Computer Systems Group

Predicting Future Page Access by Analysing Object Relationships

by

Nils Knafla
E-mail: nk@dcs.ed.ac.uk

CSG Report Series **ECS–CSG–35–97**

Computer Systems Group **December 1997**

Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ

Predicting Future Page Access by Analysing Object Relationships

Nils Knafla
E-mail: nk@dcs.ed.ac.uk

Technical Report ECS–CSG–35–97
Department of Computer Science
University of Edinburgh

December 5, 1997

Abstract

In this report we present a new approach to predicting page access probability by considering the structure of the relationships between objects. Database objects and their relationships to other objects are modelled by a *discrete-time Markov Chain*. We present two approaches to compute the page access probability: (a) Using Hitting Times and Absorption Probabilities and (b) Using the Chapman-Kolmogorov Equations. 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.

Keywords: prefetching, object-oriented databases, discrete-time Markov Chains, hitting times and absorption probabilities, EXODUS, high performance object stores, persistence, client/server computing, disk storage management

1 Introduction

1.1 Performance Bottlenecks of Object-Oriented Databases

In most commercial object-oriented database management systems (OODBMS) the database client requests pages from the server. The server retrieves the page from its local disk and sends it to the client. The client stores the page in its buffer pool and the database application can work with the objects in the page. This page fetch is an expensive operation. In fig. 1 we give an example to show the most expensive elements of such a page fetch (total cost about 8748 μs). The results were obtained by timing the EXODUS storage manager (ESM) [2]; details about our computing environment can be found in section 4.2.[1]

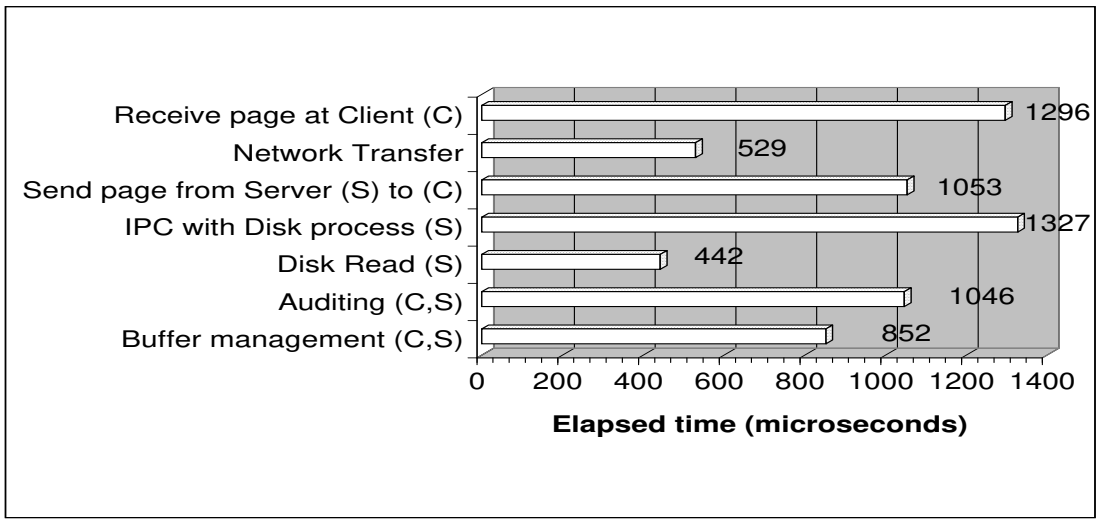

Figure 1: Expensive components of a page fetch

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 processor speed. In all our tests we used a 50 MHZ machine and therefore these costs could be reduced using up-to-date processors by a factor of 6. Then the network transfer and 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 fig. 1 because we read all pages sequentially from disk.

[1] This test was made with just one client at the server and the machine workload was low.

1.2 Approaches to Improve Database Performance

To reduce the high cost of a page fetch and disk latencies many researchers developed prefetching techniques to overcome this problem. In OODBMSs Chang and Katz [3] predict future accesses by hints from the data semantics in terms of inheritance and structural relationships. Cheng and Hurson [4] extended this work by adding multiple hints, a prefetching depth and physical storage considerations. A complex assembly operator to load component objects recursively in advance was introduced by Keller et al. [9]. In the Fido system [15], the prefetching technique employs an associative memory to recognize access patterns within a context over time. In training mode, object access information is gathered and stored with a *nearest-neighbor* associative memory. In prediction mode, this information is used to recognize previously encountered situations. Gerlhof [6] prefetches the precomputed page answer of an operation, i.e. the identifiers of all pages that were accessed during the execution of an operation. In Thor [5] each fetch request from the client causes the server to select a prefetch group containing the object requested and possibly some other objects.

In some research work the future access was predicted by probability models which have their values 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 World Wide Web [1] Bestavros speculated about document accesses based upon their document interdependency probabilities. Grimsrud et al. [8] keep 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 [13]. 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 objects. Database objects and their relationships to other objects are modelled by a *discrete-time Markov Chain*. We present two approaches to compute the page access probability:

(a) Using Hitting Times and Absorption Probabilities

(b) Using the Chapman-Kolmogorov Equations.

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.

The model for predicting the page access is explained in section 2. In this section 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. Primary results about the cost

parameters are showed in section 4. Finally, in section 5 we conclude our work and give an outlook to future work.

2 The Prediction Model

2.1 Model Definitions

In an object-oriented database system objects have relationships with other objects. Let O denote the set of objects in the database 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 $o_i \xrightarrow{x} o_j$ 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 an artificial object $o_{halt} \in O$ such that $o_i \xrightarrow{x} o_{halt}$ 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], [12]) we used a minimal *Prefetch Object Distance* (POD) to start the prefetch operation d object processing units (steps) before application access. 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.

Suppose $i \in O$ is the current object and $\alpha \in PG \backslash BF$ is a page then we will denote by $\mathbb{P}_{i,\alpha}$, the probability that starting in i, we hit[2] 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). The decision whether to prefetch a page is made by the following constraint:

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

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

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. $\mathrm{BCP}_{(d)}$ is strongly dependent on the POD parameter d. The theoretical optimal POD is computed by dividing the cost of a page fetch by the cost of object processing (see [10], [11]). The constraint (1) is the minimum condition that has to be fulfilled. In addition we use a probability threshold parameter.

Let O_{NR} ($O_{NR} \subseteq O$ and $O_{NR} \nsubseteq PG \backslash BP$) be the set of objects not in the buffer pool to which there are direct relationships from a page p_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 a distance $\leq d$. 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. If constraint (1) is fulfilled then we define object o_i as a PSO. This whole identification process is made off-line and only the PSO information is used at run time. After the analysing process we decide whether to prefetch from the database. If the estimated benefits outweigh the fixed costs (thread and socket creation) and variable costs (e.g. prediction costs for phases with no prefetching) then we will do prefetching.

2.3 Computation of the Page Access Probability

To compute the probability that a page will be hit we would like to implement two methods and compare their computational overhead and accuracy. In section 2.3.1 we explain the use of hitting times and absorption probabilities and in section 2.3.2 we explain how we could use the Chapman-Kolmogorov equations.

2.3.1 Using Hitting Times and Absorption Probabilities

A discrete-time Markov chain 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 time. 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 discrete-time Markov chain with transition matrix P. The hitting time of a page α is the random variable $H^\alpha : \Omega \to \{0, 1, 2, ...\} \cup \{\infty\}$ given by

$$H^\alpha(\omega) = \inf\{n \geq 0 : X_n(\omega) \in \alpha\} \tag{3}$$

where we agree that the infimum of the empty set $\emptyset$ is ∞. $H^\alpha(\omega)$ is one state (i.e. object) of α to be hitted at time ω. The probability starting in object i that $(X_n)_{n\geq 0}$ ever hits α is then

$$h_i^\alpha = \mathbb{P}_i(H^\alpha < \infty). \tag{4}$$

When α is a *closed class*[3], h_i^α is called the *absorption probability* but in our environment we compute in most cases the probability to transient states. The mean time taken for $(X_n)_{n\geq 0}$ to reach α is given by

$$k_i^\alpha = E_i(H^\alpha) = \sum_{n<\infty} n\,\mathbb{P}(H^\alpha = n) + \infty\,\mathbb{P}(H^\alpha = \infty) \tag{5}$$

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*

$$\begin{cases} h_i^\alpha &= 1 & \text{for } i \in \alpha \\ h_i^\alpha &= \sum_{j\in O}\ p_{ij}h_j^\alpha & \text{for } i \notin \alpha \end{cases} \tag{6}$$

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*

$$\begin{cases} k_i^\alpha &= 0 & \text{for } i \in \alpha \\ k_i^\alpha &= 1 + \sum_{j\notin\alpha}\ p_{ij}k_j^\alpha & \text{for } i \notin \alpha \end{cases} \tag{7}$$

The proof for both theorems can be found in [14]. We solve these equations either by Gaussian elimination method Gauss-Jordan or Gauss-Seidel dependent on the number of objects. In addition we defined the two following rules describing the adaption to our environment:

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

[3]We say that a class C is *closed* if $i \in C, i \to j$ imply $j \in C$. Thus a closed class is one from which there is no escape.

Rule 2: To calculate the mean time that we get absorbed in 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 recalculated the probability transitions. The new probability a values for x'_i is computed by

$$x'_i = \frac{x_i}{\sum_{j=1}^{m} x_j} \tag{8}$$

where we only consider transitions x_j to objects in λ.

Example: 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 $x'_2 = x_2/(x_1 + x_2)$.

Fig. 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:

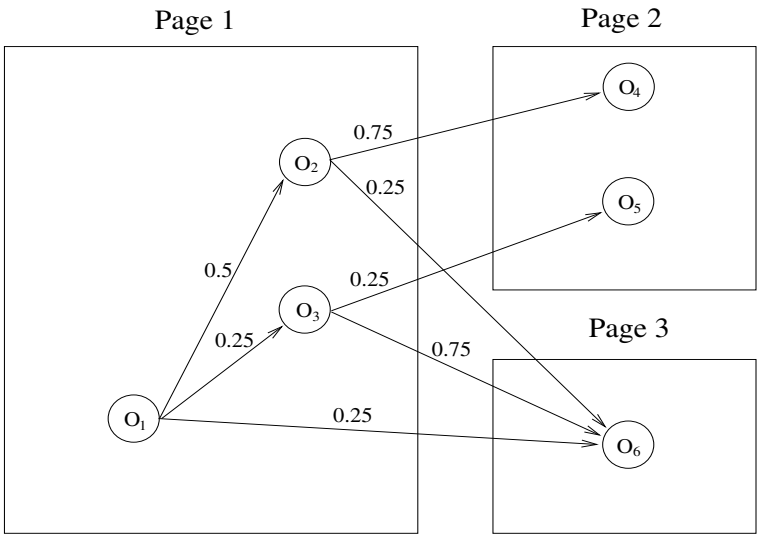

Figure 2: Probability graph

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

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

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

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

2.3.2 Using the Chapman-Kolmogorov Equations

The page access probability can also be computed with the Chapman-Kolmogorov equations. Let i be the current object and j ($j \in O$) an object in another page then the probability that we will be in object j after $n + m$ steps is computed by the equations:

$$\mathbb{P}_{ij}^{n+m} = \sum_{k=0}^{\infty} \mathbb{P}_{ik}^{n} \, \mathbb{P}_{kj}^{m} \text{ for all } n, m \geq 0, \text{all } i, j \tag{9}$$

These equations can be solved by matrix multiplications. To compute the probability to hit the page α we add up all probability values for objects in α

$$\mathbb{P}_{i}^{\alpha} = \sum_{n=0}^{\infty} \sum_{o_j \in \alpha} \mathbb{P}_{ij}^{n} \tag{10}$$

The advantage of using approach 2.3.1 is that we are able to setup the equations with a limited number of objects, whereas in approach 2.3.2 we have to perform a matrix multiplication for all objects in the database. Another advantage of 2.3.1 is that we compute exactly one mean hitting time from a current object to another page.

3 The Cost - Benefit Model

3.1 The Cost of an Incorrect Prefetch Request

Table 1 shows the cost parameters which influence the cost of an incorrect prefetch. CIP is then computed by:

$$CIP = C_{CS} + C_M + C_{PW} + C_R \tag{11}$$

Parameter	Description
C_{CP}	This is the 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. The number of prefetch threads is $\leq$ the number of available processors. Let $C_{CS(1)}$ be the context switch cost for one prefetch thread and let $\sigma_{(s)}$ be the scale-factor dependent on the number of *SupportThreads* s. $$C_{CS} = C_{CS(1)} \cdot \sigma_{(s)}$$
C_{CV}	Cost for the DemandThread to wait on a condition variable (only when the DemandThread stalls for the prefetched page).
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. $$C_{CM} = C_{CM(1)} \cdot \sigma_{(s)}$$
C_{PR}	Cost for predicting the future access.
C_{PW}	The 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	The cost for the replacement of a page with an incorrect prefetched page. The replaced page may be accessed again. $$C_R = \begin{cases} C_P & \text{if page is accessed again} \\ 0 & \text{otherwise} \end{cases}$$
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. $$C_{RB} = \begin{cases} C_P & \text{if replaced page is accessed before prefetched page} \\ 0 & \text{otherwise} \end{cases}$$
B_P	Let C_O 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 description

3.2 The Benefit of a Correct Prefetch Request

The maximum saving for a prefetch can be achieved when the prefetched page arrives at the client before application access but otherwise there will be a lower saving. The benefit BCP is dependent on the amount of savings and the cost that is incurred:

$$BCP = B_P - C_{CS} - C_{CV} - C_M - C_{PR} - C_{RB} \tag{12}$$

3.3 Cost and Benefit with Multiple SupportThreads

A *PrefetchThread* is responsible for predicting future access and for prefetching. A *SupportThread* is only responsible for prefetching pages and it is managed by the PrefetchThread. We use one PrefetchThread and multiple SupportThreads (if necessary). Every additional SupportThread induces a higher value for C_{CS}, C_M, C_{PW} which affects CIP and BCP. To compute the value of C_{CS}, C_M we use the scale factor $\sigma_{(s)}$ which is dependent on the number of SupportThreads s.

3.4 Cost and Benefit of a Multiple-Page-Request

If we predict multiple page to prefetch according to constraint (1) we could demand them by a single request from the server. We would order the requests according to their access probability and time constraints. 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 (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 Implementation

4.1 The Client-Server Architecture

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 more detailed description of the prefetching architecture can be found in [11].

4.2 Performance Parameters

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. Please note that all results are mean values from repeated tests.

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

4.3 Primary Results

In this section we present the first results that we obtained from timing ESM. To compute the POD we divide the cost of a page fetch by the cost of object preparation (806 μs). The cost of a page fetch is dependent on many factors. Its cost is determined by the number of clients connected to the server, the number of prefetch threads at the client and the interference of requests. We measure the cost of a Demand Fetch (DF) and Prefetch Fetch (PF) in four different applications:

1. **Demand:** An application without any prefetching.
2. **PC:** A prefetch application with 100% accuracy, so that there are no Demand requests.
3. **PI:** A prefetch application with 0% accuracy, so that there are Demand and prefetch requests which do not interfere.
4. **PI Int:** A prefetch application with 0% accuracy, so that there are Demand requests and the incorrect prefetch requests which are started at the same time as the Demand request.

Table 4 presents the result of this test. At the time of the test there was no other client connected to the server. The result shows that the number of active threads and interference makes a difference to the page fetch cost.

Prefetching can decrease elapsed time if prediction is accurate but otherwise prefetching can slow down performance drastically. We created a benchmark in which 100 pages were

Table 4: Page fetch cost

Application type:	Demand DF	PC PF	PI DF	PI PF	PI Int DF	PI Int PF
Elapsed time (μs):	7943	7936	8092	8084	9170	9160

accessed sequentially. Fig. 3 presents the result of the benchmark. *Incorrect* is a prefetching application which in addition prefetches 100 pages incorrectly and *Incorrect Int* does the same with the difference that the prefetch is started at the same time as the *Demand* read. The POD applications show the benefit of prefetch dependent on the prefetch object distance ('1','2', etc in POD 1, POD 2 denotes the distance). Both incorrect versions increase performance badly. We have to do more performance measurements to identify the main costs for this decrease. Prefetching in a distance up to 3 also decreases performance. The best result is achieved at a distance 10 (optimal POD) or after.

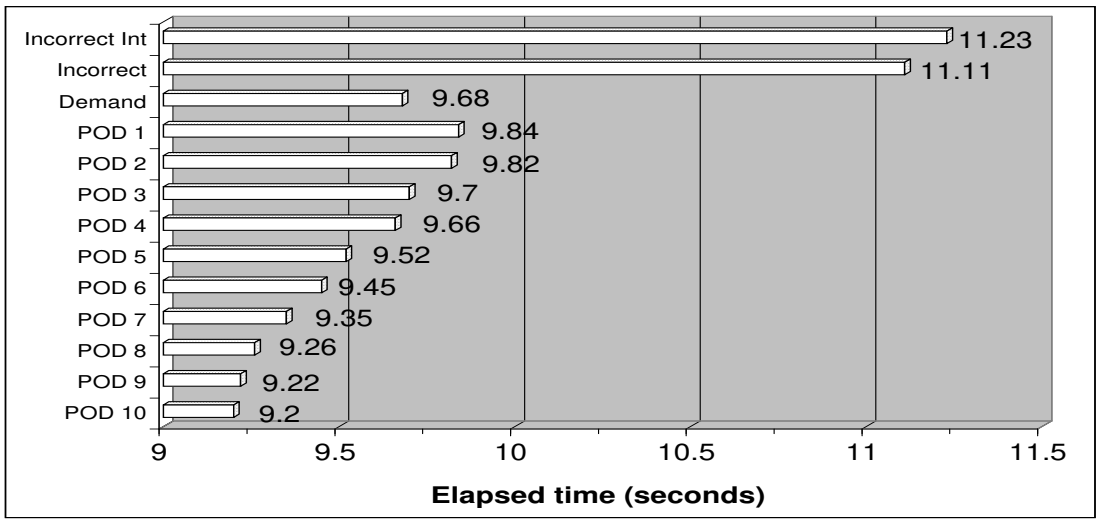

Figure 3: Demand and prefetching applications

To decide whether it is useful to fetch a set of pages by a Multiple-Page-Request we timed a disk read of a set of pages and the cost of sending pages over the network. In the network test we measured the time from sending a server request to the receipt of the page. Table 5 shows the result of the test and proves that it is quicker to send a batch of pages instead of single pages.

In the disk test we read a number of pages with one request. The result of this test (table 6) is different to the network test. Surprisingly, the cost of reading a set of pages is higher than reading these pages by a single request.

Table 5: Cost of sending n K-Bytes via the network

K-Bytes:	1	8	16	24	32
Elapsed time (μs):	1912	2520	3411	4471	5573

Table 6: Cost of reading n 8 KB pages from disk in one request

Number of disk reads:	1	2	3	4
Elapsed time (μs):	238	620	954	1236

5 Conclusions and Future Work

In this report we presented a technique to compute the page access probability. The database client navigates through the object graph. From the current position of the client we compute the probability of every adjacent page from the current. If a page is not resident and the page probability is higher than specific threshold, determined by cost and benefit parameter, then the page is candidate for prefetching.

Database objects and their relationships to other objects are modelled by a *discrete-time Markov Chain*. We presented two approaches to compute the page access probability: (a) Using Hitting Times and Absorption Probabilities and (b) Using the Chapman-Kolmogorov Equations. In the future we will compare the computational overhead of both approaches. We evaluate the prediction accuracy by simulating the object navigation.

Acknowledgments

We would like to thank Graham Clark, Isabel Rojas-Mujica, Peter Thanisch and Nigel Thomas from our department for all their help.

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. Association for Computing Machinery, 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 Conference 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 C. Small, P. Douglas, R. Johnson, P. King, and N. Martin, editors, *Advances in Databases, 15th British National Conference on Databases, BNCOD 15*, 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 International Symposium on High Performance Distributed Computing*, pages 102–111, Portland, Oregon, August 1997. IEEE Computer Society Press.

[12] N. Knafla. An Adaptable Multithreaded Prefetching Technique for Client-Server Object Bases. *Cluster Computing*, 1(1), 1998.

[13] 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.

[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.