The VLDB Journal (1998) 7: 141–162

The VLDB Journal

Integrated document caching and prefetching in storage hierarchies 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, WWW: http://www-dbs.cs.uni-sb.de/

Edited by M. Jarke. Received January 1, 1998 / Accepted May 27, 1998

Abstract. Large multimedia document archives may hold a major fraction of their data in tertiary storage libraries for cost reasons. This paper develops an integrated approach to the vertical data migration between the tertiary, secondary, and primary storage in that it reconciles speculative prefetching, to mask the high latency of the tertiary storage, with the replacement policy of the document caches at the secondary and primary storage level, and also considers the interaction of these policies with the tertiary and secondary storage request scheduling.

The integrated migration policy is based on a continuous-time Markov chain model for predicting the expected number of accesses to a document within a specified time horizon. Prefetching is initiated only if that expectation is higher than those of the documents that need to be dropped from secondary storage to free up the necessary space. In addition, the possible resource contention at the tertiary and secondary storage is taken into account by dynamically assessing the response-time benefit of prefetching a document versus the penalty that it would incur on the response time of the pending document requests.

The parameters of the continuous-time Markov chain 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. The system makes profitable use of the Markov chain model also for the scheduling of volume exchanges in the tertiary storage library. Detailed simulation experiments with Web-server-like synthetic workloads indicate significant gains in terms of client response time. The experiments also show that the overhead of the statistical bookkeeping and the computations for the access predictions is affordable.

Key words: Performance – Caching – Prefetching – Scheduling – Tertiary storage – Stochastic modeling – Markov chains

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 or days, 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 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 s or even minutes. Thus, it is crucial that the currently hot documents are indeed held at least 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. In addition, some very hot documents may be held in an in-memory cache at the primary storage level. Therefore, good cache *replacement* policies for variable-length documents are extremely important for the overall server performance. Furthermore, the vertical

data migration between the tertiary, the secondary, and the primary storage and thus the cache hit rate can be further improved by employing "intelligent" *prefetching* policies, so that the high latency of the tertiary storage can be masked from the client in many cases.

Designing good replacement and, especially, prefetching policies for the vertical data migration between tertiary, secondary, and primary storage is substantially more difficult than standard DBMS buffer management. To realize the difficulties, consider a straightforward approach that eagerly initiates the prefetching of documents whenever a tertiary storage volume is online, and keeps prefetching until it runs out of (secondary storage) cache space. Such a simplistic approach is bound to fail, as it does not properly assess the various potential bottlenecks.

1. *Cache hit rate.* Cache space is scarce and valuable, so that only sufficiently worthy documents should be prefetched and/or kept in the cache. This calls for policies that are based on good estimates of the near-future access patterns, which is far beyond the usual DBMS page prefetching on behalf of sequential scans [TG84]. Three major problems that arise in this context are:
   a. We need to quantitatively assess the cache-worthiness of a previously uncached document versus a cached one. Standard algorithms like LRU lack appropriate bookkeeping information about currently non-cached data; hence, they are susceptible to swamping the cache with newly fetched, but unworthy documents [OOW93].
   b. 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.
   c. 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. We need a quantitative understanding of when to throttle the prefetching activity and how to identify the most worthy prefetching candidates.
2. *Resource contention at the tertiary storage.* Both cache replacement and prefetching interfere with the scheduling policy of the tertiary storage library. This involves two issues:
   a. The transfer rate of the tertiary storage is fairly limited. So prefetching can lead to substantial queuing delays in serving other, pending document requests. Therefore, to control these contention effects, the utilization of the tertiary storage drives must be taken into consideration.
   b. The robot arm of the tertiary storage library is a potential bottleneck, as it incurs a high latency in every volume exchange. Throughput considerations, therefore, suggest minimizing volume exchanges, but this may adversely affect response time. So the scheduling of volume exchanges needs to be planned carefully.
3. *Resource contention at the secondary storage.* The vertical migration of documents also interferes with the scheduling policy for the disk(s) on which the secondary storage cache resides. This has two aspects:
   a. As the migration from tertiary storage into the secondary storage cache goes through primary storage, it may incur a fairly high write I/O load on the cache disk(s). On the other hand, client requests are ultimately served from memory-resident network buffers; so all secondary-storage cache hits require read I/Os from the cache disk(s). These two types of disk activities need to be reconciled under possibly high contention.
   b. Since the primary storage also provides a relatively small amount of even faster cache space, the replacement decisions at this storage level affect the intensity of having to read documents from the disk cache. However, using memory for this extra caching effect alone would potentially block the document migration path from tertiary onto secondary storage, because of insufficient intermediate buffers in memory. This tradeoff needs to be considered thoroughly.

The problems outlined above indicate the complexity of managing a storage hierarchy in an intelligent manner. Our approach in this paper is heuristic in that it addresses each potential bottleneck separately, but each one of the developed building blocks is based on rigorous mathematical reasoning along with a careful assessment of its bookkeeping and computational overhead.

1.2 Contribution and outline

This paper develops a unified approach to cache replacement and speculative prefetching, based on a stochastic model for predicting document accesses, and integrates this vertical migration policy with the scheduling policies of the tertiary storage library and the secondary storage cache. In doing so, we aim to minimize the response time of client requests. To this end, our approach considers the impact of the following potential bottlenecks: the cache hit rate at the secondary and primary storage level, the contribution of queuing delays at the tertiary storage level, and the potential queuing at the secondary storage disk cache. We are not aware of any similarly comprehensive work on managing large near-line document archives.

In addition to advancing the state of the art from a system design viewpoint, a 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 such as 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 approach in this paper was initially suggested in the conference paper [KW97], but that paper did not consider primary storage caching, nor did it take into account the performance issues that result from writing cached documents onto secondary storage. In the current, substantially extended paper, we consider these issues. Furthermore, we generally extend the quantitative cost modeling of our method in that we capture the replacement costs that arise from dropping a document at a certain storage level, and we capture the resource contention in a tertiary storage jukebox with multiple drives more accurately than in [KW97]. These issues were not of interest in [KW97], as we considered only a two-level storage hierarchy and a single-drive jukebox there, but they are of crucial importance for the general three-level storage hierarchy of the current paper.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 presents our assumptions on the overall system architecture. Section 4 develops a continuous-time Markov chain model for predicting near-future document accesses. Section 5 presents the integrated vertical migration policy that incorporates prefetching, replacement, and the scheduling policies of the tertiary and the secondary storage. In Sect. 6, we discuss the bookkeeping overhead of our policy in terms of CPU and memory consumption. Section 7 gives an overview of our prototype implementation. Section 8 presents experimental performance results based on simulation. Section 9 discusses several extensions and generalizations of the developed approach. Section 10 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, CTZ97] 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, Jo98].

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, TP97]. The effect of object prefetching is implicitly achieved (on a per-page basis) also by intelligently clustering objects into pages [CK89, TN91, TN92, GKKM93]. In the field of mobile computing, [KP97] has pursued a similar approach where reference "distances" in user and program access patterns are used for predictive file hoarding.

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] has used 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. 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.

Approaches that aim to reconcile the replacement and prefetching policies for an in-memory page cache are [GK94], [CFKL95a, CFKL95b], [AGL97], 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] and [AGL97] develop 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). [CFKL95a, CFKL95b] analyzes bounds on the suboptimality of various heuristics, whereas [AGL97] address the issue as an integer optimization problem and present a linear-programming relaxation for finding an optimum prefetching schedule. 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.

The only work known to us that considers appropriate thresholds for the throttling of prefetching activities from a stochastic viewpoint is [JK98], in the context of prefetching data from Web servers. Based on a cost model, a global prefetching threshold is derived in terms of minimum document access probabilities so as not to overload the network. However, this approach focuses on per-client prefetching and does not address server cache management. Furthermore, access predictions are based only on discrete document access probabilities with a single-step lookahead, and document-specific user reaction times are not considered.

3 System architecture

We consider a document server with a three-tier storage hierarchy:

- *primary storage (PS)* in main memory (assumed to be shared among all processors if the server is an SMP machine) which serves both as a cache for very hot documents and transfer buffer for document migrations,
- *secondary storage (SS)* on disk(s) as a cache for hot and warm documents,
- *tertiary storage (TS)* in the form of an optical-disk jukebox with one or more *drives* and a robot arm for the exchange of *volumes* (i.e., optical disk platters), which serves as the permanent home of documents.

At any given point the TS has loaded a subset of its volumes into its drives, which changes with every volume exchange of the jukebox. Thus, the TS itself constitutes two separate access levels, altogether leading to a four-tier storage hierarchy with the following two lower-half levels:

- *online tertiary storage (TS-on)* and
- *offline tertiary storage (TS-off)*.

We denote by $docs(i)$ the set of documents that reside at level i of the storage hierarchy. We assume that the upper two levels of the storage hierarchy form true caches of the lower levels in that a document that currently resides at a higher level is still available from the TS level(s). We do not assume, however, that $docs(PS)$ is always a subset of $docs(SS)$, as the data transfer path from TS to SS is through PS. Then, it can occur that a document is brought from TS to PS with the intention to write it onto SS, but, by the time this disk write is about to start, the document may have become unworthy of being cached. This situation is not even that exceptional; consider, for example, a document that is brought into PS due to a high probability that it is requested within the near future by a single session. Once it is loaded into PS, the request may be served from the PS before the document is written to the SS. After the document is requested by the session, it may become fairly cold, so that it should not be kept in SS or PS after the request is satisfied.

The overall system architecture of our approach is depicted in Fig. 1. The server can deliver documents to clients only from its PS. To bring a document into PS, the server always retrieves the document from the highest storage level at which (a copy of) the document currently resides (possibly the PS cache itself). So, in particular, documents that are requested from the SS cache are first brought into PS. When the document is copied between storage levels or dropped from a certain level, we speak of a data migration. *Upward data migrations* from a lower level to a higher level take place upon the following events:

– *fetching* an explicitly requested document from TS-on into PS,

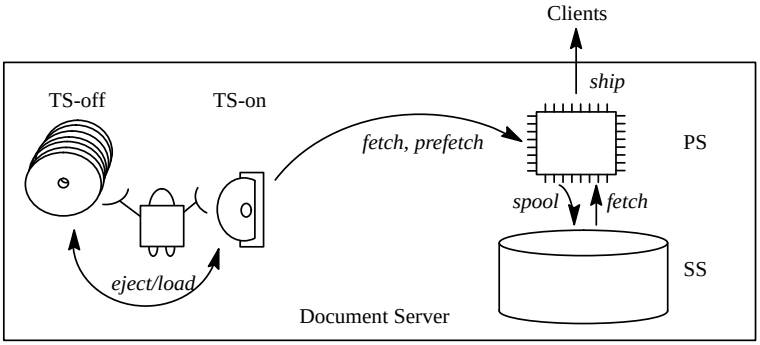

Fig. 1. Overview of the system architecture

– *fetching* an explicitly requested, previously cached document from the SS cache into PS,
– *prefetching* a document from TS-on into PS in the speculative anticipation of near-term requests,
– *spooling* a previously fetched or prefetched document from PS onto SS (i.e., writing it onto disk), which can be viewed as the second phase of prefetching a document from TS-on onto SS via PS,
– *loading* a previously offline volume into a TS drive, which can be viewed as an implicit migration of a set of documents from TS-off to TS-on.

Downward data migration results implicitly from dropping a document that used to reside at a higher level, or from ejecting an online volume from a TS drive. The implicit migrations that result from volume exchanges are performed by the *TS scheduler*. For modularity, we assume that the TS scheduler is a separate (and thus exchangeable) component within the overall system architecture. We generally assume that the TS is the most critical bottleneck, and that the SS cache may be performance-critical, too, as its disk(s) need(s) to sustain a very high read/write load for the dynamic, more or less continuous transfer of documents between PS and SS. The transfer between PS and SS is managed by the *SS scheduler*. We disregard all other scheduling issues. In particular, we do not consider CPU or network contention. Such extensions would be feasible within our framework, but would complicate the algorithms.

4 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 $d_i \in D$, $i = 1 \ldots N$. Furthermore let S denote the set of currently active user sessions $s_j \in S$, $j = 1 \ldots |S|$, 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 Sect. 4.1. We will then show in Sect. 4.2 how multiple sessions and, particularly, the dynamic arrivals of new sessions can be incorporated into the model. The Markov chain model implies that interaction times between successive requests of the same user are exponentially distributed, which has been reasonably well confirmed by the analysis of WWW server traces. In addition, we discuss possible extensions to capture general distributions in Sect. 4.3.

4.1 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 $1, 2, \ldots N$ is uniquely described by a matrix $\mathbf{P} = (p_{ij})$ 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:

$$\bar{p}_{ij} = \begin{cases} \frac{v_i}{v} * p_{ij} \ , & j \neq i \\ 1 - \frac{v_i}{v} \ , & j = i \end{cases}$$

where

$$v = \max\{v_i | i = 1 \ldots N\} \ . \quad (1)$$

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]

$$\bar{p}_{ij}^{(m)} = \sum_{k=1}^{N} \bar{p}_{ik}^{(m-1)} \bar{p}_{kj} \quad \text{with} \quad \bar{p}_{ij}^{(0)} = \begin{cases} 1 \text{ if } & i = j \ , \\ 0 \text{ otherwise.} \end{cases} \quad (2)$$

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

$$p_{ij}(t) = \sum_{m=0}^{\infty} e^{-vt} \frac{(vt)^m}{m!} * \bar{p}_{ij}^{(m)}, \quad \text{for all } i, j \text{ and } t > 0 \ . \quad (3)$$

We will show in Sect. 6 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 prefetch 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 Sect. 5.6. 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]:

$$E_{ij}(t) = \frac{1}{v} * \sum_{n=1}^{\infty} \left(e^{-vt} \frac{(vt)^n}{n!} * \sum_{m=0}^{n-1} \bar{p}_{ij}^{(m)} \right) . \quad (4)$$

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:

$$E[\text{number of accesses to } d_k \text{ in time } t] = \sum_{j=1}^{N} v * E_{ij}(t) * \bar{p}_{jk} \quad (5)$$

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 prefetching the document from tertiary storage and/or keeping it in the secondary-storage cache.

4.2 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:

$$N_{\text{spec}}(d_k, t) := E[\text{total number of accesses to } d_k \text{ in time } t] = \sum_{s \in S} \sum_{j=1}^{N} v * E_{d(s),j}(t) * \bar{p}_{jk} \ , \quad (6)$$

where $d(s)$ is the document on which session s currently resides (i.e., the current session state). We will refer to this expectation value as the expected number of *speculative requests*, and will denote it as $N_{\text{spec}}(d_k, t)$.

In addition to the derived expectation value, a second metric of potential interest is the probability that there is at least one request to a given document within time t. The transient analysis of CTMC models yields a closed formula for this first-visit probability, too [Tij94]. However, this derivation is substantially more costly than the above calculation of the expected number of accesses. In particular, it involves additional traversals of paths in the Markov chain to obtain the probabilities that a certain document will *not* be accessed within time t. To avoid this very costly computation, we rather advocate the following approximative estimation of the probability for at least one access to a given document. We view the overall process of requests to a document d_k as a Poisson process with rate

$$\bar{\lambda}(d_k) = E[\text{number of accesses to } d_k \text{ in time } t]/t \ . \qquad (7)$$

Then we can estimate the first-visit probability for d within time horizon t as

$$\Pi_{\text{spec}}(d_k, t) = 1 - e^{-\bar{\lambda}(d_k)*t} \ . \qquad (8)$$

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 newly arriving sessions into the CTMC framework. We simply add to the CTMC model additional, fictitious states $N+1, \ldots, N+c$ that represent all currently passive clients (which do not have a session in progress) from which we expect that their arrival probability within time t is above a given threshold p_{min}. We assume the overall arrival of new sessions to be a Poisson process with rate λ. Then, the state residence times of the c "passive-client" states (i.e., the time until a passive client starts a new session) are identically distributed, following an exponential distribution with mean value c/λ. The number of fictitious states that we consider to represent the entirety of currently passive clients can then be bounded by solving the following inequality for c, and this approximation "misses" at most a fraction of p_{min} of the overall session arrivals (with p_{min} set to 0.001, for example):

$$1 - e^{(-\frac{\lambda}{c}*t)} \geq p_{\text{min}} \ . \qquad (9)$$

The transition probabilities $p_{N+i,j}$ ($i = 1 \ldots c$) 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 Sect. 4.1 with states $N+1$ through $N+c$ added to the various formulas, and the only thing to do in addition is to logically add c fictitious sessions, one session residing on each of the states $N+1, \ldots, N+c$, to the set S of sessions over which the per-session expectations are summed up (formula 6). However, as the number of near-future "out-of-the-blue" accesses to a document are the same for all c sessions, the computation of the overall "out-of-the-blue" accesses can be greatly simplified by just multiplying the expected number of accesses done by a single arriving session with the number of considered sessions c. Note that, for the same reason, the transition probabilities $p_{N+i,j}$ have to be kept only once.

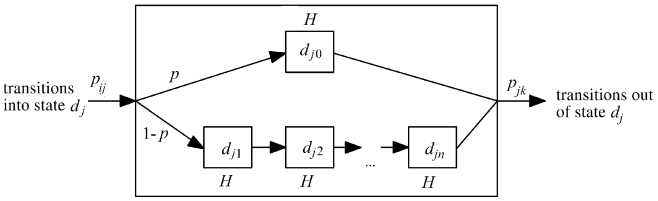

Fig. 2. Example modeling of state d_j with generally distributed state residence time

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.

4.3 Incorporating generally distributed state residence times

Analyses of WWW server traces have shown that exponentially distributed state residence times capture real interaction patterns reasonably well. In addition, our model could be easily extended to consider general distributions by approximating the actual state residence time distribution with a generalized Erlang distribution, which is a random sum of Erlang-k distributions with the same scale parameter but different k values [Tij94]. This way, a state with a general distribution of its residence times can be mapped into a "superstate" consisting of branching sequences of states with exponentially distributed residence times. The "substates" with exponentially distributed residence times can then be directly incorporated into the CTMC framework. Figure 2 depicts an example of a superstate d_j with residence time following an $E_{1,n}$ distribution, which is a random (i.e., probabilistically weighted) sum of an Erlang-1 and an Erlang-n distribution. The $E_{1,n}$ distribution can be used for a wide spectrum of general distributions with coefficient of variation c_x in the range [Tij94]:

$$\frac{1}{n} \leq c_x^2 \leq \frac{n^2+4}{4*n} \ . \qquad (10)$$

The branching probability p and the mean state residence times H can be determined from the moments of the superstate's actual residence time distribution.

Once the substates $d_{j0}, \ldots, d_{jn}$ are incorporated into the CTMC model, the expected number of accesses to the document d_j within time t is obtained by summing up the expected number of accesses within time t over the "entry" substates d_{j0} and d_{j1}.

Clearly, as this method involves introducing additional states into the CTMC, it incurs higher overhead. However, the mapping is only needed for documents whose residence

times cannot be approximated by an exponential distribution at all. Furthermore, the number of additional states is rather small for many unimodal distributions (i.e., distributions whose probability density function has a single maximum), which are the distributions of most interest in our application context.

5 Integrated migration policy

This section presents our vertical migration policy that aims to reconcile the prefetching from tertiary storage into secondary and primary storage, the replacement on primary and 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 in Sect. 4. We will first give an overview of the algorithm in Subsect. 5.1. Then, we present the prefetching and replacement algorithm in Subsect. 5.2, the prefetch request scheduling at the TS in Subsect. 5.3, and the write request scheduling at the SS cache in Subsect. 5.4. Subsequently, we elaborate on the tertiary storage scheduling of volume exchanges in Subsect. 5.5. Finally, Subsect. 5.6 discusses the fine-tuning of the lookahead time horizon used by the CTMC predictions.

5.1 Overview of the algorithm

The overall approach is to quantitatively assess the near-term worthiness (or "weight") of each document with respect to improving the mean response time of document requests. To this end, the algorithm aims to place the most worthy documents in SS with the highest ranked documents among these also cached in PS. This placement, however, is not a static one, but is actually the result of the migration steps that take into account the dynamic evolution of the server load. In measuring the worthiness of a document, the following three aspects are taken into account.

- The *near-term heat* of a document, which is the document's access frequency estimated for a certain lookahead time window of size T. In contrast to earlier work based on stationary, long-term heat, our approach considers the current state of the server's sessions with active clients and reacts dynamically to changes of these states and other general load parameters. The estimated near-term heat of a document d, $NH(d,T)$, is the expectation value *E[total number of accesses to d within time T]* that we derived in the previous section within the CTMC model.
- Given that we deal with variable-size documents, we need to normalize the heat metric in order to obtain the benefit-per-space unit. To this end, we divide the heat of a document d by the *size* of d, $S(d)$, to derive the *near-term temperature* of d, $NT(d,T) = NH(d,T)/S(d)$. (See also [Co88, SSV96] for similar considerations.)
- Finally, it may be worthwhile to discriminate documents that reside on different levels of the storage hierarchy, because this incurs different costs in the retrieval of documents if the documents are not cached at the highest level(s). For example, when deciding the worthiness of two documents with regard to caching them in PS, it is important if one document can still be fetched from SS, whereas the other resides only in TS, possibly even TS-off. This consideration is captured by the *replacement cost* of a document d, $RC(d)$, which is the response-time cost that is incurred by retrieving d from the highest one among the lower storage levels at which (a copy of) it currently resides. Then, the worthiness of a document should be proportional to its replacement cost.

Putting these three aspects together yields the following definition of the *weight* metric:

$$\text{weight}(d,T) = (NH(d,T)/S(d)) \times RC(d) \ .$$

Based on this weight metric, the rationale of our approach is the following. We maintain a list of "interesting" documents containing the top m documents in terms of the weight metric, where m is chosen such that these documents would together completely fill up the available cache space on SS (assuming that the SS cache is much larger than the PS cache). A document d from this list should then be prefetched into PS (and then further "spooled" onto SS) or SS (via a short-term PS buffer), respectively, if and only if it is not yet cached and its weight exceeds the maximum weight among the documents that would have to be dropped from PS or SS as replacement victims in order to make space for d. Here, for d itself, the replacement cost refers to its current storage level before prefetching it, whereas for the PS or SS replacement victims, the replacement cost refers to their highest storage levels below the level from which they would be dropped.

Document weight is the decisive metric for tentatively identifying prefetching candidates and the corresponding replacement victims. However, the prefetching itself should be reconsidered as late as possible as input parameters change over time. Thus, when the TS is about to start the data transfer for a prefetching request, the weights of the to-be-prefetched document and the selected replacement victims in PS and SS are re-evaluated. If it turns out that the prefetching request is no longer worthwhile, then it is cancelled at this point. A second and even more important aspect that may lead to the cancellation of prefetching requests is that overly aggressive prefetching may lead to contention at the TS drives or the robot arm of the TS. Therefore, it is crucial to consider also the potential interdependencies between data migrations and the TS request scheduling.

The solution to this throttling problem is to assess for each individual prefetching request, when it is about to be served from an online volume, the actual response-time benefit of the prefetching and the penalty that this request would incur for future requests. To this end, we define

- the *benefit* of a prefetching request as the aggregated savings in the response time of future requests to the prefetched document, and
- the *penalty* of a prefetching request as the aggregated delay of other requests, i.e., the product of the affected, i.e., delayed, requests and their response-time delay.

In Sect. 5.3, we will derive formulas for the benefit and penalty metrics. The benefit and the penalty of a prefetching request are computed when the request is about to be initiated, to reflect up-to-date session-state and load parameters. Certain intermediate formulas may be precomputed at this

point to make the benefit/penalty assessment sufficiently efficient for online decisions. Then, the prefetching request is initiated if and only if its benefit exceeds its penalty. Otherwise it is cancelled.

A similar consideration on the need for activity throttling arises at the SS. Here, the potential point of contention is that read requests that fetch cached documents from SS into PS for delivery to the client may be delayed by write requests that "spool" prefetched or fetched documents from the PS cache onto SS for further caching. Again, we address this issue by assessing the benefit and penalty of a document write request, using formulas that will be derived in Sect. 5.4.

In summary, the integrated migration policy can be viewed as a three-stage decision process, where the second and third stage each perform two comparisons.

- *Stage 1.* Tentatively identify "speculative" prefetching candidates, based on a ranking of documents by descending weight, and insert prefetching requests into the corresponding queue of the TS volume.
- *Stage 2.* When a prefetching request is about to be served from an online volume of the TS, select the lowest ranked documents of the target level (i.e., PS or SS) as replacement victims and compare the weights of the to-be-prefetched document and the chosen victims. If the document weight is smaller than the maximum weight among the victims, then the prefetching request is cancelled. If the prefetching request is still considered worthwhile, its benefit and penalty are computed. When the penalty of the prefetching request, in terms of delaying other requests, exceeds its benefit, then the prefetching request is cancelled.
- *Stage 3.* When a prefetched or fetched document is about to be written from PS to SS, the weight comparison and the benefit/penalty comparison are repeated once more, and the write request is cancelled if one of these comparisons indicate that the document is no longer sufficiently worthy.

5.1.1 Example scenario

As an example for the possible actions of this three-stage policy consider the following scenario depicted in Fig. 3. The figure shows (copies of) documents A through K that reside in PS, SS, and the online TS. Documents on the online TS are shown in the order in which they would be served by a sequential volume scan. The near-term heat of these documents is shown in parentheses; for simplicity, assume that all documents have the same unit size.

Now consider what happens when clients explicitly request the documents F, G, and C, and the TS scan has reached the position of document H (Fig. 3a). Document H could be worth being cached in PS, as there are potential victim documents B with lower near-term heat and A which is cached on disk. But now consider the penalty that prefetching H at this point would incur on other requests. We have explicit client requests to F and G pending; both would be delayed by the prefetching of H. So we need to compare the benefit of prefetching H with its penalty (with regard to F and G) quantitatively. Here, we need to consider more detailed parameters, which are not given in the figure. So let us simply assume that the benefit of H is lower than its penalty. The decision then is to cancel the prefetching request for H.

Next, document F must be fetched into PS, as there is an explicit, pending request for it. So we need to determine a replacement victim. B has the lowest near-term heat, but once it is dropped it would have to be retrieved again from TS (possibly offline TS), whereas A could still be retrieved again at a much lower cost from SS. Thus, taking into account the replacement costs of A and B suggests choosing A as the replacement victim at the PS level. Now let us focus for another moment on what happens further to document F once it is brought into PS (Fig. 3b). It is sent to the requesting client, and at the same time, the SS scheduler considers writing F to the SS cache, where it may be kept for a while. However, this write I/O may delay fetching document C from the SS cache, which is also requested by a client (think of the client request for C to arrive immediately after the fetching of F from TS). Now we need to tradeoff the benefit of keeping F in the SS cache (i.e., avoiding future TS accesses) versus the penalty that it incurs on the reading of C from SS. One outcome of this comparison could be that the response-time delay for C would be so high that the writing of F should be cancelled. Since C also needs space in PS, this decision would probably imply that F is dropped from PS right after completing its delivery to the requesting client (Fig. 3c).

Now we have documents B and C in the PS cache, and we are about to prefetch document K. Its weight is reassessed against that of the lowest ranked PS document, C, and K is clearly found to be worth being prefetched into PS. The final question then is whether this would have an overly adverse effect on further pending requests, namely, the request to document G in our scenario. Again, we compare the benefit of caching K in PS versus the response-time penalty that this prefetching request would incur on the request to G. Here, let us assume that the benefit exceeds the penalty. So, K is prefetched, first into PS, where we need to reclaim space by dropping C. Assuming that the weight of K exceeds the weight of E and as there is no pending requests on SS, K is spooled to SS by dropping document E from SS (Fig. 3d). If K is spooled to SS quickly enough, then the final pending request to G could immediately re-use the PS space that was temporarily occupied by G (Fig. 3e). Here, we also see that it is desirable to spool PS documents to SS as fast as possible in order to reclaim valuable PS space, and in the current situation there is no contention with any outstanding SS read I/Os.

This scenario should give a more concrete, albeit merely exemplary and mostly qualitative impression on the various decisions that need to be made in the overall migration algorithm.

5.2 Prefetching and replacement of documents

As the worst case access time to tertiary storage is orders of magnitude higher than a secondary-storage disk access, an important initial objective of our migration policy (to be revised shortly) would be to maximize the number of document requests that can be served from the disk cache, that is,

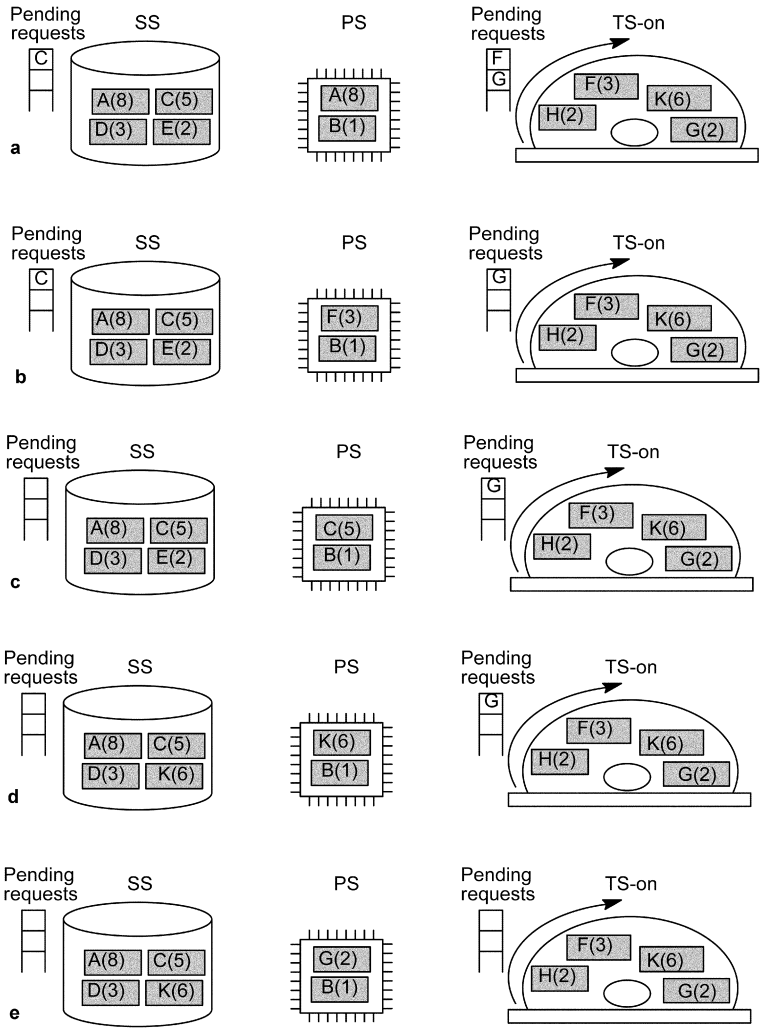

Fig. 3a–e. Example scenario for the integrated migration policy. **a** Initial situation. **b** Situation after fetching F into PS. **c** Situation after fetching C into PS. **d** Situation after prefetching K into PS and spooling it onto SS. **e** Situation after fetching G into PS

maximize the cache hit rate. Then, the caching worthiness of documents is reflected by their near-term heat, which is

$$NH(d,T) = N_{\text{spec}}(d,T)\ , \tag{11}$$

where $N_{\text{spec}}(d,T)$ denotes the Markov-chain predicted number of *speculative requests* to d, i.e., the expectation value according to formula (6). By normalizing the caching worthiness on a per-byte basis, we derive the near-term temperature:

$$NT(d,T) = N_{\text{spec}}(d,T)/S(d)\ . \tag{12}$$

If maximizing the cache hit rate were indeed our sole objective, then we would now rank documents in descending order of the near-term temperature, keeping the highest two fractions in the PS and SS cache, respectively. However, in order to minimize the mean response time, we also need to factor the replacement costs of documents into the ranking metric, which leads to the weight of a document given by

$$\text{weight}(d,T) := NH(d,T) * RC(d)/S(d)\ . \tag{13}$$

Then, at each point of time, the ideal assignment of documents to the PS and SS caches should reflect the ranking of documents by descending weight. To approach this ideal ranking behavior by an online decision-making algorithm, our migration algorithm considers prefetching requests for those documents whose weight exceeds that of currently SS-resident documents. The algorithm actually maintains a ranked list in descending weight order, where the top m documents are considered for prefetching. The value of m is chosen such that the space capacity of the SS cache would be completely exhausted by those m documents. Each document which is element of the top m documents and which is not cached on primary or secondary storage has to be prefetched from tertiary storage. At the time, a document prefetch is scheduled, the weight of the document to prefetch is compared with the weight of the corresponding replacement victims and the prefetch may be cancelled. The replacement victims are determined based on ascending weight order starting from the bottom of the sorted list.

At this point, let us consider, in more detail, the bookkeeping costs of this policy. In assessing the weight of documents, the information that is readily available anyway is the number of pending requests and the document size. The (expected) number of speculative requests is derived using the CTMC model. This is not exactly computationally inexpensive, but it is the core of our overall approach. Thus, we assume that this information is available online for all relevant documents. Finally, consider the replacement cost of a document. This is much more problematic than it seems

at first glance. What we need here is a bookkeeping of the highest storage level at which a cached documents would reside if it were dropped from a cache, since the replacement cost of such a document depends on that storage level. This information is, of course, available, but is highly dynamic. In particular, whenever a TS volume is ejected from a drive and thus changes its status from online to offline, the replacement cost of a cached document changes (unless we consider the PS cache and the document also resides on SS). Unfortunately, this would imply that the ranking of documents in the SS and PS caches has to be re-computed with every volume exchange of the TS. Contrast this with the much more stable values of the near-term heat, which can also be evaluated lazily.

These considerations on the bookkeeping costs for the weight metric lead us to the following simplifications. For the documents in the PS cache, we maintain the ranking by the weight metric. This appears to be justified, as the number of documents in the PS cache is relatively low. (Recall that we expect fairly large documents in a multimedia application.) For the SS cache, however, where the number of documents is much larger, we maintain the ranking merely on the basis of the more stable near-term temperature metric. This holds for the stage-one tentative identification of prefetching candidates. Later, in stage two, when the prefetching request is about to be served and its weight is compared against the weights of the selected replacement victims, we apply the more "accurate" comparison based on the full weight metric. The candidate document is prefetched only if its weight exceeds the maximum weight among the replacement victims. This way we avoid having to maintain a complete ranking of cached documents by weight.

What remains to be done at this point is the actual derivation of the replacement cost of a given document. We need to distinguish three cases.

1. *Document copy available from SS.* If the document is dropped from the cache, it can be retrieved from SS. (This is relevant only when we consider the PS cache.) In this case, the cost for retrieving the document back again is the mean response time of the SS conditioned by the probability that it will be accessed at least once in the lookahead time window. This replacement cost is amortized over $N_{\text{spec}}(d,T)$ expected accesses, thus yielding the following per-access cost:

$$RC_{\text{SS}}(d) = \Pi_{\text{spec}}(d,T) * RT_{\text{SS}}(d)/N_{\text{spec}}(d,T)\ , \qquad (14)$$

   where $RT_{\text{SS}}(d)$ denotes the empirically estimated mean response time of the SS for a document of size $S(d)$ and $\Pi_{\text{spec}}(d,T)$ is the probability that d will be accessed one or more times within time T.
2. *Document copy available from TS.* If the dropped document needs to be retrieved from TS, we need to further distinguish the online versus the offline case.
   a. *Offline TS.* If the document is currently not online at all, we assume that this situation lasts for the entire lookahead time window. Then the cost of retrieving it back again is the mean response time to an offline document. This yields

$$RC_{TS-\text{off}}(d) = RT_{TS-\text{off}}(d)\ . \qquad (15)$$

   b. *Online TS.* If the document is currently still online, we need to determine the *remaining online time* of the corresponding volume, as the volume may become offline at some point in the lookahead time window. Obviously, the remaining online time is continuously changing as time progresses. To avoid continuous recomputing, we rather estimate the *mean online time*, $T_{\text{on}}(v)$, of a volume v, which is derived as follows. Suppose that the time between two successive volume exchanges is T_{ex}. Under the assumption that the TS robot arm is permanently busy exchanging volumes, T_{ex} can be viewed as a constant and can be easily measured. Further assume, for tractability, that, at each volume exchange, the robot arm makes a random choice among the currently online volumes to determine an "exchange victim". Then, with L drives, the number of volume exchanges that a volume "survives" (i.e., remains online) is geometrically distributed: the probability of surviving i exchanges is $\left(\frac{L-1}{L}\right)^i * \frac{1}{L}$, and the expectation value for the number of survived exchanges is $L-1$. So, the mean online time of a volume is composed of $L-1$ time periods of length T_{ex} plus the remaining time of the current T_{ex} period, which we estimate to be $T_{\text{ex}}/2$. This holds for volumes that are not yet scheduled to be ejected. For those that are already known to be ejected by the next volume exchange (i.e., the robot arm is already moving to the corresponding drive), we set the remaining online time to zero. Putting everything together yields the following formula for the online time of volume v:

$$T_{\text{on}}(v) = \begin{cases} T_{\text{ex}}/2 + (L-1) * T_{\text{ex}} \\ \quad \text{if } v \text{ is not scheduled for ejection} \\ 0 \quad \text{if } v \text{ is going to be ejected next} \end{cases} \qquad (16)$$

   Finally, the replacement cost of an online document d can be computed as the conditioned sum of two components. (1) For the remaining online time $T_{\text{on}}(v)$, the replacement cost is given by the mean response time of the online TS conditioned by the probability that d will indeed be accessed at least once within this online time. But this cost would have to be paid only once for all $N_{\text{spec}}(d,T)$ expected requests, so we need to normalize it on a per-request basis. (2) For the fraction of the lookahead time window in which the volume is expected to be offline, each of the expected requests will have to pay the cost of the mean offline response time, but only with the probability that d is not requested at least once already in its remaining online period. This yields the formula

$$\begin{aligned} RC_{TS-\text{on}}(d) = {} & \Pi_{\text{spec}}\left(d, T_{\text{on}}(v)\right) \\ & * RT_{TS-\text{on}}(d)/N_{\text{spec}}(d,T) \\ & + \left(1 - \Pi_{\text{spec}}\left(d, T_{\text{on}}(v)\right)\right) \\ & * RT_{TS-\text{off}}(d)\ . \end{aligned} \qquad (17)$$

5.3 Prefetch request scheduling for online volumes of the tertiary storage

We generally assume that the TS scheduler aims to maximize the I/O throughput of the TS by batching requests into a *Scan* (also known as Elevator or Sweep) service policy for each volume. The scheduler maintains two queues for each volume, a queue for the actually *pending requests* that have been explicitly issued by clients, and a queue for the speculative *prefetching requests* for which prefetching is considered worthwhile. When the Scan policy begins a forward or backward sweep over the tracks of an online volume, it merges the current contents of these two queues into a single request queue ordered by track number. The scheduler keeps repeating such volume sweeps until it decides to exchange the volume, i.e., eject it from the drive and load another, previously offline volume into the drive. The policy for these volume exchanges is intentionally separated from the migration policy for modularity. It could be a simple time-slice-based round-robin policy, or a more advanced policy that considers the near-term heat of documents (where, of course, we favor the latter one, as we will discuss in Sect. 5.5). In any case, invoking a large number of speculative prefetching requests on an online volume may result in delaying the requests (both pending and prefetching requests) for another volume that is not yet online. This penalty should be recognized and quantitatively assessed by the migration manager in order to guarantee appropriate throttling of the prefetching activity.

To avoid the above form of contention at the online TS, we introduce two decision steps that allow us to cancel a prefetch request immediately before the TS would start serving it.

1. We compare the prefetching candidate and the PS replacement victims in terms of the *weight* metrics, the incentive being that we want to base this assessment on most up-to-date near-term heat statistics.
2. We compute the *benefit* of the document to be prefetched that we would obtain, in terms of expected response-time gains, from having the document available at a higher level of the storage hierarchy. We also compute the *penalty* that the prefetching would incur on other, pending requests to documents of the same online TS volume, as these pending requests would be delayed, thus yielding an adverse impact on the overall mean response time. Then, these two metrics are compared (note that they have the same dimension: (expected) number of requests times response time improvement or degradation), and we decide to cancel the prefetching request if its penalty exceeds its benefit.

The benefit of prefetching document d from an online volume v is estimated as follows. Prefetching d saves us the first fetch request to the document in that it masks the latency of the online TS. Further requests that arrive during the remaining online time of v would benefit equally from either the prefetching or the first explicit fetch request. However, this holds only if the first fetch request will be issued while the volume is still online, or in probabilistic terms, with the probability that there is at least one fetch request to d during the remaining online time of v. The latter probability is exactly the visiting probability, $\mathit{\Pi}_{\text{spec}}(d, T_{\text{on}})$ analyzed in Sect. 4, where the time horizon is set to the mean online time of a volume, T_{on}, that we derived in Sect. 5.2. Thus, like in Sect. 5.2, we replace the remaining online time of a volume with the overall mean online time for tractability reasons. The time delay that we save by the prefetching of d is the response time of later having to fetch d from the online TS, that is, the mean response time for a document of size $S(d)$, $RT_{TS-\text{on}}(d)$.

Once the online time of volume v is passed, we need to assume that the volume may go offline immediately. Thus, if we had prefetched document d before this point, we could now potentially save the delays of a request to offline TS. The number of requests that would benefit from this prefetching equals the expectation value $N_{\text{spec}}(d, T)$ derived in Sect. 4. However, this term is relevant only with the probability that d is not yet explicitly requested during the online time of v. So, we need to condition it with the factor $1 - \mathit{\Pi}_{\text{spec}}(d, T_{\text{on}})$. Altogether, this yields the following formula for the benefit of prefetching d from v:

$$\begin{aligned}\text{benefit}(d) = {} & \mathit{\Pi}_{\text{spec}}\left(d, T_{\text{on}}(v)\right) * RT_{TS-\text{on}}(d) \\ & + \left(1 - \mathit{\Pi}_{\text{spec}}\left(d, T_{\text{on}}(v)\right)\right) * N_{\text{spec}}(d\ , T) \\ & * RT_{TS-\text{off}}(d)\ . \end{aligned} \tag{18}$$

In evaluating this formula, we can re-use bookkeeping information and computations that we already identified as major building blocks in Sect. 5.2.

For estimating the penalty that the prefetching request incurs on other outstanding requests, we should first realize that the prefetching request for d delays all already *pending requests* to other documents on the same volume v. The length of the delay is given by the service time, $ST_{TS-\text{on}}(d)$, for reading d from the online TS. The above penalties all refer to requests to the volume that holds d and is currently online. In addition, however, a second category of penalties may arise from the fact that the prefetching request potentially delays the ejection of the volume, namely, by delaying the pending requests which must be served in the remaining online time. Therefore, all pending requests to currently offline volumes may be delayed as well. However, this holds only if the volume v is already scheduled for being ejected next, so that the prefetching request would actually extend its online time. These considerations lead to the following estimation of the penalty that arises at the TS:

$$\text{penalty}_{TS}(d) = \begin{cases} \sum_{d' \in \text{docs}(v) \wedge d' \neq d} N_{\text{pend}}(d') * ST_{TS-\text{on}}(d) \\ \qquad \text{if } v \text{ is not scheduled for ejection,} \\ \sum_{d' \in \text{docs}(v) \cup \text{docs}(TS-\text{off}) \wedge d' \neq d} N_{\text{pend}}(d') * ST_{TS-\text{on}}(d) \\ \qquad \text{if } v \text{ is going to be ejected next,} \end{cases} \tag{19}$$

where $N_{\text{pend}}(d')$ is the number of currently pending, explicit client requests to document d'.

5.4 Write request scheduling for the secondary storage cache

Similarly to the possible contention at the TS, the SS may also become a bottleneck in the overall performance. Here,

the potential point of contention is that read requests that fetch cached documents from SS into PS for delivery to the client may be delayed by write requests that "spool" prefetched or fetched documents from the PS cache onto SS for further caching. This spooling is the natural data transfer path for documents, but it still needs to be dynamically reconsidered when the disk utilization at the SS level becomes critical. In fact, it may occur that a document is prefetched on behalf of a single session to mask the high TS latency, but, by the time it is about to be written to the SS cache, its near-term heat has dropped sharply. In these cases, the write requests should be cancelled. Making this decision involves the same kind of benefit/penalty assessment that we introduced above for the scheduling of prefetching requests at the TS level. The only difference is in the details of the underlying quantitative formulas.

As for the benefit of a document, the assessment is, in fact, still identical to the one for the TS, as derived in Sect. 5.3. The benefit captures the expected response-time improvement that results from caching the document on disk instead of accessing it on tertiary storage. The penalty, however, is different from what we considered in Sect. 5.3. The writing of a PS-resident document onto SS delays all pending requests to the SS, and the length of this delay is the disk service time for writing a document of size $S(d)$, denoted by $ST_{\text{SS}}(d)$. This yields the following formula for the penalty at the SS level:

$$\text{penalty}_{\text{SS}}(d) = \sum_{d' \in \text{docs}(SS)} N_{\text{pend}}(d') * ST_{\text{SS}}(d) \; . \qquad (20)$$

The disk scheduler always considers those documents for writing to SS that are likely to be removed from PS next. Such document droppings from PS are caused by fetch requests from TS or SS or prefetching from TS into PS, replacing the documents in PS with the lowest weight. To save these replacement-"endangered" documents onto the SS cache, the SS scheduler maintains a list of the currently PS-resident documents in ascending weight order (i.e., starting with the document with the lowest weight) and determines a set of spooling candidates according to the following criteria.

- When no fetch request is pending at the SS, all PS-resident documents are considered "endangered" that could be replaced by the fetch or prefetch requests for the TS drives. With S denoting the average document size and L the number of TS drives, the overall PS cache size that may have to be freed up is $L * S$.
- In the case of fetch requests pending at the SS, the PS space that needs to be freed up for the first SS read request must be considered as "endangered" in addition to the space for the TS reads (i.e., the first case above). So, altogether, this requires spooling as many documents to SS as are needed to free up $L * S + S(d)$ space units in PS, where $S(d)$ is the size of the first document d that is to be read from SS.

In both cases, the SS scheduler considers spooling PS-resident documents in ascending weight order until their total size exceeds the required amount of PS space, and initiates write requests for them in that order. Also, in both cases, these write requests are assessed against the pending SS read requests by performing the benefit/penalty comparison for each spooling candidate (one at a time). Note that, in the first case above, the first write request has zero penalty because of the currently empty disk queue. Also note that the spooling candidates may still end up becoming overwritten in the PS cache without first being saved onto SS, namely, when their benefit is smaller than the penalty with regard to the pending SS read requests.

5.5 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 requests and speculative prefetching 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 prefetching 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, denoted $N_{\text{pend}}(v)$. 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 $N_{\text{pend}}(v) * T_{\text{wait}}(v)$, where $T_{\text{wait}}(v)$ is the longest waiting time among all pending requests for volume v. Once a volume is loaded into a drive, all pending requests and prefetching requests are combined and reordered for being served by a Scan-like sweep over the volume. It is at this point when the benefit and penalty of serving a prefetching request are re-assessed and prefetching requests may be cancelled (see Sect. 5.3 above). Once a volume becomes subject of ejection, a final sweep is performed for this volume, where the weights of the documents to prefetch as well as the benefit and penalty are re-adjusted (see Sects. 5.2 and 5.3 above). Requests that arrive during the final sweep and whose position on the volume has already been passed are held back in their queue until the next time when the volume is loaded.

Whenever a drive is unused with regard to the explicitly issued pending requests, the scheduling policy loads a volume solely for serving speculative prefetching 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 6) for all prefetching 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 Sect. 4) are not only useful for initiating prefetching requests, they also serve as a heuristics for scheduling volume exchanges.

5.6 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 low 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 prefetching is too aggressive, the prefetched documents may cause the replacement of documents that turn out to be (re-)used earlier than the prefetched ones.

Fortunately, there is a simple rationale for setting the lookahead time horizon. Consider the average time period between the successive ejection and loading of the same volume; this metric can be measured online and will be denoted as T_{load}. The final sweep after a volume becomes subject of ejection is the latest point in time where it is possible for a document to be prefetched. Once the volume is ejected, all requests for a non-cached document result in pending requests for offline volumes. In order to avoid these cache "misses", at least all speculative requests up to the time when the volume is online again, which is T_{load} time units later, have to be considered. So, T_{load} is a lower bound for the lookahead time horizon. On the other hand, all speculative requests arriving after time T_{load} should not be considered at all, as, for those requests, it would be sufficient to prefetch the document when the volume becomes loaded the next time. For those speculative requests after time T_{load}, prefetching the document before the volume becomes loaded again would be a waste of cache space. This consideration shows us that T_{load} is also an upper bound for the lookahead time of the access predictor. So, T_{load} is indeed a reasonable canonical choice for the lookahead time.

6 Implementation of the bookkeeping

In this section we discuss the implementation and the overhead of the bookkeeping for the access predictions, the prefetching 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_i, the expected numbers of near-future accesses to other documents change according to Eq. 5. At this point, the state-transition graph of the Markov chain is traversed, starting from d_i and always proceeding along the highest probability transition of the state with the currently highest probability of being reached from d_i. 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 Eq. 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 re-uses 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 re-used 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 prefetching requests and deciding on cache replacements are driven by the following sorted lists.

- l_{Doc}: this list keeps the document IDs, the document sizes, and the weights of all documents which have non-zero weights. It is used for determining the tentative prefetching candidates (stage 1 in Sect. 5.1) by traversing the list in weight-descending order until the aggregated size of the inspected documents exceeds the total space capacity of the SS cache. For each inspected document which is not yet cached on SS or PS, a prefetching request is generated.
- l_{PS}: this list keeps the document IDs, the document sizes, and the weights of all documents which are cached in PS.
- l_{SS}: this list keeps the document IDs, the document sizes, and the weights of all documents which are cached on SS.

All lists are sorted according to the document weights. The sorting has to be adjusted each time a session changes its state. In the case of the rather small l_{PS} list, the sorting is additionally adjusted with each exchange of volumes (as discussed in Sect. 5.2). The two lists l_{PS} and l_{SS} are used for determining the replacement victims in PS and SS, respectively. The sizes of these lists obviously depend on the number of documents residing on the corresponding storage level and should incur only negligible space overhead. The size of l_{Doc}, 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).

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

6.1 Recovery of bookkeeping data

Keeping the bookkeeping data in main memory leads to the loss of this data upon a server crash. As the bookkeeping data solely serves to control the caching of documents, the loss only results in future performance degradations, but does not affect the principal functionality of the document archive.

However, especially for infrequently accessed documents, the reconstruction of lost CTMC bookkeeping data (i.e., mean state residence times and state transition probabilities) may take a very long time, resulting in performance degradations over an extended time period. For this reason, the CTMC bookkeeping data should be kept also on stable storage (i.e., secondary storage).

A background process periodically copies updated CTMC bookkeeping data from main memory to stable storage, giving preference to the data for those documents with a high number of accesses since the last saving of the CTMC bookkeeping data. Such frequently accessed documents are the only ones where the statistics may have changed significantly and where the loss of the statistical data would have the highest impact. After a server crash, the CTMC bookkeeping data can be loaded again from stable storage into main memory.

6.2 Caching of bookkeeping data

All statistics on mean state residence times and state transition probabilities of the CTMC are kept in main memory. For document archives with millions of documents, however, this may not be possible in all cases. Although the storage requirement of the bookkeeping is very low compared to the document data itself, the bookkeeping overhead of a multi-terabyte document archive may reach tens or even hundreds of megabytes, which may exceed the amount of main memory that is available for this purpose. Therefore, the server may have to employ a cache replacement strategy for the CTMC bookkeeping data.

The objective of the bookkeeping-cache replacement is to keep those CTMC bookkeeping data in main memory that will be frequently used by the algorithm for computing the expected numbers of near-future accesses to documents. As this algorithm traverses the statistics starting from the current document of a session and navigating along the document transition probabilities, the frequently used statistics are the statistics for those documents that have a high expectation of near-future accesses. Thus, the estimation of near-future accesses can itself guide the caching of CTMC bookkeeping data. The weight of the CTMC bookkeeping data for a document is defined analogously to the document weight, considering near-future accesses to the document and the size of the bookkeeping data. The CTMC bookkeeping data with minimum weight will be chosen for replacement from main memory.

7 Prototype architecture

In this section, we describe the architecture and some implementation details of our document archive prototype. Figure 4 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 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 Session Manager is implemented as a thread receiving document requests from the network, signalling them to the underlying Migration Manager, and receiving signals from the Migration Manager upon the completion of document transfers to the client.

The Migration Manager organizes the migration of documents between tertiary storage, secondary storage and primary storage serving both prefetching requests and pending requests. It consists of the Loader which is responsible for the scheduling of document migrations from TS into PS, as well as from SS into PS and from PS to SS. The Loader is also responsible for controlling the exchange of volumes. For each of the devices, the Migration Manager contains a thread that implements the scheduling. In addition, the Loader keeps track of some online statistics needed by the Weight Watcher for estimating replacement costs and outweighting benefit and penalty of document migrations.

The most important submodule of the Migration Manager is the Weight Watcher, which implements the access predictions and manages the lists sorted by document weights. Furthermore, the Weight Watcher keeps information about volumes, for example, the volume status (i.e., online vs offline) and the number of speculative requests to the volume. Each new request of a session, as well as session arrivals and departures, are signalled to the Weight Watcher, which then updates the volume information and document weights as described in Sect. 5. The Weight Watcher is also responsible for the generation of prefetching requests. When the Loader is about to prefetch a document, the Weight Watcher is asked again if the migration should still be performed or should be cancelled, taking the potential replacements and benefit/penalty of the migration into account (stages 2 and 3 in Sect. 5.1). The Weight Watcher is informed about all performed migrations, as well as volume status changes. For the MEAT volume-exchange scheduling, the Weight Watcher is consulted about the next volume exchange to perform. Concurrent update and retrieval of the bookkeeping data provided by the Weight Watcher is implemented using standard primitives for thread synchronization.

As a base layer, 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 the two caching levels and the 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 re-

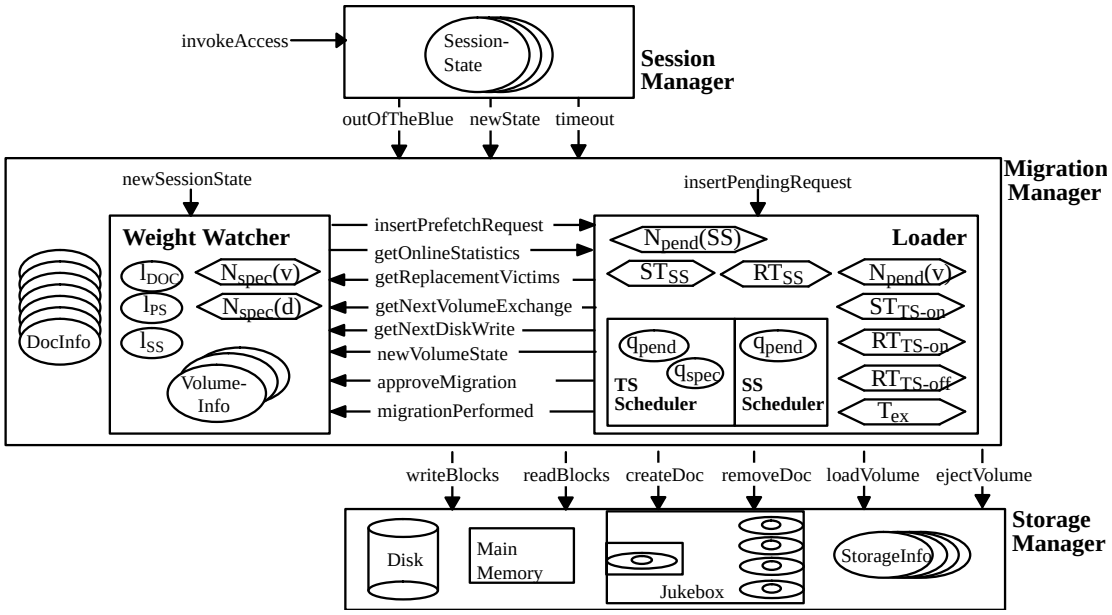

Fig. 4. Overview of the prototype system

producibility and statistical confidence of the performance results.

8 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 different variants of the Markov-chain based migration policy against purely temperature-based migration policies. All policies use the MEAT policy for the scheduling of volume exchanges.

McMin: Markov-chain based Migration policies for near-line storage

The McMin policies are all based on the integrated vertical migration policy that we have developed in this paper. The number of speculative requests are derived using the CTMC model. We have investigated three different variants of McMin.

- *McMin.* With this basic McMin variant, the document weight considers only the near-term temperature. That is, the replacement costs in formula 13 are set to unit costs: $RC(d) = 1$ for all d regardless of the storage level from which d would have to be retrieved. Furthermore, the basic McMin policy eagerly prefetches documents based only on the weight comparison against the replacement victims. So it does not perform the benefit/penalty comparison and thus does not take into account the possible contention at the TS and SS devices (stages 2 and 3 in Sect. 5.1).
- *McMin+.* The McMin+ policy extends the basic McMin policy by considering document-specific replacement costs in the weight metric and by adding the benefit/penalty comparisons. So this full-fledged variant of McMin includes all the considerations of Sect. 5.
- *McMin−.* The McMin−policy is the basic McMin variant with prefetching switched off. So this simpler and presumably strictly weaker variant corresponds to a pure cache replacement policy that uses the CTMC-based access predictions for an intelligent choice of replacement victims.

TEMP: Temperature-based migration policies

In the temperature-based migration policies, the weight of a document is its stationary, i.e., long-term temperature (i.e., long-term heat/size). The long-term 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 caching algorithm [OOW93, WHMZ94] to the case of variable-size buffering granules. We have considered three different variants of TEMP.

- *TEMP.* This base variant is the counterpart to the basic McMin algorithm, with the decisive difference that TEMP initiates prefetching and selects replacement victims based on the documents' long-term temperature as opposed to the near-term estimates of McMin. Like McMin, no document-specific replacement costs are considered in TEMP and no benefit/penalty comparisons are performed.
- *TEMP+.* This variant extends TEMP by considering replacement costs and by comparing the benefit against the penalty of the prefetching requests. So this is the stationary counterpart to McMin+.
- *TEMP−.* The TEMP−variant is a basic TEMP policy where prefetching is switched off.

Table 1. System parameters of the simulation testbed

Secondary storage		Tertiary storage	
Parameter	Value	Parameter	Value
Number of disks	1	Number of drives	3
Average seek time	6 ms	Average seek time	25 ms
Average rotational latency	3 ms	Average rotational latency	10 ms
Transfer rate	13.5 MB/s	Transfer rate	3.4 MB/s
Controller cache size	1 MB	Controller cache size	1 MB
		Number of volumes	10
		Volume exchange time	10 s

8.1 Experimental testbed

The experiments have been carried out on the prototype system described in Sect. 7 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 high-end SCSI disks and low-end magneto-optical jukeboxes. The ratio of TS drives to volumes has been chosen rather high (3:10) to better capture the influence of the TS volume exchange scheduling. Experiments with other settings have been carried out, too, but essentially show the same effects and trends as reported below. (Results for the special case with one TS drive were already reported in [KW97], which focused exclusively on that case).

We have considered an archive with 40,000 documents; document sizes are exponentially distributed with mean 600 KB. Thus, the total archive size is approximately 23 GB. 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, 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% 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 Sect. 4.2) 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% of the probability mass among the outgoing transitions is covered by the six 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.

8.2 Experimental results

We concentrate on the four policies, McMin−, McMin, McMin+, and TEMP+, and omit the results for TEMP and TEMP−, as they are consistently worse than those of the other four variants. The response time results of the remaining four policies are depicted in Fig. 5 for each of the four different workloads. The charts show the mean response time of client requests with different sizes of available primary and secondary storage cache space. Note that the disk cache sizes of 100 MB and 500 MB correspond to only 0.4 and 2% of the total archive size, respectively. The used cache sizes may appear very small, but they are of reasonable size relative to the archive size, which is also rather small for the sake of fast-running experiments (on our limited computer resources). Note that these results can be scaled up and extrapolated in a straightforward manner by increasing both the archive and the cache size by a constant factor, say

Table 2. Workload parameters for the simulation experiments

Parameter	LOW_SLOW	LOW_FAST	HIGH_SLOW	HIGH_FAST
Mean session interarrival time	150 s	150 s	50 s	50 s
Mean session length	24 requests			
Standard deviation of session length	12 requests			
Minimum mean residence time	30 s	10 s	30 s	10 s
Maximum mean residence time	180 s	60 s	180 s	60 s

Table 3. Hit rates and mean response times at the storage levels for the HIGH_SLOW workload

Disk size	Memory size	Policy	overall RT [s]	HR_{PS} [%]	HR_{SS}[%]	RT_{SS} [s]	$HR_{TS-\text{on}}$ [%]	$RT_{TS-\text{on}}$ [s]	$HR_{TS-\text{off}}$ [%]	$RT_{TS-\text{off}}$ [s]
500 MB	8 MB	McMin−	21.64	7.3	10.5	0.054	20.7	0.243	61.5	35.13
		TEMP+	18.33	4.0	30.4	0.066	14.1	0.277	51.5	35.47
		McMin	9.31	14.6	47.0	0.690	7.0	0.903	31.4	28.43
		McMin+	8.47	9.9	55.7	1.142	6.0	1.207	28.4	27.32
	40 MB	McMin−	21.86	11.6	8.4	0.039	19.5	0.241	60.5	36.04
		TEMP+	18.41	10.5	23.5	0.122	14.0	0.275	52.0	35.25
		McMin	8.24	31.1	34.8	0.773	5.7	1.25	28.4	27.86
		McMin+	6.36	28.9	42.9	0.776	4.2	1.36	24.0	24.92

1000, provided that the skewness in the access patterns remains invariant in terms of fractions of frequently followed transitions and that the number of drives in the jukebox is increased, too.

For more detailed information, Table 4 shows the hit rates (HR) and mean response times (RT) of each storage level for the HIGH_SLOW workload with 500 MB disk cache and the two different memory cache sizes. For the PS, the response time figures are omitted, as they are close to zero and indeed negligible relative to the response times for the other storage levels.

Overall, it is evident that the newly developed McMin and McMin+ policies with prefetching consistently outperform the temperature-based TEMP+ policy that includes prefetching, too. The basic McMin already improves the mean response time by a factor between 2 and 4. The highest gains are achieved for the LOW_SLOW workload, as this has the lowest interarrival time of explicit client requests and thus leaves more "idle" time to prefetch documents from tertiary storage. The detailed results in Table 3 show that for the HIGH_SLOW workload with 500 MB disk cache and 40 MB memory cache, the "miss" rate, i.e., fraction of requests that refer to offline tertiary storage ($HR_{TS-\text{off}}$), is 52% for TEMP+ and only 24% for McMin. So, McMin significantly increases the hit rates HR_{PS} and HR_{SS} of documents in PS and SS. TEMP+, on the other hand, is inherently limited by its less informative knowledge of merely stationary access probabilities. Thus, the TEMP+ policy fails to prefetch relevant documents in many cases, namely, those that are 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.

The McMin+ policy almost always further reduces the mean response time of the basic McMin policy by up to 20%. Especially with large sizes of disk and memory cache, the benefit/penalty assessment for tentative prefetching requests leads to further improvements. The reason is that the prefetching activity increases with increasing disk and memory cache sizes, potentially leading to the prefetching of documents with a low expected number of speculative requests at the cost of delaying pending requests. Experiments reported in [KW97] have shown that comparing the influence of the benefit/penalty comparison is even more significant in the case with only one TS drive, as in this case a prefetching could delay all pending requests of the entire server.

In the case of a large disk cache size and a small memory cache size (i.e., the 500 MB/8 MB combination in Table 3), the PS becomes the bottleneck in the prefetching, and, considering the replacement costs of documents, becomes very beneficial here. The hit rate at the PS is higher for McMin than for McMin+. The reason is that McMin inherently maximizes the PS hit rate, while McMin+ considers the replacement costs of documents. So, McMin+ prefers to replace documents from PS which are cached on SS or which could be loaded from an online TS volume. With this replacement policy, the throughput of prefetching requests is increased, as can be seen from the increased SS hit rate (HR_{SS}) in Table 3. With both of the two investigated PS sizes, McMin+ significantly reduces the hit rates of requests to online and offline volumes, $HR_{TS-\text{on}}$ and $HR_{TS-\text{off}}$, respectively. The drawback of the higher prefetching activity is the high response time for fetch requests that are served from SS or online TS volumes. Especially with a small PS cache, documents with high weight frequently have to be replaced from PS, making the spooling to disk more important. In these situations, comparing the benefit and penalty for prefetching a document from TS and later spooling it onto SS yields a significant response time improvement. Note that the fairly high SS response times are caused by the very high disk utilization due to many SS hits, the large document sizes, and the intensive spooling activity from PS onto SS. This high disk utilization is indeed desirable, as the SS serves as a cache to mask the much larger latency of the offline TS to the best possible extent.

A comparison of the McMin− policy (i.e., with prefetching switched off) to the TEMP+ policy shows that with a large SS cache, TEMP+ benefits from temperature-based prefetching and is able to outperform McMin− (while still

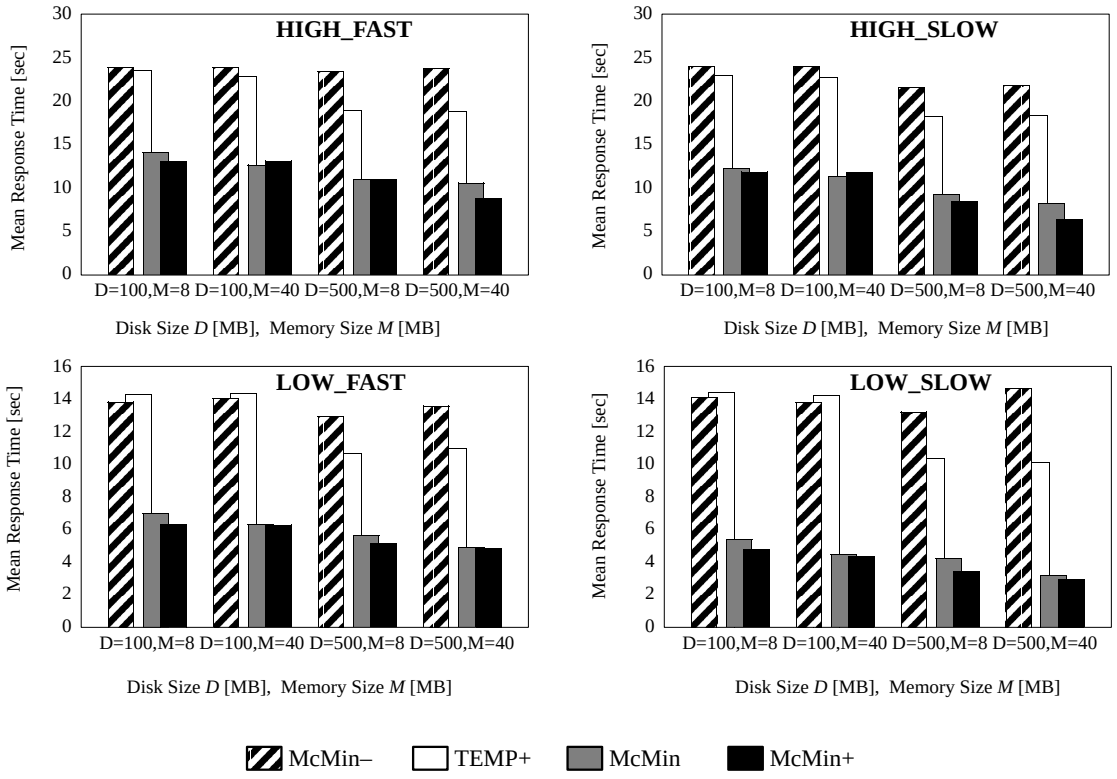

Fig. 5. Response time results for the four workloads with different disk and memory cache sizes

losing against McMin and McMin+). With a small disk cache size, however, the SS hit rate of TEMP+ decreases and McMin− benefits from the MEAT scheduling algorithm. When MEAT is based on the Markov-chain predictions rather than stationary heat as used in conjunction with TEMP+, the McMin− policy achieves a significantly better hit rate on online TS volumes.

Additional experiments with variations of the workload parameters essentially confirmed these findings and are omitted here for brevity. We also investigated the bookkeeping overhead of the prototype implementation. The total space overhead of all bookkeeping data of the McMin policies was about 10 MB (for the 23 GB document archive). The CPU consumption for the predictions per session step, including the Markov-chain computations and the adjustment of the various sorted lists, was about 200 ms on average on a Sparc20 (i.e., a low-end workstation). On a faster commodity server, this would translate into a CPU time in the order of the secondary storage access time. 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 moderate number of recently computed predictions in a special lookup buffer.

9 Extensions and generalizations

In this paper, we have applied access predictions based on continuous-time Markov chains (CTMC) to the problem of prefetching documents in a storage hierarchy with near-line storage. In addition, the CTMC model provides an interesting and potentially promising approach to other optimization issues and also other application areas where the load consists of interactive access patterns embedded in client sessions.

- *Document clustering*
  In our experiments, the documents of the archive were randomly distributed over the volumes of the TS jukebox. This random distribution typically leads to an increased number of pending requests to offline volumes. Using a more intelligent clustering of documents on volumes (see, e.g., [CTZ97]) would increase the number of pending requests to online volumes and render the prefetching of the McMin policies even more effective. For clustering documents onto volumes, one could exploit the access predictions from the CTMC model. Consider a scenario where multiple sessions have pending requests to a volume which is being loaded into a drive. Ideally, most documents with a high near-term heat within these sessions should reside on this same volume. Such clustering approaches based on Markov chain predictions have been studied by [TN91, TN92] in the context of object-oriented database systems (i.e., clustering objects into pages), but were limited to discrete-time Markov chains. For interactive accesses, however, considering document-specific residence times and thus using CTMCs seems to be a more promising approach.
- *Incremental reorganization*
  Document archives such as news archives, electronic museums, etc. are usually dynamic in that they are continuously extended by new documents. In addition, the contents of documents may change leading to evolving access patterns. Once a new document is created or updated, the document can be held for some time on secondary stor-

age for being "watched", that is, for collecting the CTMC bookkeeping information after each access. Note that new documents and updated documents are typically popular documents with frequent accesses, so that collecting the CTMC bookkeeping information for those documents should not take very much time. Once the CTMC bookkeeping data of an inserted or updated document has sufficient significance, the document may be migrated down to tertiary storage, according to the access patterns. In addition, it may be desirable that other documents be reclustered due to their evolving access patterns. This reclustering has to be done incrementally and concurrently to the normal operation of the server. The reclustering of documents can be based on the same principles as the static clustering of documents (i.e., using CTMC information).

- *Data declustering*
  In parallel database systems, declustering data objects across multiple devices is a standard performance-enhancing technique. In the presence of evolving workload patterns, horizontal data migration between devices of the same storage level may be needed, for example, for dynamic and incremental load balancing among disks. As Markov-chain models are substantially richer than the stationary-probability models that have been explored for these purposes [SWZ98], further performance improvements may be possible.
- *Prefetching and caching for Web servers*
  Within the Internet and large intranets, proxy servers are often used for the caching of documents that are likely to be accessed by the local clients (see, e.g., [Be96]). One reason for the caching of documents is to provide better performance by delivering a requested document from the (predictably) fast proxy server rather than depending on the often poor network performance and connectivity of the documents' sources. Another reason could be to mask the possible unavailability of document sources, caused by computer or network failures or intentional computer downtime. In large intranets, business units typically use servers with different availability characteristics. Then, a proxy-server cache may be used also for caching documents whose home servers are expected to be(come) unavailable within the near future. For example, when a server announces a prescheduled downtime, proxy servers may start prefetching (or "hoarding") documents from that server in order to mask the announced offline time. Obviously, an intelligent choice should be made as to which documents are most worthy of being prefetched. The CTMC-based migration strategy developed in this paper can be applied to this problem in a fairly straightforward manner, as the notion of a server going offline is similar to a TS volume becoming offline.
- *Data hoarding in mobile systems*
  Another, similar example where CTMC can be used for maximizing data access availability and performance is the problem of caching and prefetching documents in mobile computing (see, e.g., [KP97]). Here, CTMC predictions could be used to intelligently decide on which files should be "hoarded" on a mobile computer that is about to become disconnected.
- *Data dissemination in broadcast architectures*
  Because of the often asymmetric network bandwidth in sending versus receiving data between mobile clients and a server, the server may broadcast information that is likely to be useful for its mobile clients while they depend on wireless (or otherwise asymmetric) communication [AAFZ95, AFZ96]. So, broadcasting serves to minimize explicit data requests of mobile clients to the server. The idea is to broadcast "hot" data items with high frequency over the network, while colder data is sent with lower frequency. It would be interesting to investigate CTMC predictions for adapting these broadcasting frequencies to the dynamically evolving "session states" of mobile clients.

10 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 and secondary 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.

Our experimental studies have shown that the richer stochastic knowledge of an Markov-chain model can substantially outperform a simpler model that is solely based on stationary probabilities. Thus, it seems intriguing to apply the richer workload-tracking approach also to other issues in storage systems, such as distributed caching, to exploit the aggregate memory of NOWs (i.e., networks of workstations) [DWAP94, SW97, VLN97]. However, one has to be careful with regard to the overhead of an Markov-chain model. When the response gains from a richer decision model are in the order of seconds, as is the case with tertiary storage libraries, a computational overhead in the order of milliseconds and a space overhead in the order of megabytes is clearly worthwhile. On the other hand, for tuning the performance of disk accesses or remote memory accesses, both in the order of a few milliseconds, much faster decisions are needed. This rules out the same sort of full-fledged Markov-chain model that we have successfully employed for storage hierarchies with very high latency levels in this paper. But weaker, more lightweight forms of such models are conceivable, too. For example, one could make more intensive use of caching precomputed estimations (i.e., caching at the meta-level) or combine Markov-chain predictions for specifically interesting "data regions" with simpler stationary-probability access predictions for the majority of data. Encouraged by the excellent experimental results of this paper, we believe

that this research direction towards intelligent, self-tuning storage management is worthwhile to be explored further.

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

References

[AAFZ95] Acharya S, Alonso R, Franklin M, Zdonik S (1995) Broadcast Disks: Data Management for Asymmetric Communication Environments. In: ACM SIGMOD Conf., 1995, pp 199–210

[ABCO96] Almeida V, Bestavros A, Crovella M, De Oliveira A (1996) Characterizing Reference Locality in the WWW. In: Int. Conf. on Parallel and Distributed Information Systems (PDIS), 1996, Miami Beach, Fla., pp 92–103

[AFZ96] Acharya S, Franklin M, Zdonik S (1996) Prefetching from Broadcast Disks. In: Int. Conf. on Data Engineering, 1996, New Orleans, La., pp 276–285

[AGL97] Albers S, Garg N, Leonardi S (1997) Minimizing Stall Time in Single and Parallel Disk Systems. Technical Report MPI-I-97-024. Max-Planck Institute for Computer Science, Saarbrücken, Germany

[All90] Allen AO (1990) Probability, Statistics, and Queueing Theory with Computer Science Applications. Academic Press, London

[Be96] Bestavros A (1996) Speculative Data Dissemination and Service in Distributed Information Systems. In: Int. Conf. on Data Engineering, 1996, New Orleans, La., pp 180–187

[CFKL95a] Cao P, Felten EW, Karlin AR, Li K (1995) A Study of Integrated Prefetching and Caching Strategies. In: ACM SIGMETRICS Conf., 1995, pp 188–197

[CFKL95b] Cao P, Felten EW, Karlin AR, Li K (1995) Implementation and Performance of Integrated Application-Controlled Caching, Prefetching and Disk Scheduling. Technical Report TR-CS95-493, Princeton University, Princeton, Calif.

[CH91] Cheng JR, Hurson AR (1991) On The Performance Issues of Object-Based Buffering. Int. Conf. on Parallel and Distributed Information Systems (PDIS), 1991, Miami Beach, Fla., pp 30–37

[CK89] Chang EE, Katz RH (1989) Exploiting Inheritance and Structure Semantics for Effective Clustering and Buffering in an Object-Oriented DBMS. In: ACM SIGMOD Conf., 1989, Portland, Ore., pp 348–357

[CKV93] Curewitz KM, Krishnan P, Vitter JS (1993) Practical Prefetching via Data Compression. In: ACM SIGMOD Conf., 1993, Washington, D.C., pp 257–266

[CR94] Chen LT, Rotem D (1994) Optimizing Storage of Objects on Mass Storage Systems with Robotic Devices. In: Int. Conf. on Extending Database Technology (EDBT), 1994, Cambridge, UK, pp 273–286

[CTZ97] Christodoulakis S, Triantafillou P, Zioga F (1997) Principles of Optimally Placing Data in Tertiary Storage Libraries. In: VLDB Conf., 1997, Athens, Greece, pp 236–245

[Co88] Copeland G, Alexander W, Boughter E, Keller T (1988) Data Placement in Bubba. In: ACM SIGMOD Conf., 1988, Chicago, III, pp 99–108

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

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

[DWAP94] Dahlin MD, Wang RY, Anderson TE, Patterson DA (1994) Cooperative Caching: Using Remote Client Memory to Improve File System Performance. In: 1st Symposium on Operating Systems Design and Implementation, 1994, pp 267–280

[FC91] Ford DA, Christodoulakis S (1991) Optimal Placement of High-Probability Randomly Retrieved Blocks on CLV Optical Disks. ACM Trans Inf Syst 9(1): 1–30

[GK94] Gerlhof CA, Kemper A (1994) Prefetch Support Relations in Object Bases. In: Int. Workshop on Persistent Object Stores (POS), 1994, pp 115–126

[GKKM93] Gerlhof CA, Kemper A, Kilger C, Moerkotte G (1993) Partition-Based Clustering in Object Bases: From Theory to Practice. In: Int. Conf. on Foundations of Data Organization and Algorithms (FODO), 1993, Chicago, III, pp 301–316

[GMW94] Golubchik L, Muntz R, Watson RW (1994) Analysis of Striping Techniques in Robotic Storage Libraries. Technical Report. University of California, Los Angeles, Calif.

[GP87] Gray J, Putzolu F (1987) The 5-Minute Rule for Trading Memory for Disc Accesses and the 10-Byte Rule for Trading Memory for CPU Time. In: ACM SIGMOD Conf., 1987, San Francisco, Calif., pp 395–398

[HS96] Hillyer BK, Silberschatz A (1996) Random I/O Scheduling in Online Tertiary Storage Systems. In: ACM SIGMOD Conf., 1996, Montreal, Canada, pp 195–204

[JK98] Jiang Z, Kleinrock L (1998) An Adpative Network Prefetch Scheme. IEEE J Sel Areas Commun, pp 231–240

[Jo98] Johnson T (1998) Coarse Indices for a Tape-Based Data Warehouse. In: Int. Conf. on Data Engineering, 1998, Orlando, Fla.

[KP97] Kuenning GH, Popek GJ (1997) Automated Hoarding for Mobile Computers. In: ACM Symposium on Operating Systems, October 1997, St. Malo, France, pp 264–275

[KPR92] Karlin AR, Phillips SJ, Raghavan P (1992) Markov Paging. In: Symposium on Foundations of Computer Science, 1992, pp 208–217

[KW97] Kraiss A, Weikum G (1997) Vertical Data Migration in Large Near-Line Document Archives Based on Markov-Chain Predictions. In: VLDB Conf., 1997, Athens, Greece, pp 246–255

[LLW95] Lau SW, Lui JCS, Wong PC (1995) A Cost-effective Near-line Storage Server for Multimedia System. In: Int. Conf. on Data Engineering, 1995, Taipei, Taiwan, pp 449–456

[MKK95] Moser F, Kraiss A, Klas W (1995) L/MRP – A Buffer Management Strategy for Interactive Continuous Data Flows in a Multimedia DBMS. In: VLDB Conf., 1995, Zurich, Switzerland, pp 275–286

[ML97] Myllymaki J, Livny M (1997) Relational Joins for Data on Tertiary Storage. In: Int. Conf. on Data Engineering, 1997, Birmingham, UK, pp 159–168

[Nel95] Nelson R (1995) Probability, Stochastic Processes, and Queueing Theory – The Mathematics of Computer Performance Modeling. Springer, Berlin Heidelberg New York

[NKT97] Nemoto T, Kitsuregawa M, Takagi M (1997) Analysis of Cassette Migration Activities in Scalable Tape Archiver. In: Int. Conf. on Database Systems for Advanced Applications, 1997, Melbourne, Australia, pp 461–470

[OOW93] O'Neil EJ, O'Neil PE, Weikum G (1993) The LRU-K Page Replacement Algorithm For Database Disk Buffering. In: ACM SIGMOD Conf., 1993, Washington, D.C., pp 297–306

[PGG+95] Patterson RH, Gibson GA, Ginting E, Stodolsky D, Zelenka J (1995) Informed Prefetching and Caching. In: Symposium on Operating Systems Principles, 1995, pp 79–95

[PZ91] Palmer M, Zdonik SB (1991) Fido: A Cache that learns to fetch. In: VLDB Conf., 1991, Barcelona, Spain, pp 255–264

[RW94] Ruemmler C, Wilkes J (1994) An Introduction to Disk Modelling. IEEE Comput 27(3): 17–28

[Sa95] Sarawagi S (1995) Query Processing in Tertiary Memory Databases. In: VLDB Conf., 1995, Zurich, Switzerland, pp 585–596

[Smi81] Smith AJ (1981) Long Term File Migration: Development and Evaluation of Algorithms. Commun ACM 24(8): 521–532

[SSV96] Scheuermann P, Shim J, Vingralek R (1996) WATCHMAN: A Data Warehouse Intelligent Cache Manager. In: VLDB Conf., 1996, Bombay, India, pp 51–62

[Sto91] Stonebraker M (1991) Managing Persistent Objects in a Multi-Level Store. In: ACM SIGMOD Conf., 1991, Denver, Colo., pp 2–11

[SW97] Sinnwell M, Weikum G (1997) A Cost-Model-Based Online Method for Distributed Caching. In: Int. Conf. on Data Engineering, 1997, Birmingham, UK, pp 532–541

[SWZ94] Scheuermann P, Weikum G, Zabback P (1994) Disk Cooling in Parallel-Disk Systems. IEEE Data Eng Bull 17(3): 29–40

[SWZ98] Scheuermann P, Weikum G, Zabback P (1998) Data Partitioning and Load Balancing in Parallel-Disk Systems. VLDB J 7(1): 48–60

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

[TG84] Teng JZ, Gumaer RA (1984) Managing IBM Database 2 Buffers to Maximize Performance. IBM Syst J 23(2): 211–218

[Tij94] Tijms HC (1994) Stochastic Models – An Algorithmic Approach. John Wiley & Sons, Chichester

[TN91] Tsangaris MM, Naughton JF (1991) A Stochastic Approach for Clustering in Object Bases. In: ACM SIGMOD Conf., 1991, Denver, Colo., pp 12–21

[TN92] Tsangaris MM, Naughton JF (1992) On the Performance of Object Clustering Techniques. In: ACM SIGMOD Conf., 1992, San Diego, Calif., pp 144–153

[TP97] Triantafillou P, Papadakis T (1997) On-Demand Data Elevation in a Hierarchical Multimedia Storage Server. In: VLDB Conf., 1997, Athens, Greece, pp 226–235

[VLN97] Venkataraman S, Livny M, Naughton JF (1997) Memory Management for Scalable Web Data Servers. In: Int. Conf. on Data Engineering, 1997, Birmingham, UK, pp 510–519

[WHMZ94] Weikum G, Hasse C, Moenkeberg A, Zabback P (1994) The COMFORT Automatic Tuning Project. Inf Syst 19(5): 381–432

[Wo83] Wong CK (1983) Algorithmic Studies in Mass Storage Systems. Computer Science Press, New York

[WZ86] Wedekind H, Zoerntlein G (1986) Prefetching in Realtime Database Applications. In: ACM SIGMOD Conf., 1986, Washington, D.C., pp 215–226