A Performance Model of Speculative Prefetching in Distributed Information Systems

N. J. Tuah* M. Kumar S. Venkatesh
School of Computing, Curtin University of Technology
GPO BOX U1987, WA 6845, Australia
(tuahanj,kumar,svetha)@cs.curtin.edu.au

Abstract

Previous studies in speculative prefetching focus on building and evaluating access models for the purpose of access prediction. This paper investigates a complementary area which has been largely ignored, that of performance modelling. We use improvement in access time *as the performance metric, for which we derive a formula in terms of resource parameters (time available and time required for prefetching) and speculative parameters (probabilities for next access). The performance maximisation problem is expressed as a* stretch knapsack problem. *We develop an algorithm to maximise the improvement in access time by solving the stretch knapsack problem, using theoretically proven apparatus to reduce the search space. Integration between speculative prefetching and caching is also investigated, albeit under the assumption of equal item sizes.*

1. Introduction

Caching and prefetching of data have been used to improve the speed of information access. In caching, copies of remote data are kept locally to reduce access time of repeatedly accessed data [11, 8]. In prefetching, access to remote data is anticipated and the data is fetched before it is required [6]. This is in contrast to *demand fetch* where data is fetched only when it is actually requested.

Prefetching can either be *speculative* or *informed*. In this paper we investigate speculative prefetching. Previous studies in speculative prefetching (see Section 1.1 Related work) focus on building access models and evaluating the performance of such models in predicting future accesses. While these models are important, they do not constitute a complete framework for building optimal prefetch strategies. We believe that, in addition to an access model, a prefetcher requires a resource model and a performance model. A resource model allows a prefetcher to predict the amount of available and required resources. A performance model allows a prefetcher to optimise the usage of resources and adapt well to changing resource conditions.

In this paper, we develop prefetching algorithms using a performance model. Our model presupposes some knowledge about future accesses. In particular, it has a list of candidate items for the next access. It also presupposes some knowledge about available and required resources. In particular, the time available for prefetching, the retrieval time for each item, and cache size are known.

1.1. Related work

Many recent studies in speculative prefetching assume persistence in trends of user request patterns. Tait [14] uses file access pattern based on the features of UNIX-style operating system where every program gives rise to a tree of forked processes that access some files. Vitter [16] uses data compression techniques to build an access tree that can make optimal predictions if the accesses are generated by a Markov process.

Speculative prefetching has been proposed for improving web access [1, 5, 9]. Padmanabhan [9] suggests server-side prediction of document access. The server builds a dependency graph where each link is labelled with the probability of the follow-up access being made. In the ETEL electronic newspaper project [1], the client builds a patterned frequency graph that contains a path for each sequence of accesses. Jiang [5] combines server-side and client-side prediction for web browsing. Jiang suggests an adaptive prefetching scheme based on a performance model that considers network usage time and user's waiting time.

In [15], we investigate the performance of speculative prefetching under a model in which prefetching is neither aborted nor preempted by demand fetch but instead gets equal priority in network bandwidth utilisation.

Prefetching competes for memory resources with caching. We found excellent work on the integration of informed prefetching and caching [2, 10], but we found no analogous published articles in speculative prefetching.

The rest of the paper is organised as follows. Section 2 describes the parameters in our model and defines the performance metric, namely access improvement, that we want to optimise. In Section 3 we derive a formula for access improvement when the cache is empty. In Section 4, we present a solution, in the form of an algorithm, for maximising access improvement. In Section 5, the cache content is incorporated into the analysis. In Section 6, we highlight the main points of this research and suggest future work.

*Under scholarship of Brunei government through Universiti Brunei Darussalam

2. Model

The opportunity for prefetching comes when an application is waiting for the user input or carrying out some processing. For convenience, we shall refer to the time of such opportunity as the *viewing time*. We use the term *retrieval time* to refer to the time to fully retrieve an item. When a remote item is actually requested, the network may appear to be more responsive if the requested item has been prefetched and is already partially or fully retrieved. We use the term *access time* to refer to the response time to an actual request. The terminology we use for the time durations is illustrated in Figure 1.

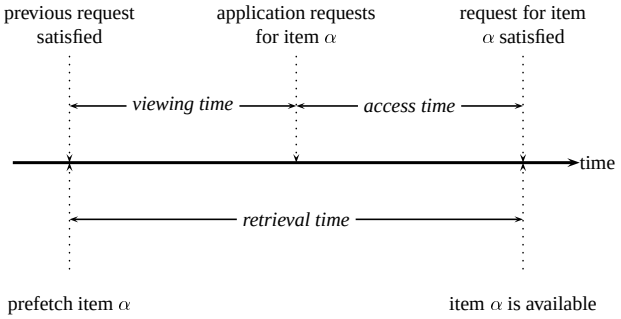


Figure 1. Time durations

When a request is made while a prefetch of a different item is still in progress, and thus necessitating a demand fetch, we assume that the prefetch completes before the demand fetch. A wrong prefetch may thus increase the access time.

We want to maximise the expected improvement in access time, referred to simply as *access improvement*. Access improvement, denoted by g, is defined as $E(T_{(\text{no prefetch})}) - E(T_{(\text{prefetch})})$ where $T_{(x)}$ is the access time given condition x.

We shall confine our analysis to one-access look-ahead. Thus, the prefetch strategy we formulate will be a greedy one in the sense that it tries to optimise the performance of the next single access without considering the effect of its decision further into the future.

Notation: We use $\langle\,\rangle$ to enclose a list of items and use this $\mathcal{FONT}$ for list names. $\mathcal{R} \cdot \mathcal{S}$ is the concatenation of $\mathcal{R}$ and $\mathcal{S}$. $|\mathcal{R}|$ is the number of elements in $\mathcal{R}$. The symbols for set operations — $\in, \subset, \subseteq$ and $\setminus$ — are used for list operations with their usual respective meanings. Items that might be accessed are uniquely numbered and they are referred to by their numbers. We use the following symbols: n for the number of items, $\mathcal{N}$ for $\langle 1, \ldots, n\rangle$ which is the list of all items, v for the viewing time, r_i for the retrieval time of item i, and P_i for $P(\alpha = i)$, where α is a random variable denoting the item to be accessed next.

3. Prefetch only

We begin by assuming that the cache is empty. We shall use the symbols T° and g° for access time and access improvement, respectively, when the cache is empty. Let the list of items to be prefetched, $\mathcal{F}$, be constructed as follows:

$$\mathcal{F} = \mathcal{K} \cdot \langle z \rangle \qquad \text{where } \mathcal{K} \subset \mathcal{N},\ z \in \mathcal{N} \setminus \mathcal{K} \text{ and } \sum_{i \in \mathcal{K}} r_i < v \tag{1}$$

The items are prefetched in sequence so that z is the last item to be prefetched. Note that this construction is general and requires only that $\mathcal{F}$ is not empty and all prefetches are initiated before the next request is made. We specify the construction for $\mathcal{F}$ more for notation rather than restriction.

When no prefetch is performed, the access time equals the retrieval time of the requested item. Hence, the expected access time is

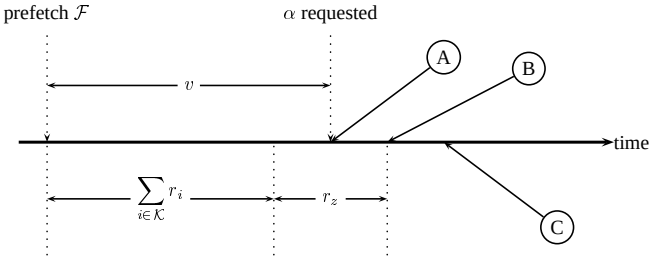
$$E(T^\circ_{(\text{no prefetch})}) = \sum_{i \in \mathcal{N}} P_i r_i$$

When $\mathcal{F}$ is prefetched, its retrieval time may exceed the viewing time. We refer to the amount by which the retrieval time of $\mathcal{F}$ exceeds the viewing time as the *stretch time* and denote it as $st(\mathcal{F})$. This is defined as

$$st(\mathcal{F}) \;=\; \max\left\{0, \sum_{i \in \mathcal{F}} r_i - v\right\} \tag{2}$$

As shown in Figure 2, the access time when $\mathcal{F}$ is prefetched can be: $T^\circ_{(\text{prefetch } \mathcal{F}, \alpha \in \mathcal{K})} = 0$, $T^\circ_{(\text{prefetch } \mathcal{F}, \alpha = z)} = st(\mathcal{F})$, or $T^\circ_{(\text{prefetch } \mathcal{F}, \alpha \notin \mathcal{F})} = st(\mathcal{F}) + r_\alpha$. Hence, the expected access time is

$$E(T^\circ_{(\text{prefetch } \mathcal{F})}) = P_z\, st(\mathcal{F}) + \sum_{i \in \mathcal{N} \setminus \mathcal{F}} P_i (r_i + st(\mathcal{F}))$$



(A) If $\alpha \in \mathcal{K}$, access time is zero. (B) If $\alpha = z$, access time equals the time to complete the prefetch. (C) If $\alpha \notin \mathcal{F}$, access time equals the time to complete the prefetch, plus the time to retrieve the item actually requested.

Figure 2. Access time

Hence, the access improvement when the cache is empty and $\mathcal{F}$ is prefetched is,

$$\begin{aligned} g^\circ(\mathcal{F}) &= E(T^\circ_{(\text{no prefetch})}) - E(T^\circ_{(\text{prefetch } \mathcal{F})}) \\ &= \sum_{i \in \mathcal{F}} P_i r_i - \sum_{i \in \mathcal{N} \setminus \mathcal{K}} P_i\, st(\mathcal{F}) \end{aligned} \tag{3}$$

4. Stretch knapsack problem

Assuming the cache is empty, the optimal list of items to prefetch can be obtained by solving the following problem:

$$\text{Find } \mathcal{F} \text{ to maximise } g^\circ(\mathcal{F}) \tag{4}$$

We refer to the problem in (4) as the *stretch knapsack problem* (SKP). SKP is like a binary knapsack problem (KP) [7] where the profit and the weight of item i are $P_i r_i$ and r_i respectively, and the knapsack capacity is v. However, unlike KP, the total weights of items inserted into the stretch knapsack may exceed its capacity, causing it to stretch by an amount of $st(\mathcal{F})$.

4.1. Anatomy of search space

The search space for KP consists of all the possible combinations of the items. The search space for SKP is a superset of the search space for KP. The former includes not only different combinations of the items, but also certain permutations of some of these combinations. In particular, if $st(\mathcal{F}) > 0$ and $\mathcal{F}^*$ is a permutation of $\mathcal{F}$, then it is possible that $g^\circ(\mathcal{F}) \neq g^\circ(\mathcal{F}^*)$.

Theorem 1 *Let $\bar{\mathcal{F}}$ be a list of items the total retrieval times of which exceeds v. Let $\bar{z}$ be the last element in $\bar{\mathcal{F}}$. If $\bar{\mathcal{F}}$ is an optimal solution to the problem in (4), then* $\min\left\{P_f : f \in \bar{\mathcal{F}}\right\} = P_{\bar{z}}$.

Proof: Suppose $\exists f \in \bar{\mathcal{F}}$ such that $P_{\bar{z}} > P_f$. We will show that this cannot be so if $\bar{\mathcal{F}}$ is an optimal solution, and hence by contradiction the theorem must be true. Let $\bar{\mathcal{K}}$ be the list of all elements in $\bar{\mathcal{F}}$ excluding $\bar{z}$. (In order words, $\bar{\mathcal{F}} = \bar{\mathcal{K}} \cdot \langle \bar{z} \rangle$). Form a list $\mathcal{K}^*$ which is the same as $\bar{\mathcal{K}}$ except that element f is replaced with $\bar{z}$. Let $\mathcal{F}^* = \mathcal{K}^* \cdot \langle f \rangle$. From (3), $g^\circ(\bar{\mathcal{F}}) = \sum_{i\in\bar{\mathcal{F}}} P_i r_i - \sum_{i\in\mathcal{N}\setminus\bar{\mathcal{K}}} P_i st(\bar{\mathcal{F}})$ and $g^\circ(\mathcal{F}^*) = \sum_{i\in\mathcal{F}^*} P_i r_i - \sum_{i\in\mathcal{N}\setminus\mathcal{K}^*} P_i st(\mathcal{F}^*)$. Since $\bar{\mathcal{F}}$ and $\mathcal{F}^*$ contain the same items, we get $\sum_{i\in\bar{\mathcal{F}}} P_i r_i = \sum_{i\in\mathcal{F}^*} P_i r_i$ and $st(\bar{\mathcal{F}}) = st(\mathcal{F}^*)$. From our supposition that $P_{\bar{z}} > P_f$ and the way $\mathcal{K}^*$ is constructed, we get $\sum_{i\in\mathcal{K}^*} P_i > \sum_{i\in\bar{\mathcal{K}}} P_i \Longrightarrow \sum_{i\in\mathcal{N}\setminus\bar{\mathcal{K}}} P_i > \sum_{i\in\mathcal{N}\setminus\mathcal{K}^*} P_i$. Hence $g^\circ(\bar{\mathcal{F}}) < g^\circ(\mathcal{F}^*)$. ■

Theorem 1 allows us to confine the search space to permutations where the items are sorted in descending order of probability. Equally probable items are sub-sorted in increasing retrieval times. Henceforth, we shall assume

$$(P_i > P_{i+1}) \text{ or } (P_i = P_{i+1} \text{ and } r_i \leq r_{i+1}) \tag{5}$$

4.2. Relaxation and upper bound

SKP is an integer programming problem. In the context of prefetching, an item is either entirely prefetched or not at all. By allowing items to be partially prefetched, we obtain the linear programming relaxation of SKP (linear SKP).

Let x_i, where $0 \leq x_i \leq 1$, be the proportion of item i that is prefetched. We use x without the subscript to refer to the entire array $x_1, \ldots, x_n$. Let z be the last item to be prefetched. Let $\tilde{\mathcal{K}}$ be the list of wholly prefetched items not including z; i.e. $\tilde{\mathcal{K}} = \langle i : x_i = 1, i \neq z \rangle$. The linear SKP is

$$\text{Maximise } \tilde{g}^\circ(x) \tag{6}$$

where $\tilde{g}^\circ(x) = \sum_{i\in\mathcal{N}} P_i r_i x_i - \sum_{i\in\mathcal{N}\setminus\tilde{\mathcal{K}}} P_i \tilde{st}(x)$ and $\tilde{st}(x) = \max\left\{0, \sum_{i\in\mathcal{N}} r_i x_i - v\right\}$.

Suppose that the items, sorted according to (5), are consecutively inserted into the stretch knapsack until the first item, $\tilde{z}$, is found which does not fit, i.e.,

$$\tilde{z} = \min\left\{ j : \sum_{i=1}^{j} r_i > v \right\}$$

Theorem 2 *The optimal solution of the linear SKP is $\bar{x}$ defined as:*

$$\bar{x}_i = \begin{cases} 1 & \text{if } 1 \leq i \leq \tilde{z}-1, \\ (v - \sum_{i=1}^{\tilde{z}-1} r_i)/r_{\tilde{z}} & \text{if } i = \tilde{z} \\ 0 & \text{if } \tilde{z}+1 \leq i \leq n \end{cases}$$

Proof: By Dantzig's Theorem [3], $\bar{x}$ is the solution to the linear programming relaxation of KP where the profit and the weight of item i are $P_i r_i$ and r_i respectively, and the knapsack capacity is v. Hence this is also the solution to (6) when $\tilde{st}(x) = 0$. We are left with the case of $\tilde{st}(x) > 0$ to prove.

Suppose the optimal solution is x^* where $\tilde{st}(x^*) > 0$ and the last item inserted into the knapsack is z^*. But if we decrease the value of $x^*_{z^*}$ by ε while maintaining $\tilde{st}(x^*) > 0$, the value of $\tilde{g}^\circ$ changes by an amount of $(1 - \sum_{i\in\tilde{\mathcal{K}}} P_i - P_{z^*}) r_{z^*} \varepsilon$, which is greater or equal to zero. Thus we have obtained a better or an equally good solution. We can iterate to obtain an even better (or an equally good) solution. At the limit of $\tilde{st}(x^*) \to 0$, we can apply the Dantzig's Theorem. ■

Since SKP solution space is a subset of linear SKP solution space, a tight upper bound on g° is given by,

$$U_{g^\circ} = \tilde{g}^\circ(\bar{x}) = \sum_{i=1}^{\tilde{z}-1} P_i r_i + (v - \sum_{i=1}^{\tilde{z}-1} r_i) P_{\tilde{z}} \tag{7}$$

4.3. An algorithm for exact solution

Our algorithm for the exact solution of SKP is shown in Figure 3. We shall refer to it simply as the *SKP algorithm*. It is based on Horowitz-Sahni algorithm for KP [4].

Theorem 3 $g^\circ(\mathcal{F}) = g^\circ(\mathcal{K}) + \delta$ *where* $\delta = P_z r_z - (1 - \sum_{i\in\mathcal{K}} P_i) st(\mathcal{F})$, *and $\mathcal{F}$, $\mathcal{K}$ and z are as defined in (1).*

Proof: From (3), $g^\circ(\mathcal{F}) = \sum_{i\in\mathcal{F}} P_i r_i - \sum_{i\in\mathcal{N}\setminus\mathcal{K}} P_i st(\mathcal{F})$ and $g^\circ(\mathcal{K}) = \sum_{i\in\mathcal{K}} P_i r_i$ (since $st(\mathcal{K}) = 0$). The theorem follows. ■

The SKP algorithm assumes that the items are sorted according to 5. It performs *forward moves* and *backtracking moves*. A forward move consists of inserting as many consecutive items as possible to raise the value of g°, using the formula in Theorem 3 to calculate it incrementally. When an item causes g° to decrease, it is excluded and

the upper bound of the currently constructed solution is computed. If the upper bound is lower than the value of the best solution so far, the algorithm backtracks; otherwise it performs a deeper forward move. When the knapsack stretches to accommodate an item (or when no item remains), the current solution is complete with the remaining items deemed excluded. A backtracking move consists of removing the last inserted item from the solution.

input: n, P, r, v **output:** $\mathcal{F}$

1. $j := 1$ (* initialise *)
   $x := 0, \ldots, 0$ (* best item selectors *)
   $g := 0$ (* g°(best solution) *)
   $\hat{x} := 0, \ldots, 0$ (* current item selectors *)
   $\hat{g} := 0$ (* g°(current solution) *)
   $\hat{v} := v$ (* current residual capacity $(v - \sum_{i=1}^{n} r_i \hat{x}_i)$ *)
   $P_{n+1} := 0$
   $r_{n+1} := \infty$
2. Find $\tilde{z} = \min\left\{k : \sum_{i=j}^{k} r_i > \hat{v}\right\}$ (* compute upper bound *)
   $U := \sum_{i=j}^{\tilde{z}-1} P_i r_i + (\hat{v} - \sum_{i=j}^{\tilde{z}-1} r_i) P_{\tilde{z}}$
   if $g \geq \hat{g} + U$ **then goto** 5 **endif**
3. **while** $j \leq n$ and $\hat{v} > 0$ **do** (* perform a forward step *)
   $\delta := P_j r_j - \sum_{i=j}^{n} P_i \max\{0, r_j - \hat{v}\}$
   if $\delta \leq 0$ **then**
   $\hat{x}_j := 0$
   $j := j + 1$
   if $j < n$ **then goto** 2 **endif**
   else
   $\hat{v} := \hat{v} - r_j$
   $\hat{g} := \hat{g} + \delta$
   $\hat{x}_j := 1$
   $j := j + 1$
   endif
   endwhile
4. **if** $\hat{g} > g$ **then** (* update the best solution *)
   $g := \hat{g}$
   $x := \hat{x}$
   endif
5. Find $k = \max\{i < j : \hat{x}_i = 1\}$ (* backtrack *)
   if no such k **then goto** 6 **endif**
   $\hat{x}_k := 0$
   $\hat{v} := \hat{v} + r_k$
   $\delta := P_k r_k - \sum_{i=k}^{n} P_i \max\{0, r_k - \hat{v}\}$
   $\hat{g} := \hat{g} - \delta$
   $j := k + 1$
   goto 2
6. $\mathcal{F} := \langle i : x_i = 1 \rangle$ (* final solution *)

Figure 3. SKP algorithm

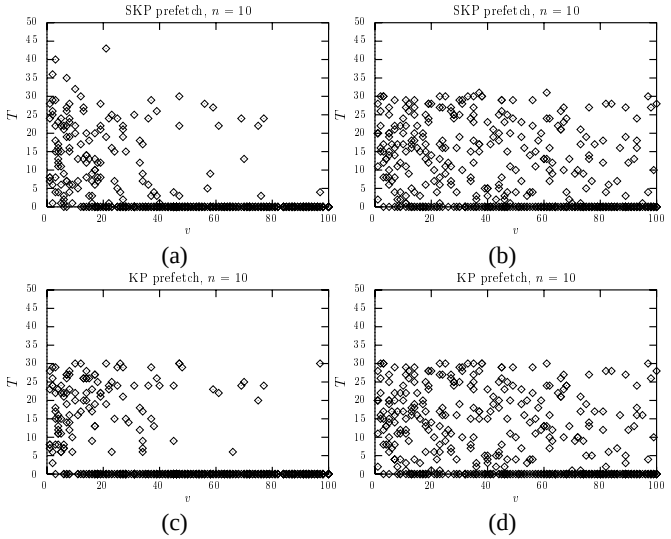
4.4. Effect of stretch time

The SKP algorithm optimises the list of items to prefetch for the next single access. Accesses further into the future are disregarded. In particular, the stretch time may intrude into the next viewing time and thus reducing the asset for the next prefetch.

To investigate the effect of the use of stretch time, we perform 'prefetch only' simulation. In the 'prefetch only' simulation the cache is used only for prefetching items. Once a request is satisfied the cache is flushed out. The simulation consists of running 50,000 iterations through the following steps: 1) generate n, P, r and v randomly, 2) prefetch, 3) generate a random request, 4) calculate access time, 5) output v and T. The values for P are generated using two different methods: *skewy* method and *flat* method. The skewy method generates a situation where the next request is highly predictable. The flat method results in a less predictable situation.

Four different prefetch methods are employed in the simulation: *SKP prefetch*, *KP prefetch*, *perfect prefetch* and *no prefetch*. The SKP prefetch and the KP prefetch use, respectively, the SKP solution and the KP solution to select items for prefetch. The perfect prefetch always prefetches the correct item.

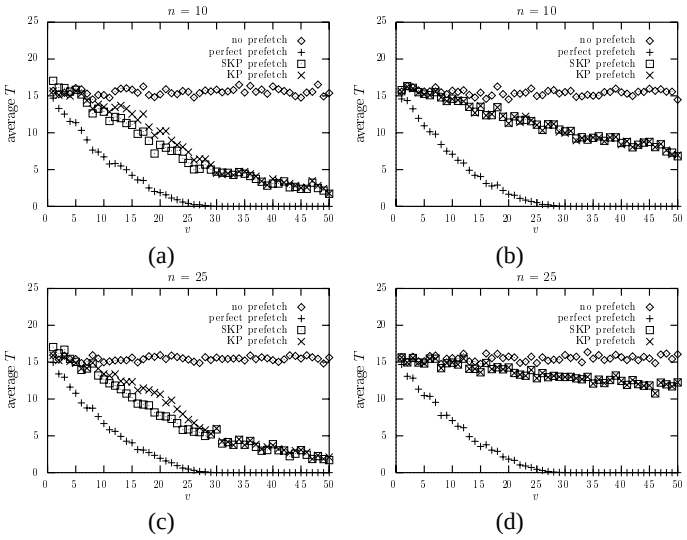
Figure 4 shows scatter plots of T against v for the SKP prefetch and the KP prefetch. The negative effect of using stretch time can be seen in Figure 4a where some points appear above $T = 30$ even though the maximum value for r is only 30. On the other hand, the more conservative approach of the KP prefetch may result in under utilisation of the viewing time, as can be seen in Figure 4c. The dense triangular area above the line $T = v$ can be explained by failure to prefetch highly probable items whose retrieval times exceed v. Figures 4b and 4d, for which future accesses are less predictable, are almost identical.



The result of 500 iterations of the 'prefetch only' simulation is plotted for the SKP prefetch and the KP prefetch. The simulation parameters are: $n = 10$, v is uniformly distributed from 1 to 100, r is uniformly distributed from 1 to 30. For figures (a) and (c), P is generated using the skewy method; for figure (b) and (d), the flat method is used.

Figure 4. Scatter plot for 'prefetch only'

Figure 5 shows the average access time against v. In Figures 5a and 5c, for which the skewy method is used, the performances of the SKP prefetch is slightly better than that of the KP prefetch. The exception is when v is small where the SKP prefetch performs worse than no prefetch. In Figures 5b and 5d, for which the flat method is used, the performances of the SKP prefetch and the KP prefetch are almost the same. Increasing the number of items from 10 to 25 has the effect of increasing the average access time. The increase is expected; in the extreme case when $n = \infty$ and there is no clearly dominating items, any speculative prefetch will be in vain.



Each plot is obtained by running the 'prefetch only' simulation for 50000 iterations. The simulation parameters are: $n = 10$ and 25, v ranges from 1 to 100 (though the plot is clipped at $v = 50$), r ranges from 1 to 30, and P is generated using the skewy method for figures (a) and (c), and the flat method for figures (b) and (d).

Figure 5. Performance of prefetch

5. Prefetch and cache

In this section, we consider the cache not empty. Items to be prefetched must contest the items already in the cache. We first derive a formula for access improvement, $g(\mathcal{F}, \mathcal{D})$, where $\mathcal{F}$ is the list of prefetched items, and $\mathcal{D}$ is the list of items ejected from the cache. We then discuss how to solve the following problem:

$$\text{Find } \langle \mathcal{F}, \mathcal{D} \rangle \text{ to maximise } g(\mathcal{F}, \mathcal{D}) \tag{8}$$

5.1. Access improvement

Let $\mathcal{F}$ be constructed as in (1) except that now $\mathcal{F}$ cannot have any elements in common with the cache, $\mathcal{C}$.

When no prefetch is performed, the expected access time is given by

$$E(T_{(\text{no prefetch})}) = \sum_{i \in \mathcal{N} \setminus \mathcal{C}} P_i r_i$$

When $\mathcal{F}$ is prefetched, and $\mathcal{D}$ is ejected to give room, the access time is as follows: $T_{(\mathcal{F} \text{ ejects } \mathcal{D}, \alpha \in \mathcal{K} \cdot \mathcal{C} \setminus \mathcal{D})} = 0$, $T_{(\mathcal{F} \text{ ejects } \mathcal{D}, \alpha = z)} = \mathit{st}(\mathcal{F})$ and $T_{(\mathcal{F} \text{ ejects } \mathcal{D}, \alpha \notin \mathcal{F} \cdot \mathcal{C} \setminus \mathcal{D})} = \mathit{st}(\mathcal{F}) + r_\alpha$. Hence, the expected access time is given by

$$E(T_{(\mathcal{F} \text{ ejects } \mathcal{D})}) = \sum_{i \in \mathcal{C} \setminus (\mathcal{F} \cdot \mathcal{C} \setminus \mathcal{D})} P_i r_i + \sum_{i \in \mathcal{N} \setminus (\mathcal{K} \cdot \mathcal{C} \setminus \mathcal{D})} P_i \mathit{st}(\mathcal{F})$$

Hence, the access improvement is

$$\begin{aligned} g(\mathcal{F}, \mathcal{D}) &= E(T_{(\text{no prefetch})}) - E(T_{(\mathcal{F} \text{ ejects } \mathcal{D})}) \\ &= g^\circ(\mathcal{F}) - (\sum_{i \in \mathcal{D}} P_i r_i - \sum_{i \in \mathcal{C} \setminus \mathcal{D}} P_i \mathit{st}(\mathcal{F})) \end{aligned} \tag{9}$$

5.2. Maximising g

The complexity of the search space for the problem in (8) is the same as that of SKP. However, we have not been able to come up with a bounding technique to subdue its combinatorial explosion. So we will settle for a suboptimal solution. Furthermore, we shall assume that item sizes are equal. Consequently, the number of prefetched items must equal the number of ejected items, i.e. $|\mathcal{F}| = |\mathcal{D}|$.

Equation (9) suggests the following method. First, using SKP algorithm, find $\mathcal{F}$ to maximise $g^\circ(\mathcal{F})$. Then find $\mathcal{D}$ to minimise $\sum_{i \in \mathcal{D}} P_i r_i - \sum_{i \in \mathcal{C} \setminus \mathcal{D}} P_i \mathit{st}(\mathcal{F})$, which we will refer to as *anti-*g. The minimisation problem can simply be solved by sorting $\mathcal{C}$ in ascending order of $P_i(r_i + \mathit{st}(\mathcal{F}))$ and taking the first $|\mathcal{F}|$ elements. There may exist $f \in \mathcal{F}$ and $d \in \mathcal{D}$ such that the contribution of f to $g^\circ(\mathcal{F})$ is less than the contribution of d to anti-g. In this case $\langle \mathcal{F} \setminus \langle f \rangle, \mathcal{D} \setminus \langle d \rangle \rangle$ is a better solution than $\langle \mathcal{F}, \mathcal{D} \rangle$. So we must include an arbitration step to prevent ejection of an item from the cache by a less-worthy replacement.

For the purpose of arbitration, we assume that $\mathit{st}(\mathcal{F})$ is zero so that item $f \in \mathcal{F}$ contributes $P_f r_f$ to $g^\circ(\mathcal{F})$ and item $d \in \mathcal{D}$ contributes $P_d r_d$ to anti-g. Item f can only be prefetched if it can find a victim d such that $P_d r_d = \min_{i \in \mathcal{C}} P_i r_i$ and $P_f r_f > P_d r_d$. Demand-fetched item, however, *must* have a victim and only requires the first condition. We call this Pr*-arbitration*. Our algorithm to maximise g is shown in Figure 6.

input: $n, P, r, v, \mathcal{C}$ **output:** $\mathcal{F}, \mathcal{D}$
Find $\hat{\mathcal{F}} \subseteq \mathcal{N} \setminus \mathcal{C}$ to maximise $g^\circ(\hat{\mathcal{F}})$
$\mathcal{F} := \langle\rangle$
$\mathcal{D} := \langle\rangle$
for each $f \in \hat{\mathcal{F}}$ sorted in descending $P_f r_f$ **do**
  Find $d \in \mathcal{C}$ with smallest $P_d r_d$
  if $P_f r_f < P_d r_d$ **then break endif**
  $\mathcal{F} := \mathcal{F} \cdot \langle f \rangle$
  $\mathcal{D} := \mathcal{D} \cdot \langle d \rangle$
  $\mathcal{C} := \mathcal{C} \setminus \langle d \rangle$
endfor

Figure 6. SKP algorithm with Pr-arbitration

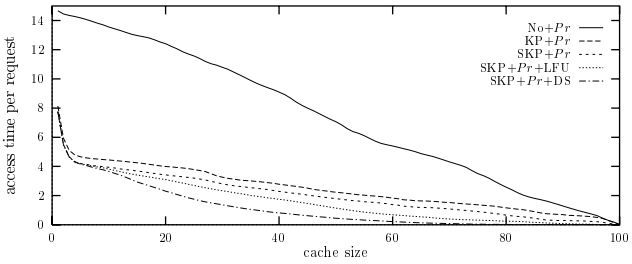
To choose from among potential victims that have the same Pr value, we employ a sub-arbitration. For this purpose, we define for each item a value called *delay-saving profit* as $\mathit{freq}_i \times r_i$, where freq_i is the frequency of accesses to item i. This formula is a simplified form of the one used by WATCHMAN cache [12] and its web-related spawn [13]. When sub-arbitration is required, we choose the item with the lowest delay-saving profit. We call this *DS-arbitration*. Admittedly, while everything else is meticulously derived to solve problem (8), the sub-arbitration is included ad-hoc.

5.3. Performance

We found no published prefetching techniques that can be used as a fair yardstick against our work. This is partly because of its assumption of equal item sizes and partly because our emphasis is different from previous work by other people (see Section 1.1). The experiment in this section, therefore, only serves to gain insight especially on the effect of sub-arbitration, which is an ad-hoc inclusion.

We use Monte Carlo simulation to see how SKP with arbitration performs. We simulate five different prefetch-cache policies: 1) *No+*Pr: no prefetch is performed and Pr-arbitration is used to select cache victims, 2) *KP+*Pr: KP solution is used with Pr-arbitration, 3) *SKP+*Pr: SKP solution is used with Pr-arbitration, 4) *SKP+*Pr*+LFU*: Same as previous one, with sub-arbitration using LFU (least frequently used), 5)*SKP+*Pr*+DS*: Same as previous one, except DS-arbitration is used instead of LFU.

Figure 7 shows the result of the simulation. The figure confirms that SKP prefetch performs better than KP prefetch. Adding sub-arbitration clearly improves the result. We are not surprised that SKP+Pr+DS gives the best result. Pr-arbitration protects immediate candidates and DS-arbitration keeps in cache the items that would otherwise consume too much network time.



Each curve is plotted by joining 100 points. Each point is obtained by generating 50000 requests and taking the average access time. The requests are generated using a 100-state Markov source. When going to state i, the Markov source generates a request for item i and, after the request is served, it waits for the duration of v_i, where $1 \leq v_i \leq 100$, before changing to another state. The state transition matrix is constructed such that there are 10 to 20 possible transitions from any state. Retrieval times for items are between 1 to 30. We vary cache size from 1 to 100.

Figure 7. Performance of prefetch-cache

6. Conclusions and further work

We have presented in this paper a performance model for speculative prefetching, incorporating resources (retrieval time, viewing time and cache) and access prediction (probabilities for next access). A prefetching algorithm is developed to maximise access improvement. The algorithm uses theoretically proven apparatus to reduce its search space. We integrate the prefetching algorithm with cache replacement using a two-stage arbitration. However, we assume uniform size for all items. We are currently addressing this limitation.

The SKP algorithm considers only one access ahead. Obviously, looking ahead deeper will improve the performance. However, the complexity of the problem can be daunting.

The SKP algorithm with arbitration maximises access improvement without regard to the increase in network usage. Even if the most probable items are already in the cache, it will prefetch the lesser candidates if, by doing so, it can improve the expected access time even by an insignificant amount. A policy is needed to weigh the opposing goals of maximising access improvement and minimising network usage.

Our model presupposes some knowledge about future accesses. Access modelling has received much attention (see Section 1.1 Related work). One of the models proposed in the literature (e.g. [1, 16]) might serve the purpose of providing this knowledge. Our model also presupposes some knowledge about available and required resources. There is much work to be done in this area.

References

[1] M. Banâtre, V. Issarny, F. Leleu, and B. Charpiot. Providing quality of service over the web: A newspaper-based approach. In *6th Int WWW Conf*, Apr. 1997.

[2] P. Cao. *Application-Controlled File Caching and Prefetching*. PhD thesis, Department of Computer Science, Princeton University, Jan. 1996.

[3] G. B. Dantzig. Discrete variable extremum problems. *Operations Research*, 5:266–277, 1957.

[4] E. Horowitz and S. Sahni. Computing partitions with applications to the knapsack problem. *Journal of ACM*, 21:277–292, 1974.

[5] Z. Jiang and L. Kleinrock. An adaptive network prefetch scheme. *IEEE Journal on Selected Areas in Communications*, 16(3):358–368, Apr. 1998.

[6] H. Lei and D. Duchamp. An analytical approach to file prefetching. In *Proc USENIX Annual Technical Conf*, Jan. 1997.

[7] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementation*. Wiley, 1990.

[8] M. N. Nelson, B. B. Welch, and J. K. Ousterhout. Caching in the sprite network file system. *ACM Trans on Computer Systems*, 6(1), 1988.

[9] V. N. Padmanabhan and J. C. Mogul. Using predictive prefetching to improve world wide web latency. *ACM SIGCOMM Computer Communication Review*, pages 22–36, July 1996.

[10] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed prefetching and caching. In *Proc 15th ACM Symposium on Operating System Principles*, pages 79–95, Dec. 1995.

[11] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the sun network file system. In *Proc Summer 1985 USENIX Conf*, pages 119–130, June 1985.

[12] P. Scheuermann, J. Shim, and R. Vingralek. WATCHMAN: A data warehouse intelligent cache manager. In *22nd VLDB Conf*, 1996.

[13] P. Scheuermann, J. Shim, and R. Vingralek. A case for delay-conscious caching of web documents. In *6th Int WWW Conf*, Apr. 1997.

[14] C. D. Tait. *A File System for Mobile Computing*. PhD thesis, Graduate School of Arts and Sciences, Columbia University, 1993.

[15] N. J. Tuah, M. Kumar, and S. Venkatesh. Investigation of a prefetch model for low bandwidth networks. In S. K. Das, editor, *1st ACM Int Workshop on Wireless Mobile Multimedia*, pages 38–47, Oct. 1998.

[16] J. S. Vitter and P. Krishnan. Optimal prefetching via data compression. In *IEEE 32nd Annual Symposiums on Foundation of Computer Science*, pages 121–130, 1991.