disk hit rate within the next t time units is maximized by trying to bring onto disk and keeping those documents that have the highest weights. (We assume that the disk space is so large compared to the document size that space fragmentation is not an issue.) To approach this behavior by an online decision-making algorithm, our migration algorithm considers preloading requests for those documents whose weight exceeds that of currently disk-resident documents. The algorithm actually proceeds in incremental steps by picking in a single step one preloading candidate, based on descending weight order, and one or more replacement candidates (depending on the required space), based on ascending weight order. The disk space for the preloading candidates is pre-reserved but not yet freed up until the tertiary storage is actually ready to transfer the document. At that time, the weights of the preloading candidates and the replacement victims are re-assessed (as they may have changed in the meantime), and the preloading request may be cancelled if the document's weight no longer indicates the worthiness of being cached.

From the above considerations we would expect to obtain a response time distribution for the client requests with a shape as shown in Figure 1. This is a bimodal probability density function (i.e., with two local maxima) where the first mode corresponds to requests served from disk and the second mode to the much slower tertiary storage accesses. If preloading and caching work effectively, then most of the probability mass (i.e., area under the curve) should be around the first mode; so the probability of still having to fetch a document from tertiary storage after it has been explicitly requested by a client should be low. The distribution of that remaining probability mass around the second mode is an important factor, too. Namely, we want to ensure that the mode itself (say the mean access time of the tertiary storage if these times were normally distributed) occurs at a minimum horizontal position, and that the tail of the probability distribution is as "light" (pictorially, quickly approaching zero) as possible so as to bound the tail some percentile.

Unfortunately, this desired result is not self-guaranteed. Namely, if preloading requests are initiated too aggressively, they may lead to high contention for tertiary storage drives. The net effect could be that speculative requests are served at the expense of delaying pending requests. Therefore, in spite of a high disk hit rate, a performance problem may arise if the response time of disk misses, i.e., explicit client requests that end up having to be served from tertiary storage, deteriorates. To avoid such adverse effects, our algorithm performs a benefit-penalty comparison for each preloading candidate and initiates the speculative request only if its benefit exceeds the penalty that it incurs on pending requests. Let T_{mig} denote the average migration time from tertiary storage, i.e., the time between an explicit client request to a document and the arrival of the document on disk from where it can be delivered to the client; this time may include volume exchanges. The benefit of a preloading request within the time horizon t can be defined as its expected number of accesses $N_{spec}(d,t)$ times the response time T_{mig} that it would have if it is not preloaded:

$$benefit(d) := N_{spec}(d,t) * T_{mig} \qquad (4.2)$$

On the other hand, pending requests are potentially delayed by an additional speculative preloading request; this is the penalty of initiating the preloading. Obviously, we would not want to cause a volume exchange merely for speculative requests; rather we should start serving a speculative request only when the required volume is online anyway. This time point depends on the tertiary storage scheduling policy, but fortunately we need to assess the penalty of the preloading request only when we are about to serve it, which is exactly when its volume becomes online, so that our considerations do not depend on the scheduling policy. Let N_{pend} be the total number of pending requests to off-line volumes at the time when we consider serving a speculative request. Each of these requests is delayed by the extra time it takes to tranfer the speculative document from tertiary storage (into memory and then onto disk). Let R_{trans} denote the effective transfer rate of the tertiary storage, taking into account head positioning delays on a volume but no volume exchanges. Then the penalty of speculatively preloading document d is given by:

$$penalty(d) := N_{pend} * (size(d) / R_{trans}) \qquad (4.3)$$

Finally, tertiary storage drive contention is taken into account by initiating only such speculative requests, based on the weight ranking, whose benefit exceeds the penalty. Requests that have been initiated but, at the time when they are about to be served, turn out to have a penalty that is higher than the benefit, are cancelled.

4.2 Scheduling of Volume Exchanges

Because of the very high delay incurred by volume exchanges, any reasonable tertiary storage scheduling policy must attempt to batch requests for the same volume so as to limit or even minimize the unproductive time wasted by volume exchanges. This holds for both pending and speculative requests, but pending requests are critical in terms of response time so that they should not be postponed too much for the benefit of batching. The scheduling policy that we advocate therefore keeps two queues q_{pend} and q_{spec} of volume IDs for which explicit, pending requests and speculative preloading requests have been issued. As long as the q_{pend} queue is not empty, those volumes are loaded into drives which have pending requests. Naturally, preference should be given to volumes v with a high number of pending requests, $N_{pend}(v)$. On the other hand, setting volume priorities only on this basis would cause the danger of request starvation. Thus, to prevent starvation, volumes are actually loaded into drives in descending order of the product $N_{pend}(v)*T_{wait}(v)$ where $T_{wait}(v)$ is the longest waiting time among all pending requests for volume v. Once a volume is loaded into a drive, all pending and speculative requests are combined and reordered for being served by a SCAN-like pass on the volume. It is at this point when the benefit and penalty of serving a speculative request are re-assessed and speculative requests may be cancelled (see Section 4.1 above). Requests

that arrive during the pass and whose position on the volume have already been passed are held back in their queue until the next time when the volume is loaded (which may be right after finishing the current pass if there is no contention for drives).

Whenever a drive is unused with regard to the explicitly issued pending requests, the scheduling policy loads a volume solely for serving speculative requests. The volume selection policy that we advocate here is to give preference to volumes with a high number of expected accesses. For each volume we simply sum up the expected number of near-future accesses (formula (3.6)) for all preloading candidate documents that reside on the volume, and maintain a queue of volumes in descending order of this total number of expected accesses. We refer to this algorithm as the MEAT (most expected accesses top-priority) policy. So the results from our stochastic model (see Section 3) are not only useful for initiating preloading requests, they also serve as a heuristics for scheduling volume exchanges.

4.3 Fine-Tuning of the Lookahead Time Horizon

So far it may appear that the migration policy crucially depends on a proper setting of the lookahead time horizon t that plays a prominent role in predicting the benefit of a speculative request. It is indeed true that a careless choice of this fine-tuning parameter can cause adverse performance effects: setting it too small means that even highly likely but speculative requests are recognized too late, and setting it too high would overestimate the benefit of speculative requests and may cause high contention for tertiary storage drives. In particular, the second case may lead to situations where a volume is loaded into a drive solely on behalf of speculative requests and an explicit pending request that is issued a second later is delayed for a long time as the speculative request queue for the volume in the drive may be very long. Furthermore, if secondary storage space is scarce and the preloading is too aggressive, the preloaded documents may cause the replacement of documents that turn out to be (re-) used earlier than the preloaded ones.

Fortunately, there is a simple rationale for setting the lookahead time horizon. Consider the average time period between two successive points where the same volume is loaded into a drive; this metric can be measured online and will be denoted as T_{load}. A speculative request can be served "conveniently" (i.e., at low marginal cost) when the corresponding volume resides in a drive, and such a speculation will eventually turn out to be a good decision if the preloaded document will be explicitly requested within the time T_{load} (i.e., until the same volume is accessible again). Otherwise, if the speculatively requested document turns out to be not used at all within time T_{load}, then loading it now would be a waste of cache space and it should better be loaded upon the next opportunity when the required volume is accessible, which will be T_{load} time units later. This consideration shows us that T_{load} is an upper bound for the lookahead time of the access predictor. Conversely, if a document that is expected to be accessed within time T_{load} is not preloaded, it would eventually become a pending request and penalize the client response time as that pending request would likely have to wait for the volume to be loaded again into a drive. So T_{load} is indeed a reasonable canonical choice for the lookahead time.

5 Implementation of the Bookkeeping

In this section we discuss the implementation and the overhead of the bookkeeping for the access predictions, the preloading and replacement of documents, and the tertiary storage scheduling. We consider both consumption of memory space and CPU time.

We keep moving-average statistics for all state transition probabilities of the CTMC (i.e., a dynamically aging counter of how frequent a transition from d_i to d_j occurs) and all state residence times (i.e., the total residence time of the last z visits to d_i where z is a fine-tuning parameter which we set to 50). In principle, this incurs space overhead that grows quadratically with the number of documents. However, the transition probability matrix P of the CTMC is typically very sparse provided that the workload exhibits spatial locality, which has been observed for Web server traces [ABCO96] and is a reasonable assumption for other document-archive applications, too. All the statistical information is kept in hash tables indexed on state i.

Each time a session changes its state by requesting the next document d_j, the expected numbers of near-future accesses to other documents change according to equation (3.5). At this point, the state-transition graph of the Markov chain is traversed, starting from d_j and always proceeding along the highest-probability transition of the state with the currently highest probability of being reached from d_j. During this traversal, upon each visit of a state j its $E_{ij}(t)$ value is incremented by the product of the mean residence time and the accumulated probability of reaching j within time t (see formula (3.4)). The procedure is terminated when the probability of reaching a state within the lookahead time t drops below a specified threshold δ (which we have set to 0.01). Note that this stopping threshold allows us to bound the computational overhead of the traversal. Further note that the entire computation is incremental in that it reuses intermediate results to the most possible extent. Finally note that the computed $E_{ij}(t)$ values are actually session-independent; thus they are kept and reused for all subsequent access predictions of other sessions, as long as the basic statistical parameters, the transition probabilities and the state residence times, do not change much. In real applications, we would expect exactly such quasi-stable base parameters with major shifts occurring only occasionally on a long-term scale.

The algorithms for generating preloading requests and deciding on cache replacements are driven by two sorted lists, l_{sec} and l_{ter}, which keep the document IDs and the weights of all documents that reside on secondary storage and all relevant documents that reside on tertiary storage, respectively. Both lists are sorted by weight. The size of l_{sec} obviously depends on the number of disk-resident documents and should incur only negligible space overhead. The size of l_{ter}, on the other hand, is bounded by considering only documents for which at least one session is predicted to access the document within time t (with probability above the aforementioned threshold δ). In addition, the requirement that benefit(d) > penalty(d) should hold for a document d to be preloaded can be exploited to further prune the list l_{ter}, as this inequation yields a lower bound on $N_{spec}(d,t)$ and thus weight(d) for document d to be relevant at all. Note that this consideration leads to the nicely adaptive effect that the overhead of the migration algorithm's data structures is automatically reduced when the load in terms of explicit, pending requests becomes very high.

The overhead of the tertiary storage scheduling algorithm, MEAT, is negligible. We merely have to track the values of N_{pend} and T_{wait} on a per volume basis.

6 Prototype Architecture

In this section we describe the architecture and some implementation details of our document archive prototype. Figure 2 shows the general architecture of the system, with data structures depicted by ovals and dynamically adjusted control parameters depicted by hexagons. At the top level, the system consists of the Session Manager that maintains information about the state of active sessions, the

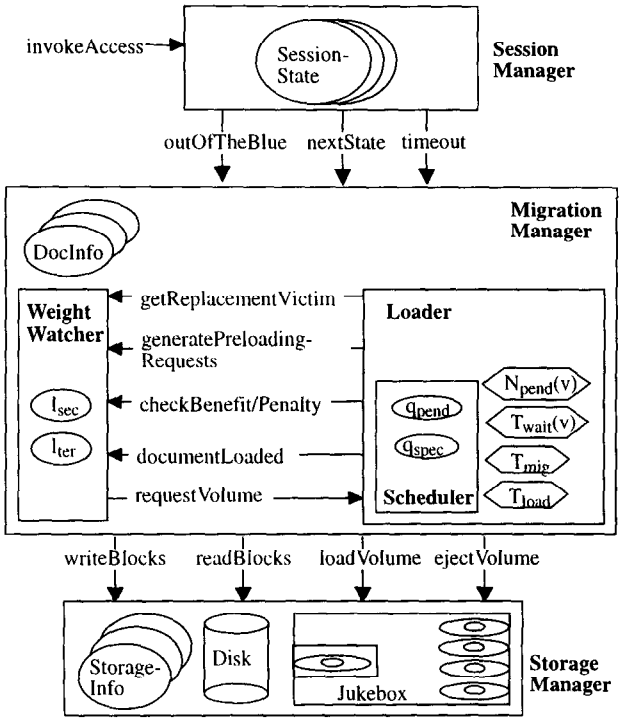

Figure 2: Architectural overview of the prototype system

Migration Manager for predicting and scheduling document accesses, and the Storage Manager.

The Session Manager tracks the arrivals of new sessions and the current state of the active sessions, and also decides that a session is terminated by means of a timeout and then discards all session-specific bookkeeping information. The Migration Manager organizes the migration of documents between tertiary storage and secondary storage and the transfer to the requesting client. The most important submodule of the Migration Manager is the Weight Watcher which implements the access predictions and manages the sorted lists l_{ter} and l_{sec}. Each new request of a session as well as session arrivals and departures are signalled to the Weight Watcher which then updates the document weights as described in Section 5. Each time a volume becomes loaded into a drive, the Weight Watcher is requested to generate speculative requests.

Finally, the StorageManager provides a block-oriented interface to the secondary storage disks and the tertiary storage jukebox and maintains directory information such as address mapping tables. Free space is managed by a first-fit allocation algorithm for both disks and tertiary storage volumes. The Storage Manager can interact with real devices, or it can use simulated devices from a library of detailed device models based on the CSIM simulation package [DevSim,CSIM]. The simulated devices include controller caching, realistic seek times, rotational latencies, head switch times, and so on [RW94]. In this paper we have considered only simulated devices for better reproducibility and statistical confidence of the performance results.

7 Experimental Evaluation

In this section we present simulation results on the performance and overhead of our vertical migration policy. We restrict ourselves to studies with synthetic workloads whose key parameters have been derived from current Web applications; in contrast to trace-based studies, this gives us higher statistical confidence and the ability to systematically vary certain parameters so as to obtain insights on workloads that are expected for future applications. We compare three different policies, all of which use the MEAT policy for the scheduling of volume exchanges:

- *MCMin* (MC-based migration policy for near-line storage): the integrated vertical migration policy that we have developed in this paper, based on the CTMC model.
- *TEMP*: a replacement-only algorithm with no preloading at all, which selects replacement victims based on the temperature of documents (i.e., heat / size). The heat of a document reflects its stationary access probability, and is dynamically estimated by tracking a moving average of the interarrival times of the last 20 requests to a document. This method can be viewed as a straightforward generalization of the LRU-K buffering algorithm [OOW93, WHMZ94] to the case of variable-size buffering granules.
- *TEMP-P*: this algorithm extends TEMP by additionally preloading those documents from tertiary storage whose estimated temperature exceeds the lowest temperature among the currently disk-resident documents. Like MCMin, the preloading is subject to a comparison of benefit vs. penalty (see Section 4.1) but with N_{spec} based on stationary access probabilities only.

7.1 Experimental Testbed

The experiments have been carried out on the prototype system described in Section 6 with simulated secondary and tertiary storage devices. The parameters of the devices used throughout all experiments are given in Table 1; these devices reflect today's commodity SCSI disks and low-end magneto-optical jukeboxes.

Table 1: System parameters of the simulation testbed

Secondary Storage	
Parameter	**Value**
Number of Disks	1
Average Seek Time	8 ms
Average Rotational Latency	4.3 ms
Transfer Rate	7.5 MBytes/s
Controller Cache Size	1 MByte

Tertiary Storage	
Parameter	**Value**
Number of Drives	1
Average Seek Time	25 ms
Average Rotational Latency	10 ms
Transfer Rate	3.4 MBytes/s
Controller Cache Size	1 MByte
Number of Volumes	8
Volume Exchange Time	10 sec

We have considered an archive with 20,000 documents; document sizes are exponentially distributed with mean 500 KBytes. Thus, the total archive size is approximately 10 GBytes. The documents are allocated randomly across the volumes of the tertiary storage library. We have analyzed the Web-server traces of two virtual museums to characterize the skewness of transition probabilities and the distribution of state residence times. We have incorporated these observations in a synthetic workload that we believe is realistic for large archives and also allows us to investigate a wide spectrum of different workloads. The synthetic workload is generated as follows:

- The documents are first arranged into a tree with constant fanout 4. This tree serves as a skeleton for generating the state transitions,

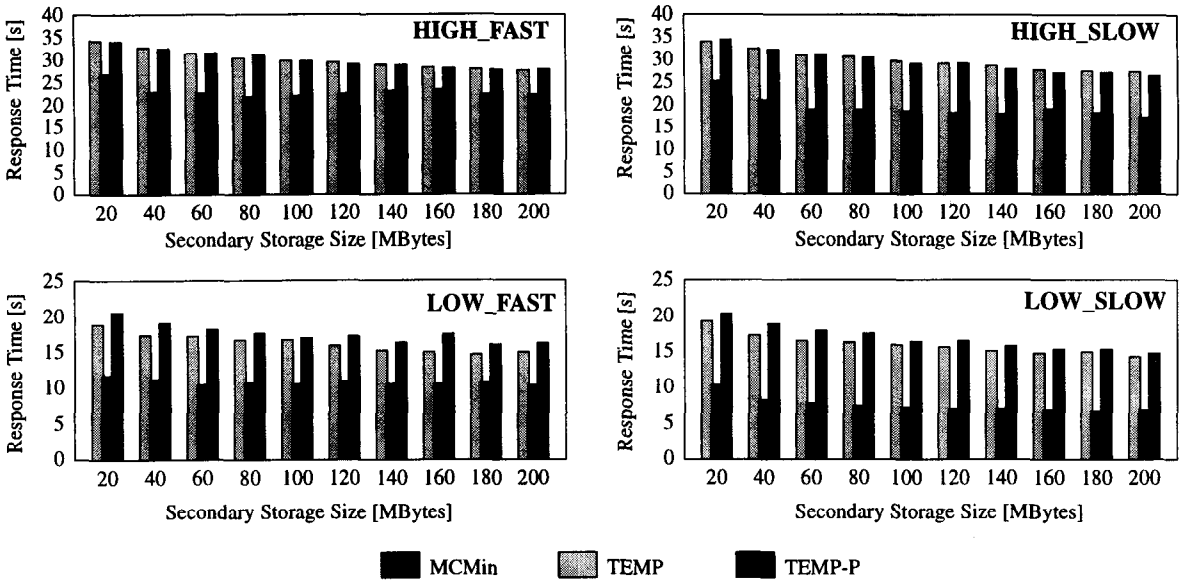

Figure 3: Response time results for the four workloads

which themselves are not tree-based and may even have cycles. For each document d_i (i.e., node in the tree) the transition probabilities to all other documents (including ancestors in the tree) are generated by a Gamma distribution with a coefficient of variation greater than 1 (namely, 1.5, to be specific) [All90]. This captures a skewed ranking of the transition probabilities from d_i to all other documents which starts with document d_{4i-2} and proceeds by document number modulo the total number of documents. This means that the highest-probability transition out of d_i leads to its leftmost child in the tree, and transitions to ancestors are less likely than to nodes further down in the tree. For leaf documents the highest-probability transition target is chosen randomly among all documents.

- The skewness of the transition probabilities out of a document d_i is determined by the expectation value of the Gamma distribution from which the transition probabilities are drawn. We assume that these values are exponentially distributed among the documents and generate this parameter of the document-specific Gamma distribution accordingly. The mean value for this exponential distribution is set to 6, where the value 6 would imply that 90 percent of the probability mass among a document's outgoing transitions is covered by the 16 most probable transitions of a document. The fictitious documents d_{N+1} through d_{N+c} (see Section 3.2) are treated separately as their successors represent the "entry" documents of new sessions; the mean value for the Gamma distribution associated with these states is set to 2, which implies that 90 percent of the probability mass among the outgoing transitions is covered by the 6 most probable successor states.

- The state residence times are exponentially distributed with the document-specific mean values drawn from a uniform distribution. New sessions arrive according to a Poisson process with rate λ (i.e., exponentially distributed interarrival times with mean $1/\lambda$).

- The duration of a session is specified in terms of the number of requests that a session is going to issue; this number is generated according to a normal distribution.

We have considered four workloads which differ in their session arrival rate and the distribution of mean residence times. The workload LOW_SLOW has low session arrival rate and high state residence times, whereas workload LOW_FAST has lower residence times. The workloads HIGH_SLOW and HIGH_FAST both have higher session arrival rate but differ in their residence times. The workload parameters are listed in Table 2.

Table 2: Workload parameters for the simulation experiments

Parameter	LOW_ SLOW	LOW_ FAST	HIGH_ SLOW	HIGH_ FAST
Mean session interarrival time	180 s	180 s	60 s	60 s
Mean session length	24 requests			
Standard deviation of session length	12 requests			
Min. mean residence time	30 s	10 s	30 s	10 s
Max. mean residence time	180 s	60 s	180 s	60 s

7.2 Experimental Results

The response time results for the three policies under comparison are shown in Figure 3 for each of the four different workloads. The charts show the mean response time of client requests as a function of the secondary storage cache space which is varied from 20 MBytes (0.2 percent of the archive size) to 200 MBytes (2 percent of the archive size).

Overall, it is evident that the newly developed MCMin policy consistently outperforms the other two policies. It improves the mean response time by up to a factor of 2. The highest gains are achieved for the LOW_SLOW workload, as this has the lowest utilization of the jukebox drives on behalf of explicit, pending requests and thus has the largest potential for preloading benefits. With 200 MBytes secondary storage space, MCMin achieves a disk hit rate of 68.1%, compared to 27.1% and 30.6% of the TEMP and TEMP-P methods, respectively. For all three methods, the disk arm utilization is less than 15 percent, whereas the jukebox drive is highly utilized, namely, by 99.6 percent with MCMin versus 77.3 and 99.8 percent with

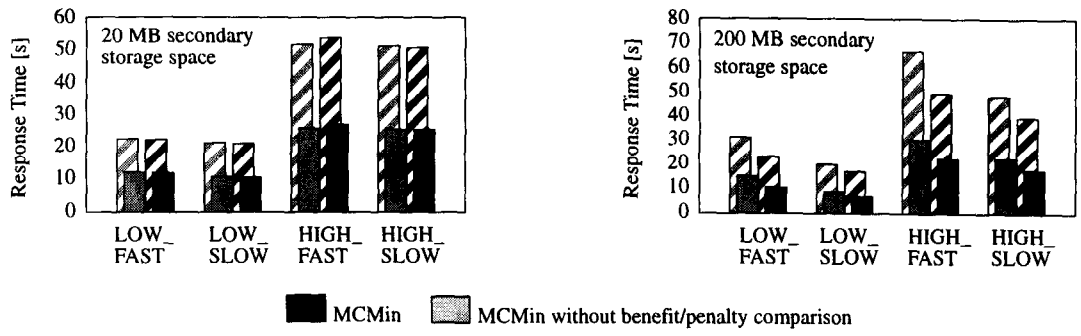

Figure 4: Impact of the benefit/penalty comparison

TEMP and TEMP-P, respectively. Note that the full utilization of the jukebox drives is exactly desirable if it contributes to shorter client response times by using idle time for preloading.

The gains for the other workloads are less impressive as the higher jukebox utilization prevents more preloading, and for the same reason the relative improvements become smaller with increasing cache size. Note that in these cases, where preloading is less attractive, the MCMin method automatically throttles the preloading activity based on its benefit-penalty computation (see Section 4.1); this is what makes it a robust, practically viable method. Figure 4 compares the MCMin response time (mean values depicted by filled bars and 80th percentiles by hatched bars) for small and large cache size to the performance that is achieved when the benefit-penalty comparison is switched off. The importance of this component of our integrated policy is evident.

It is interesting to note that the preloading-enriched TEMP-P policy has virtually no advantage over the TEMP method. The reason is its inherent limitation to stationary access probabilities. Under high jukebox utilization, particularly with the HIGH_FAST workload, TEMP-P would preload only documents that have very high stationary access probability, and these documents reside on secondary storage anyway. Under lower jukebox utilization, particularly with LOW_SLOW, on the other hand, it initiates many preloading requests, but these fail to fetch the relevant documents, namely, those that are actually fairly cold in terms of their stationary access probability but have a high near-future access probability because a session currently resides in the document's "proximity". This nicely illustrates the fundamental superiority of our Markov-chain-based approach over a stationary probability model. Only in the HIGH_SLOW scenario, TEMP-P is able to obtain a small gain over TEMP, for in this case, the arrivals of new sessions benefit from the preloading of documents in the "proximity" of the "entry" documents (which have a high stationary access probability).

Additional experiments with variations of the workload parameters essentially confirmed these findings and are omitted here for space limitation. We also investigated the bookkeeping overhead of the prototype implementation. The total space overhead of all bookkeeping data was 1.8 MBytes, and the CPU consumption for the predictions (i.e., the Markov chain computations) per session step was 100 milliseconds on average on a Sparc20 (i.e., a low-end server). This is clearly a small price for achieving such substantial gains in terms of client response time. The CPU overhead can be reduced even further by keeping a small number of recently computed predictions in a special lookup buffer; our experiments have shown that we can achieve an acceleration by approximately a factor of 10 with a lookup buffer of 1 MByte for the synthetic workloads that we considered.

8 Concluding Remarks

The vertical migration method for storage hierarchies presented in this paper is based on an integrated, quantitative assessment of the benefits and costs that arise in the cache replacement decisions, the initiation of speculative prefetching, and the scheduling of tertiary storage devices. The key to this reconciliation of the different aspects is the continuous-time Markov chain model that we have developed for predicting near-future accesses and its underlying mathematical theory. We believe that analytic models of this kind deserve much more attention for their ability to drive online decisions in the resource management of large-scale information systems. Note that the developed method is completely self-reliant in that it does not require any intervention by human administrators or tuning experts. All input parameters are automatically estimated by means of online statistics. Furthermore, although the method includes a number of control parameters that may be fine-tuned, we have provided simple, practically viable guidelines for choosing appropriate, robust values for these parameters.

The experimental studies of this paper have shown significant performance gains, up to a factor of two in the client response time, at the cost of an acceptable bookkeeping overhead. Thus, it is promising to generalize the approach to other, more comprehensive settings:

- Although the method can be carried over from jukeboxes with magneto-optical platters to tape library without any changes, we realize that the volume scheduling may pose more complex problems with tapes. Particularly, policies that emphasize batching and perform a complete volume scan when a volume is loaded into a drive may incur unacceptable delays with modern tape recording techniques like serpentine recording, where a complete tape scan would take several minutes [HS96]. Therefore, the scheduling issues need to be reconsidered with tape libraries.
- It is tempting to generalize our vertical migration method to an entire storage hierarchy with main memory and possibly distributed memory [SW97, VLN97] considered in addition to secondary and tertiary storage. However, the bookkeeping overhead for the access predictions may then become a critical issue, as they require a substantial fraction of the managed resources themselves, namely, main memory. Under these conditions, the bookkeeping information should itself be subdivided and dynamically migrate between secondary and primary storage, and we could employ the weight of a document as the criterion for whether we should cache the information on a document's outgoing transitions. Thus, the mathematical basis of our approach should pay off also at the meta level.

- The continuous-time Markov chain model upon which our method is based may be used not only for vertical data migration, but also for data placement (e.g., clustering) on both secondary and tertiary storage devices and for horizontal data migration between devices at the same storage level, for example, for dynamic and incremental load balancing among disks. As Markov chain models are substantially richer than the stationary-probability models that have mostly been suggested for these purposes [Wo83, TCG96, SWZ94], improvements of performance could be expected.

Encouraged by the very positive results that we obtained for the architectural setting of the current paper, we have started investigating the above issues.

References

[All90] A.O. Allen: *Probability, Statistics, and Queueing Theory with Computer Science Applications*, Academic Press, 1990

[ABCO96] V. Almeida, A. Bestavros, M. Crovella, A. de Oliveira: *Characterizing Reference Locality in the WWW*, Int. Conf. on Parallel and Distributed Information Systems (PDIS), Miami Beach, 1996

[Be96] A. Bestavros: *Speculative Data Dissemination and Service in Distributed Information Systems*, Int. Conf. on Data Engineering, New Orleans, 1996

[CFKL95a] P. Cao, E.W. Felten, A.R. Karlin, K. Li: *A Study of Integrated Prefetching and Caching Strategies*, ACM SIGMETRICS Conf., 1995

[CFKL95b] P. Cao, E.W. Felten, A.R. Karlin, K. Li: *Implementation and Performance of Integrated Application-Controlled Caching, Prefetching and Disk Scheduling*, Technical Report TR-CS95-493, Princeton University, 1995

[CK89] E.E. Chang, R.H. Katz: *Exploiting Inheritance and Structure Semantics for Effective Clustering and Buffering in an Object-Oriented DBMS* , ACM SIGMOD Conf., Portland, 1989

[CR94] L.T. Chen, D. Rotem: *Optimizing Storage of Objects on Mass Storage Systems with Robotic Devices*, Int. Conf. on Extending Database Technology (EDBT), Cambridge, UK, 1994

[CH91] J.R. Cheng, A.R. Hurson: *On The Performance Issues of Object-Based Buffering*, Int. Conf. on Parallel and Distributed Information Systems (PDIS), Miami Beach, 1991

[Co88] G. Copeland, W. Alexander, E. Boughter, T. Keller: *Data Placement in Bubba*, ACM SIGMOD Conf., Chicago, 1988

[CKV93] K.M. Curewitz, P. Krishnan, J.S. Vitter: *Practical Prefetching via Data Compression*, ACM SIGMOD Conf., Washington, DC, 1993

[CSIM] *CSIM17 User's Guide*, Mesquite Software Inc., Austin

[DevSim] M. Gillmann, W. Gross: *User's Guide of DevSim- A Library of Secondary and Tertiary Storage Device Simulations (in German)*, University of the Saarland, 1996

[FC91] D.A. Ford, S. Christodoulakis: *Optimal Placement of High-Probability Randomly Retrieved Blocks on CLV Optical Disks*, ACM Transactions on Information Systems, 9(1), 1991

[GK94] C.A. Gerlhof, A. Kemper: *Prefetch Support Relations in Object Bases*, Int. Workshop on Persistent Object Stores (POS), 1994

[GKKM93] C.A. Gerlhof, A. Kemper, C. Kilger, G. Moerkotte: *Partition-Based Clustering in Object Bases: From Theory to Practice*, Int. Conf. on Foundations of Data Organization and Algorithms (FODO), Chicago, 1993

[GMW94] L. Golubchik, R. Muntz, R.W. Watson: *Analysis of Striping Techniques in Robotic Storage Libraries*, Technical Report, University of California, Los Angeles, 1994

[GP87] J. Gray, F. Putzolu: *The 5 Minute Rule for Trading Memory for Disc Accesses and the 10 Byte Rule for Trading Memory for CPU Time*, ACM SIGMOD Conf., San Francisco, 1987

[HS96] B.K. Hillyer, A. Silberschatz: *Random I/O Scheduling in Online Tertiary Storage Systems*, ACM SIGMOD Conf., Montreal, 1996

[KPR92] A.R. Karlin, S.J. Phillips, P. Raghavan: *Markov Paging*, IEEE Symposium on Foundations of Computer Science, 1992

[LLW95] S.W. Lau, J.C.S. Lui, P.C. Wong: *A Cost-effective Near-line Storage Server for Multimedia System*, Int. Conf. on Data Engineering, Taipeh, 1995

[MKK95] F. Moser, A. Kraiss, W. Klas: *L/MRP - A Buffer Management Strategy for Interactive Continuous Data Flows in a Multimedia DBMS*, VLDB Conf., Zurich, 1995

[ML97] M. Myllimaki, M. Livny: *Relational Joins for Data on Tertiary Storage*, Int. Conf. on Data Engineering, Birmingham, UK, 1997

[Nel95] R. Nelson: *Probability, Stochastic Processes, and Queueing Theory - The Mathematics of Computer Performance Modeling*, Springer, 1995

[NKT97] T. Nemoto, M. Kitsuregawa, M. Takagi: *Simulation Studies of the Cassette Migration Activities in a Scalable Tape Archiver*, Int. Conf. on Database Systems for Advanced Applications, Melbourne, 1997

[OOW93] E.J. O'Neil, P.E. O'Neil, G. Weikum: *The LRU-K Page Replacement Algorithm For Database Disk Buffering*, ACM SIGMOD Conf., Washington, DC, 1993

[PZ91] M. Palmer, S.B. Zdonik: *Fido: A Cache that learns to fetch*, VLDB Conf., Barcelona, 1991

[PGG+95] R.H. Patterson, G.A. Gibson, E. Ginting, D. Stodolsky, J. Zelenka: *Informed Prefetching and Caching*, ACM Symposium on Operating Systems Principles, 1995

[RW94] C. Ruemmler, J. Wilkes: *An Introduction to Disk Modelling*, IEEE Computer 27(3), 1994

[Sa95] S. Sarawagi: *Query Processing in Tertiary Memory Databases*, VLDB Conf., Zurich, 1995

[SSV96] P. Scheuermann, J. Shim, R. Vingralek: *WATCHMAN: A Data Warehouse Intelligent Cache Manager*, VLDB Conf., Bombay, 1996

[SW97] M. Sinnwell, G. Weikum: *A Cost-Model-Based Online Method for Distributed Caching*, Int. Conf. on Data Engineering, Birmingham, UK, 1997

[SWZ94] P. Scheuermann, G. Weikum, P. Zabback: *Disk Cooling in Parallel Disk Systems*, IEEE Data Engineering Bulletin 17(3), 1994

[Smi81] A.J. Smith: *Long Term File Migration: Development and Evaluation of Algorithms*, Communications of the ACM 24(8), 1981

[Sto91] M. Stonebraker: *Managing Persistent Objects in a Multi-Level Store*, ACM SIGMOD Conf., Denver, 1991

[TG84] J.Z. Teng, R.A. Gumaer: *Managing IBM Database 2 Buffers to Maximize Performance*, IBM Systems Journal 23(2), 1984

[Tij94] H.C. Tijms: *Stochastic Models - An Algorithmic Approach*, John Wiley & Sons, 1994

[TCG96] P. Triantafillou, S. Christodoulakis, C. Georgiadis: *Optimal Data Placement on Disks: A Comprehensive Solution for Different Technologies*, HERMES Technical Report, Multimedia Systems Institute of Crete, 1996

[TN91] M.M. Tsangaris, J.F. Naughton: *A Stochastic Approach for Clustering in Object Bases*, ACM SIGMOD Conf., Denver, 1991

[TN92] M.M. Tsangaris, J.F. Naughton: *On the Performance of Object Clustering Techniques*, ACM SIGMOD Conf., San Diego, 1992

[VLN97] S. Venkataraman, M. Livny, J.F. Naughton: *Memory Management for Scalable Web Data Servers*, Int. Conf. on Data Engineering, Birmingham, UK, 1997

[WZ86] H. Wedekind, G. Zoerntlein: *Prefetching in Realtime Database Applications*, ACM SIGMOD Conf., Washington, DC, 1986

[WHMZ94] G. Weikum, C. Hasse, A. Moenkeberg, P. Zabback: *The COMFORT Automatic Tuning Project*, Information Systems 19(5), 1994

[Wo83] C.K. Wong: *Algorithmic Studies in Mass Storage Systems*, Computer Science Press, 1983