Analysing Object Relationships to Predict Page Access for Prefetching

Nils Knafla
Dept. of Computer Science
University of Edinburgh
Edinburgh EH9 3JZ, Scotland
http://www.dcs.ed.ac.uk/home/nk

Abstract

We present a new approach to predicting page accesses to facilitate prefetching persistent objects in a client/server computing environment. The relationships between persistent objects are modelled by a *discrete-time Markov Chain*, which allowed us to use a method called *hitting times* to compute the page access probability and the mean time to access a page. If the probability of a page is higher than a threshold defined by cost/benefit parameters then the page is a candidate for prefetching. To determine the prefetching threshold we consider various cost parameters to compare the benefit of a correct prefetch with the cost of an incorrect prefetch. In addition, we compute the best possible time to start the prefetch operation. We incorporated this prefetching algorithm into the EXODUS storage manager, and used the timing results in a simulation.

Keywords: prefetching, persistent object stores, object-oriented databases, discrete-time Markov Chains, hitting times and hitting probabilities, EXODUS storage manager, high performance object stores, client/server computing.

1 Introduction

1.1 Performance Bottlenecks of Persistent Object Stores

In many client/server persistent object stores the client requests pages from the server. The server retrieves the page from its local disk and sends it to the client. The page is inserted into the client's address space and the application can work with the objects in the page. This page fetch is an expensive operation. In Figure 1 we give an example to show the most expensive elements of such a page fetch. The results were obtained by timing the EXODUS storage manager (ESM) [2]; details about our computing environment can be found in Section 4.1. This test was made with just one client at the server and the machine workload was low.

A major cost factor of the page fetch is the cost for sending and receiving a page (setup costs) via the network. The network transfer cost is low, compared with the setup costs. All elapsed times of the cost components (apart from network transfer and disk read) are dependent on the speed of the processor. Thus these costs could be reduced, using up-to-date processors, by a factor of 6 and network transfer could be reduced by higher bandwidths in which case the disk access would emerge as the major bottleneck. The seek cost is the most expensive part of the disk access but does not appear in Figure 1 because we read all pages sequentially from disk. The IPC cost could be reduced by using a disk thread instead of a disk process.

1.2 Approaches to Improve Database Performance

In order to reduce the high cost of a page fetch many researchers have developed prefetching techniques. In object-oriented databases Chang [3] predicts future accesses by hints from the data semantics in terms of inheritance and structural relationships to prefetch one object ahead. Cheng [4] extended this work by adding multiple hints and by taking into account the prefetching depth and physical storage considerations. A complex assembly operator to load component objects recursively in advance was introduced by Keller [9]. In Fido [15], the prefetching technique employs an associative memory to recognise access patterns within a context over time. In training mode, object access information is gathered and stored with a *nearest-neighbour* associative memory. In prediction mode, this information is used to recognize previously encountered situations. Gerlhof [6] records the answer of an operation, i.e. finds the identifiers of all pages that were accessed during the execution of an operation and uses this information

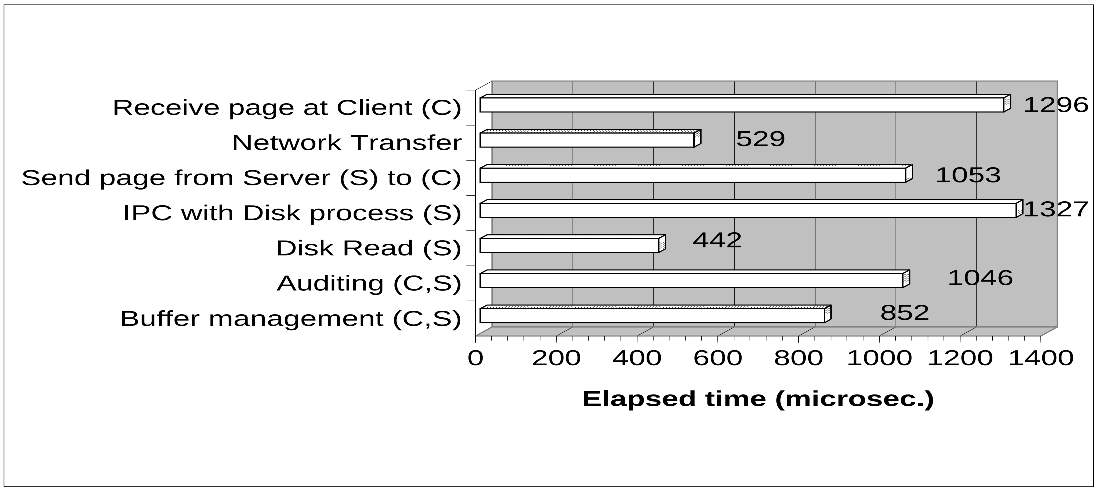

Figure 1: Expensive components of a page fetch

to prefetch this sequence of pages. In Thor [5] each fetch request from the client causes the server to select a prefetch group containing the object requested and related objects.

In some research work an application's future accesses are predicted by probability models, the values of which were obtained from past accesses. In [7] a probability graph was used to predict the future file access, i.e. the probability that one file will be opened after another. In the WWW Bestavros [1] speculated about document accesses based upon their document interdependency probabilities. Grimsrud [8] keeps a record of the disk clusters which were accessed immediately after another cluster with an associated weight for the access probability. An interesting study to preload documents from tertiary storage was made by Kraiss [12]. This work is probably most related to our work because it uses a *continuous-time Markov-chain model* to predict the document access. The main difference to our work is that they predict the documents access whereas we predict the future page access based on object relationships.

The new approach of our work is the computation of the page access probability considering the structure of the relationships between persistent objects. Objects have pointers to other objects with associated transition probabilities. Every object belongs to exactly one page. From the current position of the client navigation we compute the access probability of all adjacent pages. We do this by evaluating all paths from the current object to objects in the adjacent pages. The object relationships are modelled using a Discrete-Time Markov Chain (DTMC) and a method called *hitting times* is used to compute the page access probability. If the probability of a page is higher than a threshold defined by cost/benefit parameters then the page is a candidate for prefetching. To determine the prefetching threshold we consider various cost parameters to compare the benefit of a correct prefetch with the cost of an incorrect prefetch.

In Section 2 we will give an introduction to the model definitions, explain the decision process for prefetching and the computation of a page probability. In Section 3 we present the cost parameters for a prefetch operation in a client/server architecture. The results from the implementation of the prefetching technique and simulation results are presented in Section 4. Finally, in Section 5 we conclude our work and give an outlook to future work.

2 Prediction Model

2.1 Model Definitions

In a persistent object store objects have relationships with other objects. Let O denote the set of objects in the store and let $R \subseteq O \times [0,1] \times O$ denote the set of object relationships between objects, along with a weight for each such relationship. The weight denotes the probability that we traverse from one object to another. Further, let $o_i \in O$ be the current object that the database client is processing. Let $o_j \in O$ and $x \in [0,1]$. If $(o_i, x, o_j) \in R$ then we say that $\mathrm{o}_i \xrightarrow{x} \mathrm{o}_j$ denoting the probability x that we go from o_i to o_j.

Let PG be the set of database pages and $pg_i \in PG$ the page that contains the object o_i, i.e. the page on which the client is currently processing. A page pg_j is said not to be resident in the client buffer pool BP, with $BP \subseteq PG$, if $pg_j \in PG \backslash BP$. The condition for an object relationship is $\forall o_i \in O, \sum\{x : \exists o_j \in O : (o_i, x, o_j) \in R\} = 1$, i.e. the sum of the probabilities associated with the emerging arcs from o_i must add up to 1. For the case when the traversal terminates at an object we introduce a self-loop for an object o_i such that $o_i \xrightarrow{x} o_i$ denotes the probability x that the traversal will be terminated at object o_i.

2.2 Prefetch Decision Model

In our previous work ([10], [11]) we used a *Prefetch Object Distance* (POD_{pf}) to start the prefetch operation d object processing units (steps) before application access. POD_{pf} is computed by dividing the cost of a page fetch[1] by the cost of object processing[2]. We denote a *Prefetch Start Object* (*PSO*) as an object that when encountered will start the prefetch operation. The advantage of this approach is that the savings in elapsed time are high but the probability that the traversal will be from the *PSO* to an object in a non-resident page could be low. Prefetching a page less than d objects before access has certainly a lower saving but the probability that we traverse from the current object to an object in a non-resident page could be higher.

In this work we introduce a *Prefetch Distance Range* (*PDR*) with a *minimal POD* (POD_{min}) and *maximal POD* (POD_{max}) in which we would identify a *PSO*. POD_{min} is defined to be the break-even-point when the prefetch benefit starts to outweigh the prefetch costs. POD_{max} has a higher value than POD_{pf} because its value takes into account possible delays of the page fetch. A prefetch started earlier than POD_{max} would result in the same benefit.

First we explain when we prefetch pages and in the next sections we describe the components that influence this decision process. Suppose $i \in O$ is the current object and $\alpha \in PG \backslash BP$ is a page then we will denote by $\mathbb{P}_{i,\alpha,}$ the probability that starting in i, we hit[3] page α (definition in Section 2.3). Also let CIP be the Cost of an Incorrect Prefetch (definition in Section 3.1) and $\mathrm{BCP}_{(d)}$ the Benefit of a Correct Prefetch (definition in Section 3.2) which is dependent on the POD parameter d. The decision whether to prefetch a page is made by the following constraint:

$$\mathbb{P}_{i,\alpha} > \frac{CIP}{BCP_{(d)}+CIP} \tag{1}$$

To explain this equation it was derived from:

$$\mathbb{P}_{i,\alpha} \cdot BCP_{(d)} > (1-\mathbb{P}_{i,\alpha}) \cdot CIP \tag{2}$$

If the probability that the page will be accessed multiplied by the benefit of the page is greater than the probability that the page is not accessed multiplied by the cost of an incorrect prefetch then we will prefetch the page.

Let O_{NR} ($O_{NR} \subseteq O$ and $O_{NR} \nsubseteq PG \backslash BP$) be the set of objects which are not in the buffer pool where there are paths from the current page pg_i. For the purpose of our model, for every element o_k ($o_k \in O_{NR}$) we check constraint (1) for every object o_i which has path to o_k in the distance range ($POD_{Min} \leq d \leq POD_{max}$). If constraint (1) is fulfilled then we define object o_i as a *PSO*. There may be a number of paths from o_i to o_k that is exponential in d. However, as we shall see, we do not have to examine each path individually.

Let O_{PDR} ($O_{PDR} \subseteq O$) be the objects in the *PDR* which have a path to a page α. Then we compute the *heat* of an object $o_i \in O_{PDR}$ to access page α by:

$$heat(o_i,\alpha) = \mathbb{P}_{i,\alpha} \cdot BCP_{(d)} - (1-\mathbb{P}_{i,\alpha}) \cdot CIP \tag{3}$$

For objects with an in-degree of 1 we compare the $heat(o_i,\alpha)$ value of a referencing object with the referenced object and identify only the object with the higher value as a *PSO*. This process could involve a comparison over multiple objects. The identified *PSO* has then the theoretical optimal distance to prefetch a page (POD_{opt}).

After the analysing process we decide whether to prefetch from the persistent store. If the estimated benefits outweigh the fixed costs (thread and socket creation) then we will use prefetching.

2.3 Computation of the Page Access Probability

A DTMC is a stochastic process which is the simplest generalisation of a sequence of independent random variables. A Markov Chain is a random sequence in which the dependency of the successive events goes back only one unit in

[1] We distinguish between a page fetch from server memory and one from the server's disk.

[2] Object processing includes a fixed system cost and variable user processing cost.

[3] To hit a page means the traversal from a current object to an object in another page.

time.[4] In other words, the future probabilistic behaviour of the process depends only on the present state of the process and is not influenced by its past history.

Let $(X_n)_{n\geq 0}$ be a DTMC with transition matrix P. We associate one state in the DTMC with one object and the current state is associated with the current object. The hitting time of a page α is the random variable $H^\alpha : \Omega \rightarrow \{0,1,2,\ldots\} \cup \{\infty\}$ given by

$$H^\alpha(\omega) = \inf\{n \geq 0 : X_n(\omega) \in \alpha\} \tag{4}$$

$H^\alpha(\omega)$ is one state of α to be hit at time ω. The probability starting in object i that $(X_n)_{n\geq 0}$ ever hits α is then

$$h_i^\alpha = \mathbb{P}_i(H^\alpha < \infty). \tag{5}$$

The mean time taken for $(X_n)_{n\geq 0}$ to reach α is either n steps or ∞ steps and given by

$$k_i^\alpha = E_i(H^\alpha) = \sum_{n<\infty} n\,\mathbb{P}(H^\alpha = n) + \infty\,\mathbb{P}(H^\alpha = \infty) \tag{6}$$

The mean hitting time and the hitting probability can be calculated by linear equations. With Theorem 1 we are able to establish the equations for the hitting probability.

Theorem 1 *The vector of hitting probabilities* $h^\alpha = (h_i^\alpha : i \in O)$ *is the minimal non-negative solution to the system of linear equations*

$$\left\{ \begin{array}{lcll} h_i^\alpha & = & 1 & \textit{for } i \in \alpha \\ h_i^\alpha & = & \sum_{j\in O} p_{ij} h_j^\alpha & \textit{for } i \notin \alpha \end{array} \right. \tag{7}$$

The mean hitting time can also be calculated by linear equations:

Theorem 2 *The vector of mean hitting times* $k^\alpha = (k^\alpha : i \in O)$ *is the minimal non-negative solution to the system of linear equations*

$$\left\{ \begin{array}{lcll} k_i^\alpha & = & 0 & \textit{for } i \in \alpha \\ k_i^\alpha & = & 1 + \sum_{j\notin\alpha} p_{ij} k_j^\alpha & \textit{for } i \notin \alpha \end{array} \right. \tag{8}$$

The proof for both theorems can be found in [14]. We solve these equations online by an iterative method called *conjugate gradient* and off-line by the *LU decomposition algorithm* (for more implementation details see section 4). In addition, we define the two following rules describing the adaption to our environment:

Rule 1: Let λ be the set of states corresponding to the objects in O that have a path to an object in a page α. For the setting of the equations to calculate the hitting probability (according to Theorem 1) and the mean hitting time (according to Theorem 2) we only consider states that are elements of λ ($o_i \in \lambda$).

Rule 2: To calculate the mean time that we hit a page α we have to consider only transitions from states in λ to states in λ. If the condition $\forall o_i \in O, \sum\{x : \exists o_j \in O : (o_i, x, o_j) \in R\} = 1$ is not fulfilled anymore because a state is not in λ then we have to recalculate the probability transitions. The new probability values for x_i' are computed by a method called *re-normalisation*:

$$x_i' = \frac{x_i}{\sum_{j=1}^m x_j} \tag{9}$$

where we only consider transition probabilities x_j to the objects corresponding to the states in λ.

Example re-normalisation: Suppose we have $o_t, o_u, o_v, o_w \in O$ and $x_1, x_2, x_3 \in [0,1]$ with the transitions $o_t \xrightarrow{x_1} o_u$, $o_t \xrightarrow{x_2} o_v$ and $o_t \xrightarrow{x_3} o_w$. Let $o_t, o_u, o_v \in \lambda$ and $o_w \notin \lambda$. Then the values for x_1' and x_2' are computed by $x_1' = x_1/(x_1+x_2)$ and for $x_2' = x_2/(x_1+x_2)$.

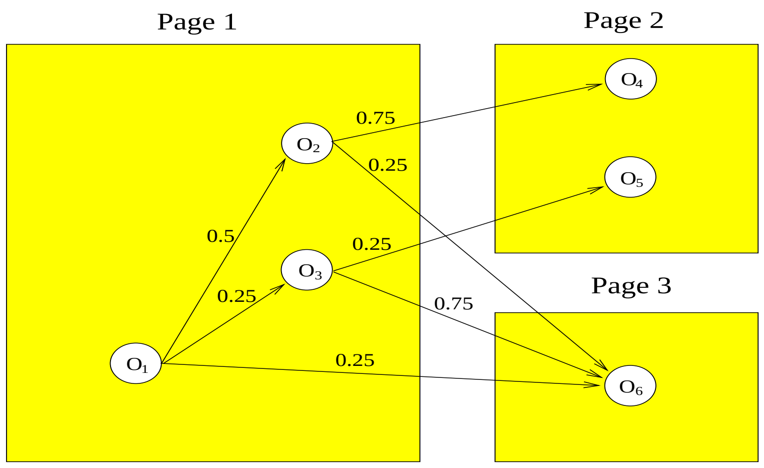

Figure 2: Probability graph

Example hitting equations: Figure 2 depicts a simple example of objects that are resident in a page with references to other objects. The probability to hit page 2 starting in object o_1 is computed as follows:

$$\begin{aligned} h_5 &= 1 \\ h_4 &= 1 \\ h_3 &= 0.25h_5 \\ h_2 &= 0.75h_4 \\ h_1 &= 0.5h_2 + 0.25h_3 \end{aligned}$$

As a result, the access probability of page 2 is 0.4375 and of page 3 is 0.5625. The mean time to access page 2, starting in o_1, is then computed by the following equations:

$$\begin{aligned} k_5 &= 0 \\ k_4 &= 0 \\ k_3 &= 1 + 1k_5 \\ k_2 &= 1 + 1k_4 \\ k_1 &= 1 + \frac{2}{3}k_2 + \frac{1}{3}k_3 \end{aligned}$$

Starting from o_1 the mean time to access page 2 is 2 and to page 3 is 1.75. The transition values for equations k_1 to k_3 are obtained according to Rule 2.

3 Cost-Benefit Model

3.1 Cost of an Incorrect Prefetch Request

This cost describes the additional elapsed time for the application due to an incorrect prefetch. Every incorrect prefetch imposes a higher synchronisation cost for the DemandThread to access global data. Table 1 shows the cost parameters which influence the cost of an incorrect prefetch. CIP is then computed by:

$$CIP = C_{CS} + C_M + C_R \tag{10}$$

[4]Time in the context of a DTMC means simply a number of steps.

Parameter	Description
C_{CP}	Cost for client processing which includes Auditing, Buffer Management (except C_R and C_{RB}), IO, Concurrency Control, Network Processing, Memory Management.
C_{CS}	Increased cost of context switches due to prefetch threads. Let $C_{CS(1)}$ be the context switch cost for one prefetch thread and let $\sigma_{(p)}$ be the scale-factor dependent on the number of prefetch threads p. $$C_{CS} = C_{CS(1)} \cdot \sigma_{(p)}$$
C_M	Additional waiting time and processing cost for the DemandThread to acquire and release mutexes. Let $C_{CM(1)}$ be the mutex cost for one prefetch thread. $$C_{CM} = C_{CM(1)} \cdot \sigma_{(p)}$$
C_{PR}	Cost for using prefetch information (not for solving the hitting times equations).
C_{PW}	Cost for waiting until a page request arrives at the client. Let $C_{PW(1)}$ be the waiting cost for a request to the server with 1 client and $\delta_{(c)}$ a scale-factor for the delay of a page fetch dependent on the number of clients c at the server. $$C_{PW} = C_{PW(1)} \cdot \delta_{(c)}$$
C_R	Cost for the replacement of a page with an incorrect prefetched page. The replaced page may be accessed again. $$C_R = \begin{cases} C_{PW} + C_{CP} & \text{if page is accessed again} \\ 0 & \text{otherwise} \end{cases}$$
C_{RB}	The cost for the replacement of a page with a correct prefetched page. The replaced page may be accessed before the prefetched page. $$C_{RB} = \begin{cases} C_{PW} + C_{CP} & \text{if replaced page is accessed before prefetched page} \\ 0 & \text{otherwise} \end{cases}$$
C_S	Cost for the DemandThread to wait on a semaphore (only when the DemandThread stalls for the prefetched page).
B_P	Let C_O be the cost of processing one object; recall that d is the prefetch distance parameter. $$B_P = \begin{cases} C_{PW} + C_{CP} & \text{if prefetched page is resident on access} \\ C_O \cdot d & \text{otherwise} \end{cases}$$

Table 1: Cost/Benefit parameters

Table 2: Computer performance specification

Parameter	Server	Client
SPARCstation	20 Model 502	10 Model 514
Main Memory	512 MB	224 MB
Virtual Memory	491 MB	657 MB
Number of CPUs	2	4
Cycle speed	50 MHz	50 MHz

Table 3: Disk controller performance

Parameter	Disk controller
External Transfer Rate	20 Mbytes/sec
Average Seek (Read/Write)	9/10.5 msec
Average Latency	5.54 msec

3.2 Benefit of a Correct Prefetch Request

The maximum saving for a prefetch is only achieved when the prefetched page arrives at the client before application access. The benefit *BCP* depends on the amount of savings minus the prefetch costs:

$$BCP = B_P - C_{CS} - C_M - C_{PR} - C_{RB} - C_S \tag{11}$$

3.3 Cost and Benefit of a Multiple-Page-Request

If we predict multiple pages to prefetch according to constraint (1) we could demand them by a single request from the server. The server would read the pages from disk and send them back to the client either (a) separately when time constraints are tight or (b) in a batch if time is not a problem.

A Multiple-Page-Request has the advantage that the processing cost on the client and the server is lower than two single requests (which reduces C_{CP} and C_{PW}) because some functions have to be executed only once. It also reduces the network costs (which affects C_{PW}). The costs of thread management (C_M and C_{CS}) are also lower because multiple pages are requested by just one thread.

4 Performance Analysis

4.1 Environment

We incorporated prefetch threads into the client ESM. Each prefetch thread has an associated socket to communicate with the server. The server serves each request sequentially. A separate thread is used to solve the hitting times equations at run time. A more detailed description of the prefetching architecture can be found in [11].

In Table 2 we give a specification of the computers used in our experiments. The *Sun Fast Ethernet* network is running at 100 Mb/sec. The performance of the disk controller is presented in Table 3.

4.2 Implementation Results

In this Section we present the results that we obtained from timing ESM with one client connected to the server. Dividing the cost of a page fetch from disk (7943 μs[5]) by the cost of processing one object (799 μs) results in a value of 10 for POD_{pf}.

In Figure 3 we show the amount of savings that can be achieved by one prefetch request dependent on the prefetch distance. A prefetch is already successful with a distance of 1 (POD_{min}) and the maximum improvement is achieved at a distance of 11 objects (POD_{max}). From these empirical results we developed a Benefit formula (12) which computes

[5]This is an optimistic value. We assume disk pages to be stored in clusters which reduces the average seek time.

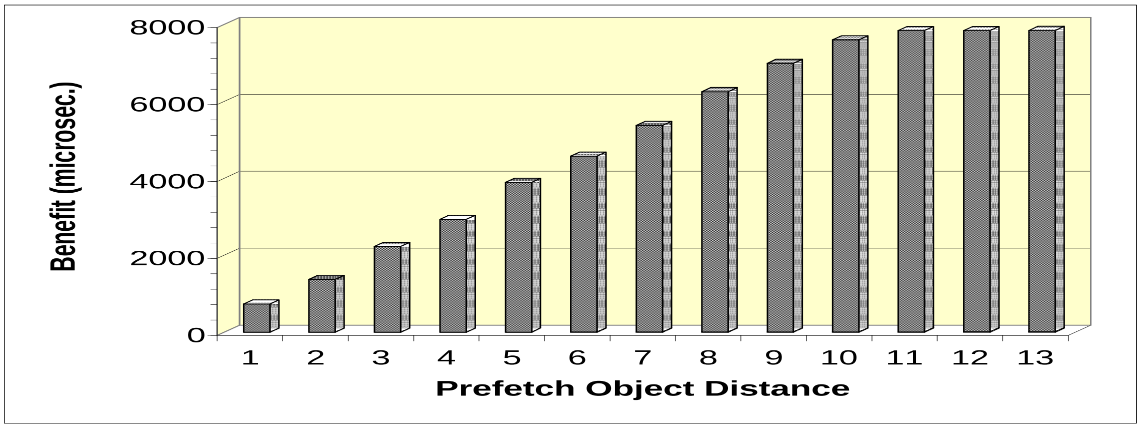

Figure 3: Savings of one prefetch operation

the amount of savings of a prefetch given the distance. We used the least squares method to find the line of best fit relating distance and benefit which results in:

$$benefit(d) = -782\mu s \cdot d + 109\mu s \tag{12}$$

The variable benefit in formula 12 is almost as high as the cost of object processing. The benefit values are highly influenced by the settings of the object processing cost and the page fetch cost. We also measured the cost of an incorrect prefetch which is about 1573 μs higher than a Demand fetch.

For an efficient implementation to solve the hitting times equations we compared 5 algorithms. We used two direct methods - Gauss-Jordan (GJ) and LU-decomposition (LU), which compute an exact solution; and three iterative methods - Successive Over-Relaxation (SOR), Gauss-Seidel (GS) and Conjugate Gradient (CG). A detailed description of these algorithms can be found in [16].

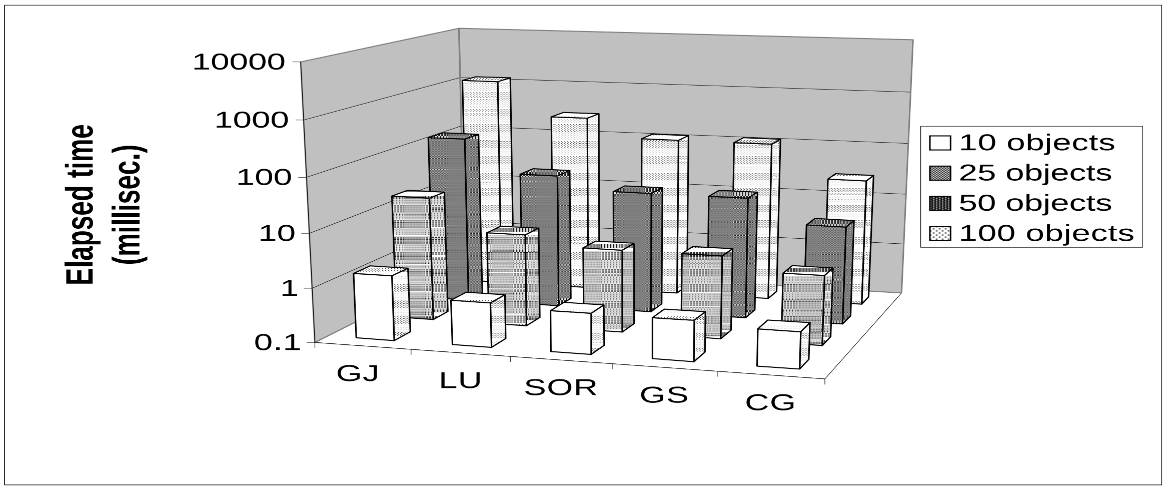

Figure 4: Computation time to solve linear equations

Figure 4 shows the elapsed times of these algorithms to compute a matrix with *n* objects. LU is the faster direct method and CG is the fastest iterative method. Therefore we use the CG method in a separate thread on-line and the LU method off-line. In our application the number of objects in the matrix is determined by the number of objects in the equivalence class, i.e. all the objects that have a path from the current page to the page to be computed. If

Table 4: Simulation parameters

Parameter		Setting (μs)
Object processing		799
1 Page Prefetch :	Page fetch	7943
	Incorrect prefetch	1573
	Variable benefit	-782
	Fixed cost benefit	109
	POD_{min} / POD_{max}	1/11
2 Page Prefetch :	Page fetch	9804
	Incorrect prefetch	2360
	Variable benefit	-782
	Fixed cost benefit	163
	POD_{min} / POD_{max}	1/13

the thread for the computation gets too busy we have to continue the computation after the transaction. Please note that all the computation is done automatically and the database administrator has only a few adjustments to make, e.g. determine the user object processing cost.

4.3 Simulation Results

In the simulation we want to present the benefits of prefetching which are very dependent on the object relationship structures and the probability transitions. For example a linked list is a very easy candidate for prediction whereas objects with fan-out of 10 referenced objects are very difficult to predict. We used the discrete process based simulation package C++Sim [13] and presented constant cost factors in Table 4. We assume that the buffer space is infinite.

To test our prefetching algorithm we created two different benchmarks. In both benchmarks every branch object has an out-degree of 2. In the first benchmark the distance from the entry object in the page to an object in another page is 10. In this distance there are 4 branch objects and 6 non-branch objects which makes a total of 62 objects in a page and 16 references to different pages. Every page has the same structure and we access 1000 pages. In Figure 5 we show the result of this test with 3 applications: Demand (without any prefetching), a 1 Page Prefetch (1PP) and a 2 Page Prefetch (2PP) technique. The applications with a + sign also consider the *heat* parameter to start the prefetch at the best possible object.

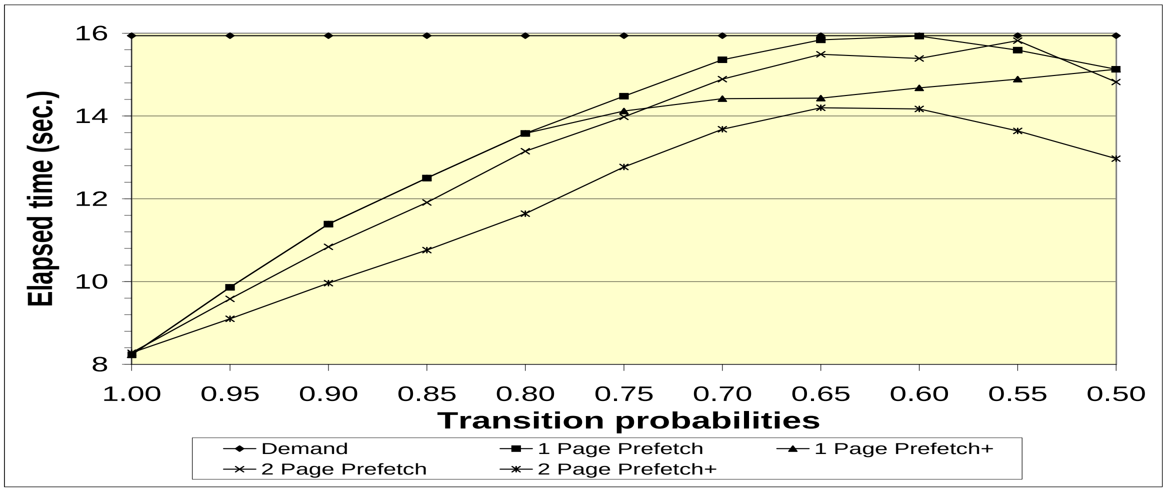

Figure 5: Test result of Benchmark 1

We varied the transition probabilities of the branch objects from 1.0 down to 0.5. The navigation through the object graph was controlled by a *draw-operator*[6]. The Demand application has constant values and is independent of the transition probabilities. 1PP and 1PP+ perform equally well until the transition probability of 0.8 after which 1PP+ is better because of a later but more accurate prefetch. 2PP+ is always better than 2PP and the 1 page prefetching techniques because it has a higher hit ratio. At the transition probabilities of 0.65 and 0.6 all prefetching applications suffer from a bad hit ratio imposed by difficult page predictions. Figure 5 shows the general advantage of our technique: If the transition probabilities allow prefetching it can reduce elapsed time drastically but if not it won't decrease performance.

In the second benchmark the distance from an entry object to an object in another page is now 16, with 3 branch objects in between which results in 8 references to other pages. The major difference to benchmark 1 is that the last branch object in the high probability path has a 0.5 probability to both objects. This results in 2 referenced pages with high probabilities. Figure 6 shows the result of this test. In general the prefetching application show a better performance than in the previous benchmark because there are only 8 adjacent pages and these pages are easier to predict. 1PP and 1PP+ show the same elapsed time for all transition probabilities. Both applications do not perform well at the probability of 1.0 because the last branch object with 0.5 probability imposes a high incorrect prefetching time for both. 2PP+ shows its superiority again especially with lower probabilities because of a more accurate prefetch. Figure 6 also shows that it is beneficial to prefetch given all possible probabilities.

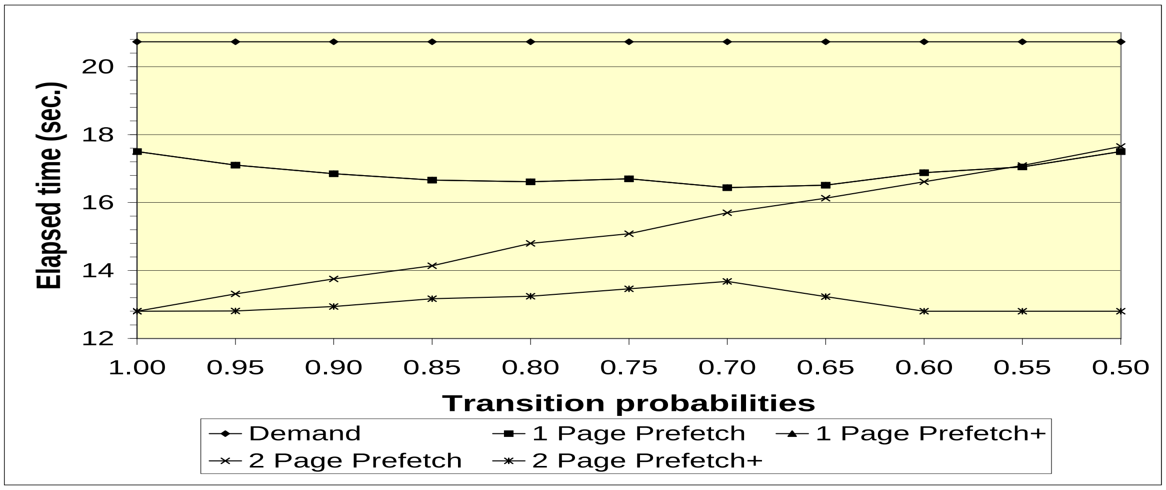

Figure 6: Test result of Benchmark 2

5 Conclusions and Future Work

We presented a prefetching technique to fetch pages before access to avoid expensive page faults. The page access probability is computed by a method called *hitting times*. The database client navigates through the object graph. From the current position of the client navigation in a page we compute the probability of every adjacent page. If a page is not resident and the page probability is higher than specific threshold, determined by cost and benefit parameters, then the page is a candidate for prefetching. The result of our simulation is that prefetching can improve performance if object access is reasonably predictable.

We are currently continuing our simulations to evaluate our technique under different object clustering structures. We also want to integrate a buffer replacement strategy. The effect of the number of updates is another open research problem.

Acknowledgments

We would like to thank Jane Hillston, Peter Thanisch and Nigel Thomas from our department for all their help. We also thank the anonymous referees for their valuable comments.

[6]Given a probability value it decides to continue navigation with reference 1 or 2.

References

[1] A. Bestavros. Using Speculation to Reduce Server Load and Service Time on the WWW. In *Proc. of the 1995 ACM CIKM Int. Conf. on Information and Knowledge Management.*, pages 403–410. ACM, December 1995.

[2] M.J. Carey, D.J. DeWitt, G. Graefe, D.M. Haight, J.E. Richardson, D.T. Schuh, E.J. Shekita, and S.L. Vandenberg. The EXODUS Extensible DBMS Project: An Overview. In S. Zdonik and D. Maier, editors, *Readings in Object-Oriented Database Systems*, pages 474–499. Morgan Kaufmann, 1990.

[3] E.E. Chang and R.H. Katz. Exploiting Inheritance and Structure Semantics for Effective Clustering and Buffering in an Object-Oriented DBMS. In *Proc. of the ACM SIGMOD Int. Conf. on the Management of Data*, pages 348–357, Portland, Oregon, June 1989.

[4] J.R. Cheng and A.R. Hurson. On the Performance Issues of Object-Based Buffering. In *Proc. First Int. Conf. on Parallel and Distributed Information System*, pages 30–37, Miami Beach, Florida, December 1991.

[5] M.S. Day. *Client Cache Management in a Distributed Object Database*. PhD thesis, Massachusetts Institute of Technology, Laboratory for Computer Science, 1995.

[6] C.A. Gerlhof and A. Kemper. Prefetch Support Relations in Object Bases. In *Proc. of the Sixth Int. Workshop on Persistent Object Systems*, pages 115–126, Tarascon, Provence, France, September 1994.

[7] J. Griffioen and R. Appleton. Improving File System Performance via Predictive Caching. In *Parallel and Distributed Computing Systems*, pages 165–170, Orlando, Florida, September 1995.

[8] K.S. Grimsrud, J.K. Archibald, and B.E. Nelson. Multiple Prefetch Adaptive Disk Caching. *IEEE Knowledge and Data Engineering*, 5(1):88–103, February 1993.

[9] T. Keller, G. Graefe, and D. Maier. Efficient Assembly of Complex Objects. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 148–157, Denver, USA, May 1991.

[10] N. Knafla. A Prefetching Technique for Object-Oriented Databases. In *Advances in Databases, 15th British National Conf. on Databases*, Lecture Notes in Computer Science, pages 154–168, London, United Kingdom, July 1997. Springer-Verlag.

[11] N. Knafla. Speed Up Your Database Client with Adaptable Multithreaded Prefetching. In *Proc. of the Sixth IEEE Int. Symp. on High Performance Distributed Computing*, pages 102–111, Portland, Oregon, August 1997. IEEE Computer Society Press.

[12] A. Kraiss and G. Weikum. Vertical Data Migration in Large Near-Line Document Archives Based on Markov-Chain Predictions. In *Proc. of the 23rd Int. Conf. on Very Large Databases*, pages 246–255, Athens, Greece, August 1997.

[13] M.C. Little and D.L. McCue. Construction and Use of a Simulation Package in C++. Technical Report 437, Department of Computing Science, University of Newcastle, UK, July 1993.

[14] J.R. Norris. *Markov Chains*. Cambridge series on statistical and probabilistic mathematics. Cambridge Uni Press, 1997.

[15] M. Palmer and S.B. Zdonik. Fido: A Cache That Learns to Fetch. In *Proc. of the 17th Int. Conf. on Very Large Data Bases*, pages 255–264, Barcelona, Spain, September 1991.

[16] W. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton, 1994.