Computer Systems Group

Page versus Object Prefetching: A Performance Evaluation

by

Nils Knafla
E-mail: nk@dcs.ed.ac.uk
http://www.dcs.ed.ac.uk/home/nk

CSG Report Series **ECS–CSG–43–98**

Computer Systems Group **August 1998**

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

Page versus Object Prefetching: A Performance Evaluation

Nils Knafla
E-mail: nk@dcs.ed.ac.uk
http://www.dcs.ed.ac.uk/home/nk

Technical Report ECS–CSG–43–98
Department of Computer Science
University of Edinburgh

August 1, 1997

Abstract

In this report we compare the performance of a prefetching page server system with a prefetching object server system. We simulate the object access pattern by assigning transition probabilities to the object relationships. According to the transition probabilities we compute the access probability of pages and objects. We designed several prefetching techniques for a prefetching page server and a prefetching object server. We compared the performance of the prefetching techniques in a simulation.

Keywords: prefetching, object-oriented databases, high performance object stores, persistence, client/server computing, disk storage management, page server, object server

1 Introduction

There have been many efforts to integrate prefetching techniques into databases [6, 14, 23, 7, 9, 17, 11, 15, 16]. Object-oriented database management systems (OODBMSs) and prefetching techniques can be classified according to the unit of transfer between client and server. Some OODBMSs transfer pages to the client (*page server*), for example ObjectStore [18], Objectivity/DB [21], Shore [1]. Other OODBMSs transfer objects or group of objects (*object server*), for example Versant [24], GemStone [8]. Some OODBMSs products, e.g. GemStone, avoid the problem by executing the request on the server and only return the result to the client. The unit of transfer for prefetching techniques is dependent on the system architecture: If the system is a page server the prefetching technique will prefetch one or multiple pages [17, 11, 15, 16] and in case of an object server one or a group of objects [6, 14, 23, 9].

All the previous research was conducted in one of these two types but there is no research, to the best of our knowledge, which accesses a prefetching technique by comparing its performance in a page server and object server implementation. For both systems the important problem to solve is how to avoid the I/O bottleneck. Here we have to distinguish two cases at the server side: disk pages are resident at the server and disk pages are not resident at the server. In theory, if all pages are resident the Object Prefetching Technique (OPT) has the advantage that it can put together all the relevant objects for a prefetch request independent of how these objects are dispersed on pages. If pages are not likely to be resident, a Page Prefetching Technique (PPT) has the advantage that it requests only a few high priority disk pages.

Day [9, 19] made an interesting study in the Thor database in which he compared the performance of a page server system (without prefetching) with an object server system that prefetches groups of objects. The motivation was that the performance of a single-object fetching system is unacceptable [10, 13]. Thor transfers groups of objects from server to client. On receiving a fetch request, a Thor server selects objects to send in response. The group of objects selected is called a *prefetch group*. Thor's dynamic selection of the group contrasts with most distributed object databases which cluster objects statically into pages and transfers pages. The selection of objects consider various techniques that prefetch objects in the transitive closure of the current object. It can use depth-first or breadth-first search and considers whether the object is resident at client or not. The performance evaluation was made with the OO7 benchmark [4] using the dense and sparse traversal. The general result showed that the best technique was *bf-cutoff*: a simple breadth-first traversal that cuts off its exploration upon encountering an object already sent to the client. The page server system performed reasonably well in the dense traversal where the access pattern was similar to the clustering. The problem with this study is that it does not give a fair comparison between an object and page server prefetching system because:

1. It assumes that all pages are resident in the server buffer pool. This is advantageous to the OPT as explained before.

2. It compares an OPT with a demand paging system instead of a PPT.

A general problem with this group prefetching is that if the server is already the bottleneck of the system then the additional selection process of objects costs valuable server execution time. Another problem is the knowledge about the client cache: either it discard this information or it needs to be transfered to the server. The advantage is that the prefetching information is available locally.

The differences between a demand object, page or file servers was studied by DeWitt et al. [10]. The object server transfers only one object between client and server at a time and both client and server can execute methods. The page server transfers 4 KB pages. The file server uses NFS to access files and is not relevant to our study. The general result of this research was that the object-server architecture is relatively insensitive to clustering. It is sensitive to clients buffer size up to a certain point, after which the cost of fetching objects using the RPC mechanism dominates. They conclude that for the object server it is viable to send a group of objects. The page-server architecture is very sensitive to the size of the client's buffer pool and to clustering when traversing or updating complex objects. While the page-server architecture is far superior on sequential scan queries, the object server architecture demonstrates superior performance when the database is poorly clustered or the workstation's buffer pool is very small relative to the size of the database.

The OO7 benchmark [3] compared the performance of four object-oriented databases. This test evaluated three page servers (Objectivity/DB, EXODUS [2] and Ontos [22][1]) and one object server (Versant). The benchmark structure is very complex and offers many test opportunities. In most of the tests the EXODUS storage manager achieved the best result. Versant achieves the best results for queries with a hot cache and inserts.

Carey et al. [5] study the interaction of locking and the database architecture. They presented three page server variants that allow concurrent data sharing at the object level while retaining the performance advantages of shipping pages to the client. The results indicated that a page server is clearly preferable to an object server. Moreover, the adaptive page server with object locking was shown to provide very good performance and outperformed the pure page and object server. Both Carey et al. [5] and DeWitt et al. [10] study the difference in the object/page architecture but not considering prefetching.

The performance evaluation of the three commercial OODBMSs was subject to the study of [12]. Two of the three systems were page server and one object server. In contrast to previous benchmark studies this evaluation was performed with a concrete data warehouse application. One result is that OODBMSs differ substantially in their performance, often more than standard benchmarks had shown [3]. Another result is that numerous tuning possibilities can improve the performance considerably. It came out that the architecture of the system (page vs. object server) is often more important than the object-oriented paradigm itself. In the application, the object server has, besides the superior performance, an additional advantage due to the finer locking granularity.

In this report we compare the performance of a prefetching page server system with a prefetching object server system. In section 2 we give an overview over the client/server architecture. The prefetching algorithms are presented in section 3. Section 3.1 explains

[1]Ontos can be used as an object server, and page server or group server.

the algorithms for the page server and section 3.2 the algorithms for the object server. Section 4 presents the environment for the simulation. All the results of the performance evaluation are given in section 5. Finally, section 6 concludes the work and gives an outlook to future work.

2 Client/Server Architecture

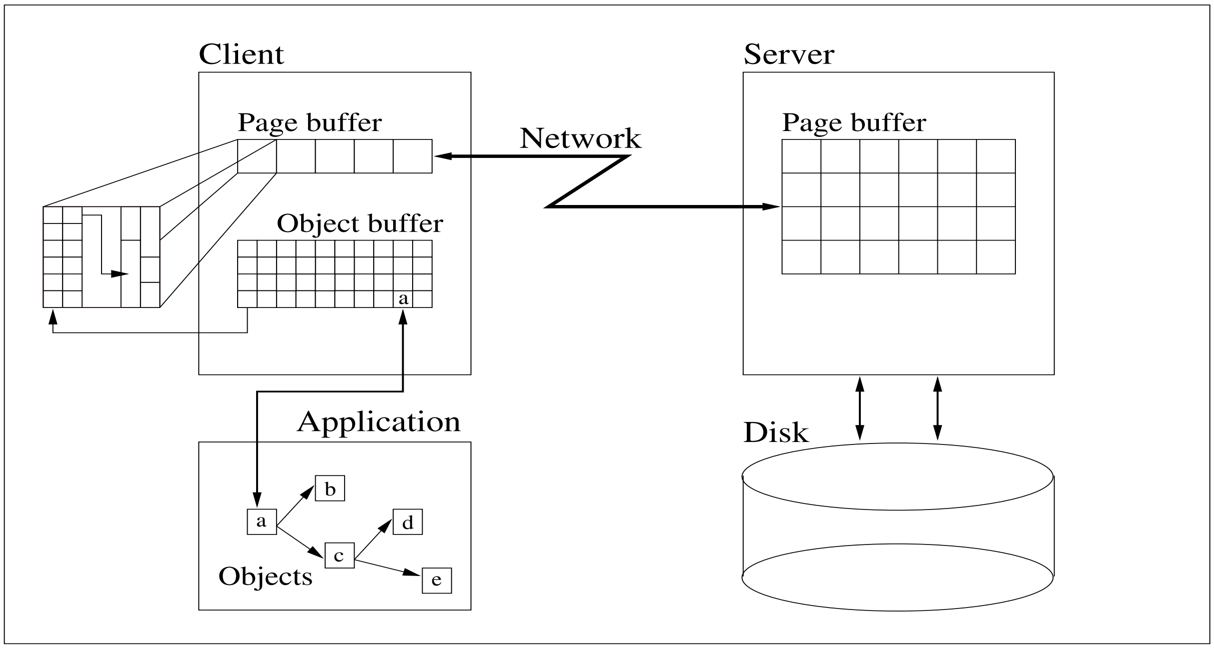

Figure 1: Page server architecture

Fig. 3 shows a typical page server architecture. Objects at the client are copied from the page buffer into the object buffer. The network is responsible for the transfer of demand and prefetch requests. A database page is divided into slots and data. The slots have information about the data and a pointer to the offset of the data. The application program has pointers to other objects. In this architecture the pointer is pointing to the object via the object buffer table (in-direct approach). Otherwise a pointer could point directly to the object (*pointer swizzling*).

Fig. 7 shows the architecture of an object server. The client has one object buffer whereas the server has an object buffer and a page buffer. The objects at the server are copied from the page buffer into the object buffer. The transfer between client and server is an object or a group of objects. The object server stores objects in units of pages or segments on disk.

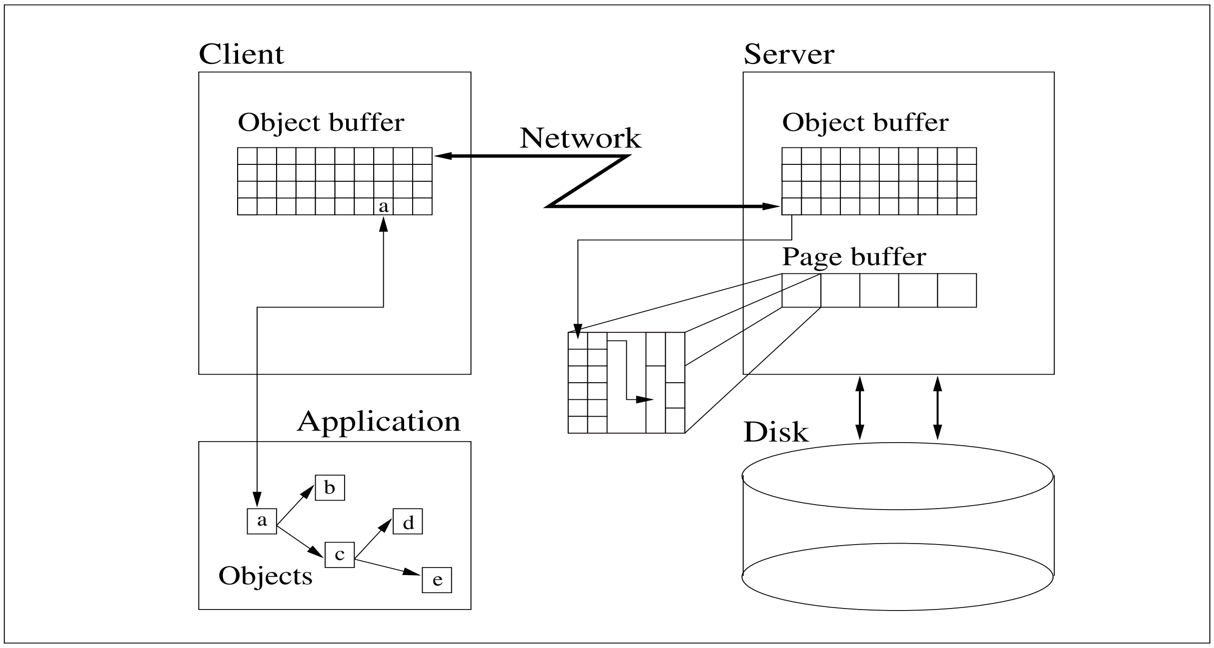

Figure 2: Object server architecture

3 Prefetching Algorithms

We compare some page prefetching techniques with several variants of object prefetching techniques. Objects have pointers to other objects. These pointers are transfered to persistent pointer by an OODBMS. In our study we only consider the static navigation through these persistent pointers for prefetching.

We assign every pointer a transition probability. All techniques compute the access probability of objects according to the transition probabilities between objects. Every transition has an associated probability between 0 and 1.

3.1 Page Prefetching Algorithms

The PPT fetches the page with the highest access probability from the current object being processed. A detailed description can be found in [16]. A difference of this technique to the technique in [16] is that this algorithm is not using a minimal threshold to start the prefetch; instead it prefetches always the page with the highest probability and only one page at the time. It also not considers any negative effects of prefetching. We compare four different techniques:

1. PSDemand

   A demand application without any prefetching.

2. PSPrefetch

A prefetching application which fetches the highest probability page.

3. PSDemandServerResPages, PSPrefetchServerResPages

   Identical applications as PSDemand and PSPrefetch but all the requested pages are resident in the servers' buffer pool. No disk request is required.

3.2 Object Prefetching Algorithms

The object prefetching techniques always load a group of objects in advance. The relevant group of objects is computed by the *Chapman-Kolmogorov equations.* Let i be the current object and j another object then the probability $\mathbb{P}$ that we will be in object j after $n+m$ steps[2] is computed by the equations:

$$\mathbb{P}_{ij}^{n+m} = \sum_{k=0}^{\infty} \mathbb{P}_{ik}^{n} \, \mathbb{P}_{kj}^{m} \text{ for all } n, m \geq 0, \text{all } i, j \tag{1}$$

These equations can be solved by matrix multiplications.[3] If the probability of an object is higher than a threshold then we insert this object into the request group. We use four different threshold parameters (0.0, 0.001, 0.01 and 0.1). Another important parameter for the prediction is the lookahead n. We vary the depth of this parameter from 1 to 19 objects. In our tests we will compare the following object prefetching techniques:

1. OSDemand

   On an object fault this technique fetches the missing object and all the objects in the lookahead n.

2. OSPrefetch

   It prefetches all the objects in the lookahead n. In the case of a miss it sends a demand request to the server.

3. OSDemandServerResPages, OSPrefetchServerResPages

   Identical applications as OSDemand and OSPrefetch but all disk pages are resident in the servers' buffer.

4. OSServerPrefetch

   The server prefetches the page with the highest priority into its buffer pool. The prefetch is executed so far in advance that when a request arrives for a page it will be already resident in the buffer pool.

[2]A step means the traversal from one object to another object.

[3]These computations are very expensive and are used to compare equal probability values with a page server. We do not compare these computation cost in this study.

5. OSPrefetchSmallDemand

   The prefetch operations are executed like in OSPrefetch but the demand operations only fetches the page with the faulted object from disk.

6. OSAbortPrefetch

   On the arrival of a new prefetch request the server will destroy all previous requests from that client. It destroys the request itself and all connected requests to the disk. All the objects that are resident are send to the client from the aborted request.

7. OSServerSendEarly

   The server sends all the arriving disk pages to the client. This means one request could result in many initiated transfers from the server.

4 System Environment

For the performance evaluation we used the simulation language C++Sim [20]. It is a discrete process based simulation language. Every component of our client/server architecture (client, server, disk, prefetch engine, network) is simulated as a process.

Table 1 shows the shared performance settings of both object and page server, table 2 shows the values of the page server and Table 3 the values from the object server. The page server has a page fetch time of 12000 μs. If the object server fetches the same amount of objects from the same page as the page server then its fetch time will be 12100 μs (slightly higher because of the object overhead involved). The cost of the network transfer depends on the amount of K-Bytes to be transfered. These variable costs appear at the client side (for receiving data), at the network (for transferring data) and at the server (for sending data). We distinguish at the server between costs for processing resident or non-resident pages. If the page is not resident there is an overhead involved to communicate with the disk thread.

A prefetch in a page server system must be started 10 processing objects ahead to achieve the total amount of savings. This is also true for the object server system if it fetches only one page from disk.

The benchmark structure we used for this test is a simple tree structure. Branch objects have two pointers to other objects. At each level in the depth of the tree structure one branch object alternates with an object which has only one pointer to another object. Every branch object has an associated probability which is varied from 0.5 to 1.0. The navigation through the object graph was controlled by a *draw-operator*[5] There are 62 objects on a disk page (8 KB) and the object size is 132 bytes. In a depth-first traversal would access 10 objects from the same page and the 11th would be an object from the next page. We assume perfect clustering for the objects on disk pages. Prefetching is only used to prefetch

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

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

Table 1: Shared performance parameter of object/page server

Parameter	μs
Object processing time	1200
Fixed network cost for one transfer	1557
Variable network time for one transfer (per kb)	132
Percentage of variable network time client/network/server	45%/18%/37%
Client message send	267
Server message receive	156
Client message receive	156
Average disk access (seek+transfer)[4]	5615
Client server-request processing	1530

Table 2: Page server performance specification

Parameter	μs
Page fetch time	12000
Server processing (page resident)	1359
Server processing (page not resident)	1664

Table 3: Object server (server side) performance specification

Parameter	μs
Initial processing	18
Call disk	105
Processing non-resident page	790
Object lookup	5
Processing after disk arrival	978

the objects/pages at the page border. We also assume the buffer space to be infinite and we do not consider any locking in our tests.

5 Performance Evaluation

5.1 Page Server Result

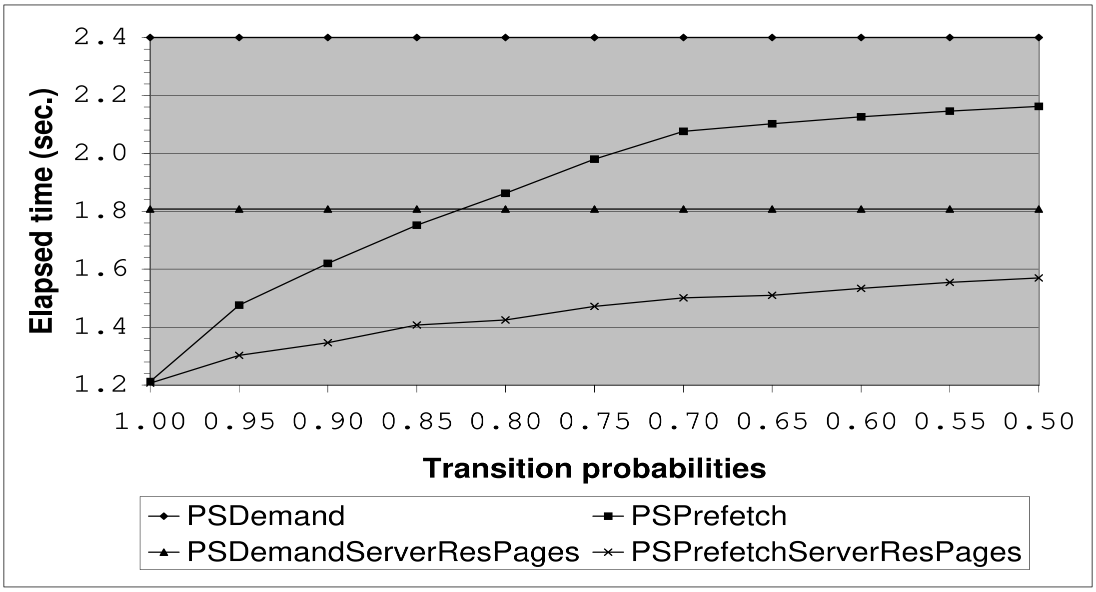

Figure 3: Page server result

Fig. 3 shows the result of the page server applications. The performance of the demand applications is independent of the transition probabilities. At the probability of 1.0 PSPrefetch (1.206) is almost as good as PSPrefetchServerResPages (1.204). With lower transition probabilities the prediction is less accurate which causes more incorrect prefetches and the elapsed time increases.

5.2 Object Server Result

The result of the object server with a threshold of 0.0 is presented in Fig. 4. At the transition probability of 1.0 Lookahead 19 performs best. A lower lookahead reduces the amount of savings. At the other transition probabilities (0.95 down to 0.5) the elapsed times for all versions are immutable. The high lookahead applications perform very badly since they fetch a lot of pages from disk and consume a lot of server time.

The explanation for these times can be found in Fig. 5. This chart shows the number of disk requests of the different applications and is identical to Fig. 4. The number of disk

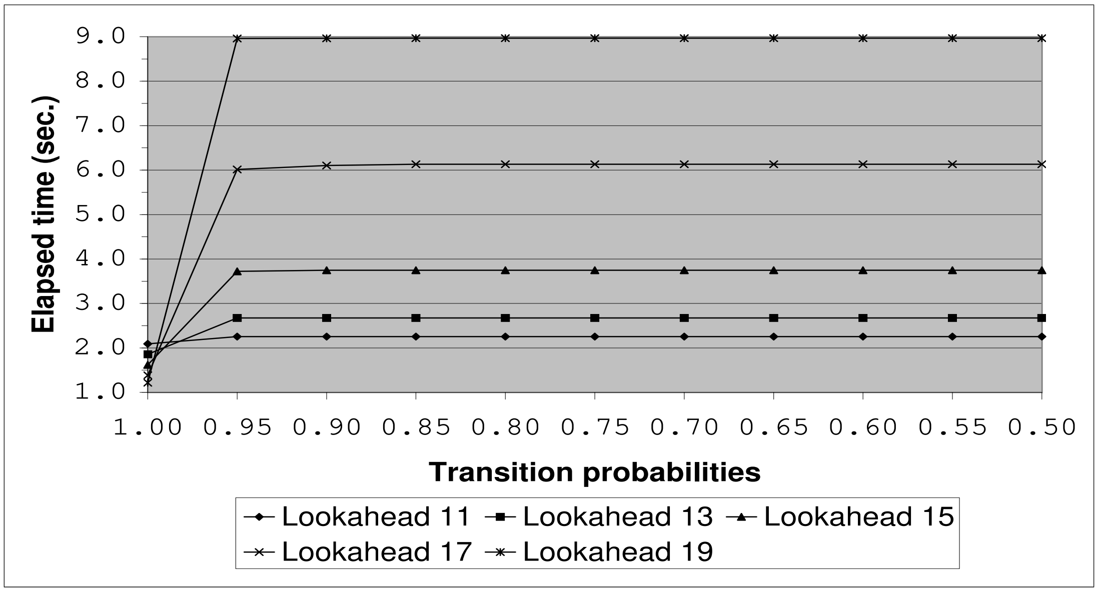

Figure 4: Object server / threshold 0.0

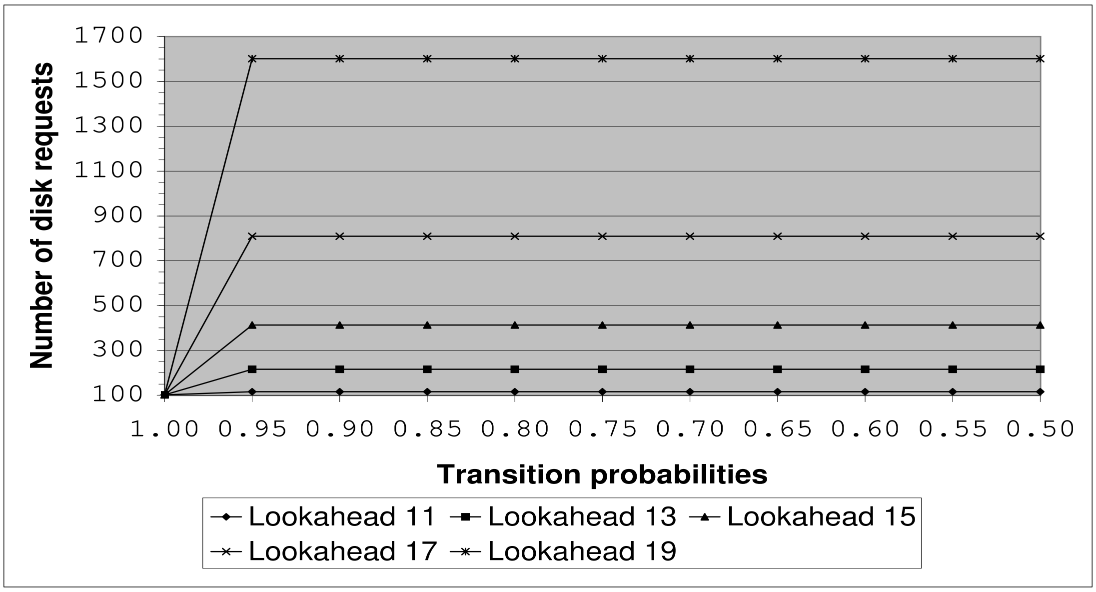

Figure 5: Object server / threshold 0.0 / Number of disk requests

Table 4: Object server / threshold 0.0 / component results

Parameter	Lookahead 11	Lookahead 15	Lookahead 19
Server processing time	0.3	1.07	3.74
Disk processing time	0.65	2.32	8.99
Number of object requests	6292	23414	73346
Number of server waits	0	0	493
Number of disk waits	0	297	1583

requests determine the elapsed time of the applications. All the main differences of single components can be found in Table 4. The lookahead 19 application prefetches always 16 pages from the current location of client processing and accesses only one of these pages.

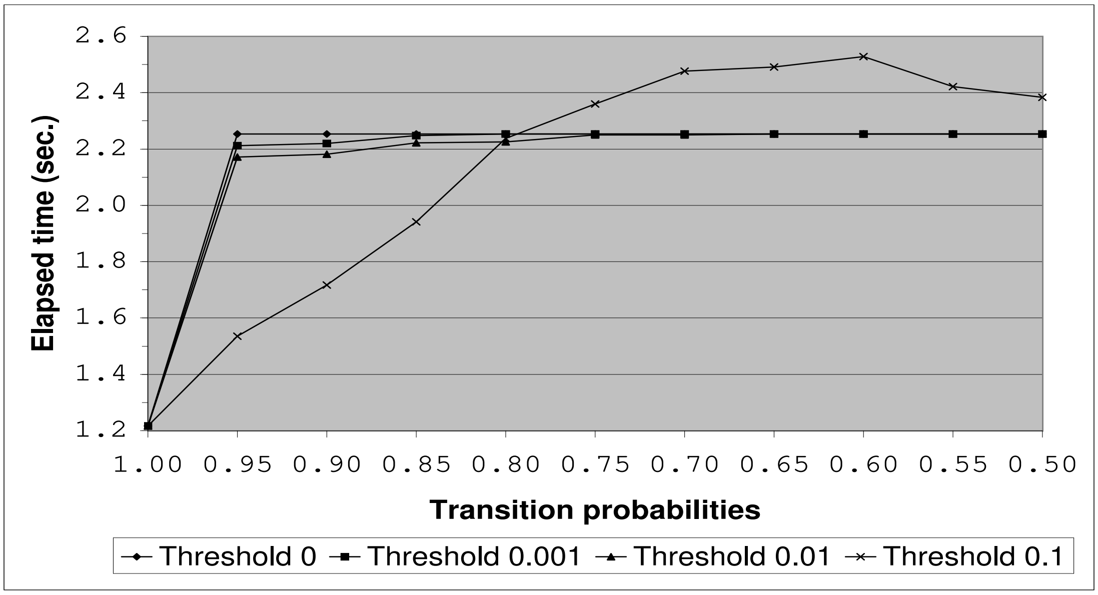

Figure 6: Object server / all thresholds

Fig. 6 shows the results of all four threshold applications. All applications with a threshold smaller and equal than 0.01 have the same elapsed time between the transition probabilities of 0.75 down to 0.5 and at 1.0. At the other probabilities these application vary slightly because of the different number of disk and object requests. A higher threshold reduces the number of object fetches but if it is too high it also aggrandises the number of demand fetches (e.g. threshold 0.1 has 4 more demand fetches). The threshold 0.01 application shows a high improvement between 0.95 and 0.85. The main difference of the application with the 0.95 transition probability can be seen in Table 5. The threshold 0.01 application starts the prefetch just two objects before access whereas threshold 0.1 has the

Table 5: Object server / transition probability 0.95 / threshold 0.01 and 0.1

Parameter	Threshold 0.01	Threshold 0.1
Prefetch distance	2	10
Demand fetch time	0.109	0.335
Prefetch wait time	0.862	0.000
Number of disk requests	109	127
Number of object requests	3094	1386
Number of object demand group requests	5	27

best performance with a prefetch distance of 10 objects. The Threshold 0.01 program has a short demand fetch time but a long prefetch wait time as it starts the prefetch too late. It fetches many objects with a few disk requests. The threshold 0.1 program prefetches more incorrect pages because of the prefetch distance.

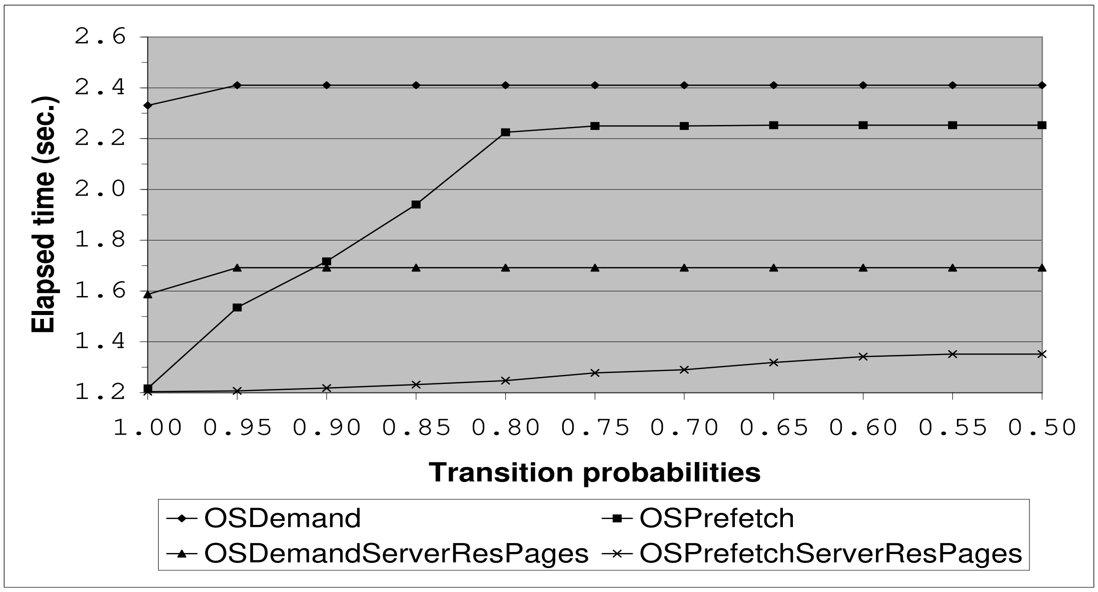

Figure 7: Object server result

Fig. 7 we see the final result of the object server performance. The OSDemand has a constant performance; only at the transition probability of 1.0 it is slightly better because of fewer object fetches. OSPrefetch has a sharp increase in performance until the probability of 0.8 and then stays constant. The explanation for this is the same as the difference between the threshold 0.01 and threshold 0.1 program in Fig. 6. Before the 0.8 transition probability the prefetch distance is 10 and savings are high; after 0.8 the distance is 2. OSDemandServerResPage shows similar results to OSDemand. OSPrefetch-

ServerResPage performs best at all transition probabilities. It increases slightly with lower transition probabilities because of lower object destination probabilities and a later start of the prefetch.

5.3 Object and Page Server Comparison

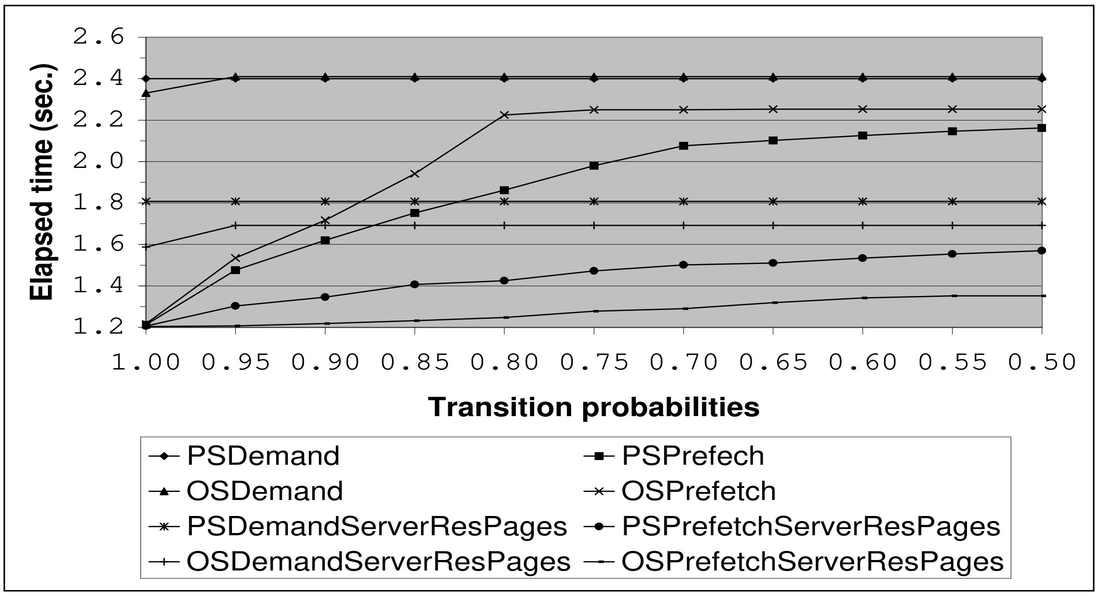

Figure 8: Object/page server comparison

In Fig. 8 all previously presented results are compiled into one graphic. OSPrefetch is worse at every transition probability than PSPrefetch. The highest difference is at 0.8 when OSPrefetch uses a prefetch distance of 2 and PSPrefetch still achieves the best result at a distance of 8. The result is different for the applications where the pages are resident at the server. OSPrefetchServerResPages is able to select all the important objects for an object group without doing any expensive page fetches. PSDemand and OSDemand are almost identical: at 1.0 OSDemand is slightly better because of less object fetches and after 1.0 PSDemand improves as it has less overhead in object management. OSDemand-ServerResPages clearly performs better than OSPrefetchServerResPages in all probabilities.

5.4 Object Server Performance Optimisations

In the previous object prefetching technique the client send many prefetch requests to the server. The server is processing this request as long all the objects are resident at the server and the disk request arrived. This means that the server could process several requests from the client at the same time. Some of these requests could be out of date, i.e. the

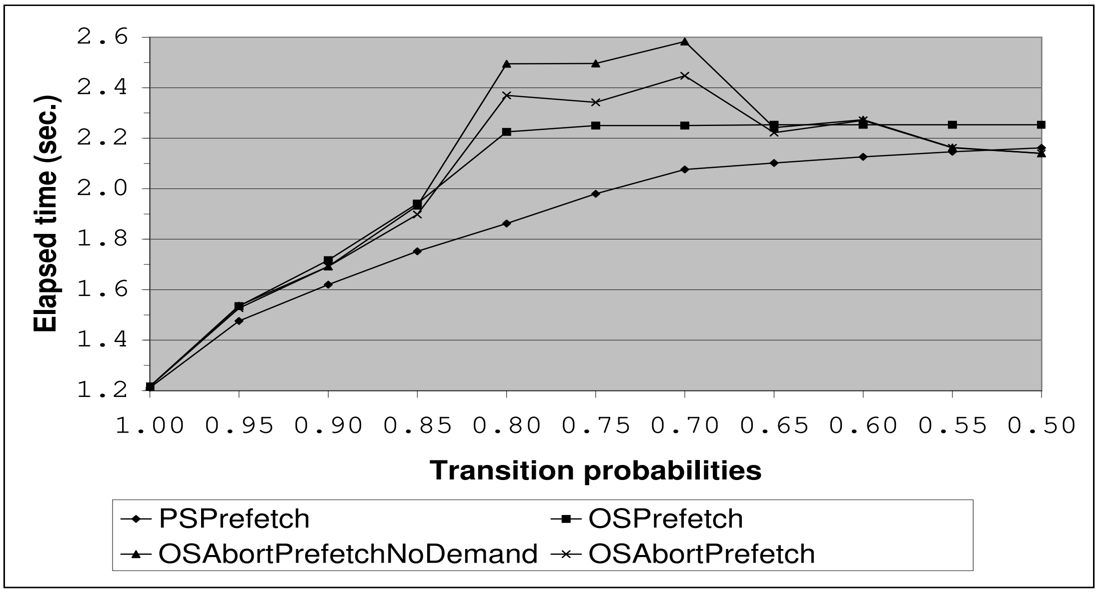

Figure 9: Server prefetch abort

applications' navigation is ahead of the prefetch request or the application changed its navigation totally.

We developed the two algorithms (Fig. 9) OSAbortPrefetch and OSAbortPrefetchNoDemand that abort all previous requests from the client at server on the receipt of a new request. It will abort all the disk requests from the server request and the server request itself. OSAbortPrefetch aborts also previous demand requests from the client whereas OSAbortPrefetchNoDemand only aborts previous prefetch requests. The performance of both abort versions are very close to the OSPrefetch version. Between the transition probabilities of 0.7 and 0.8 both abort versions perform badly. The reason for the performance is shown in Table 6.

The abort prefetching application does not check which objects are currently prefetched. It computes the relevant objects from the current context and sends this request away. This approach obviously increases the number of object requests. This server and disk time is also higher because of the processing overhead at the server. More object requests involve more disk requests and the disk system is busy with requests that are later not needed because of an abort. Every time a served disk requests arrives at the server, the server will check if the request is aborted. If yes, it sends all the objects that are resident to client and reduces the open request list from the client. The overhead for the server processing and the start of the network increases the server time. OSAbortPrefetchNoDemand and OSAbortPrefetch do better than PSPrefetch and OSPrefetch at the 0.5 probability but cannot improve performance at the other probabilities.

The object server showed its superiority when all pages are resident at the server. For times when the server workload is low the server could prefetch pages from disk according

Table 6: Object prefetching without and with abort (transition probability: 0.7 / prefetch distance: 2 / threshold: 0.01)

Parameter	No Abort	Abort
Demand fetch time	0.109	0.097
Prefetch wait time	0.941	1.151
Total server time	0.354	1.850
Total disk time	0.646	2.184
Number of disk requests	115	389
Number of object requests	6090	15101
Number of prefetch group requests	99	199

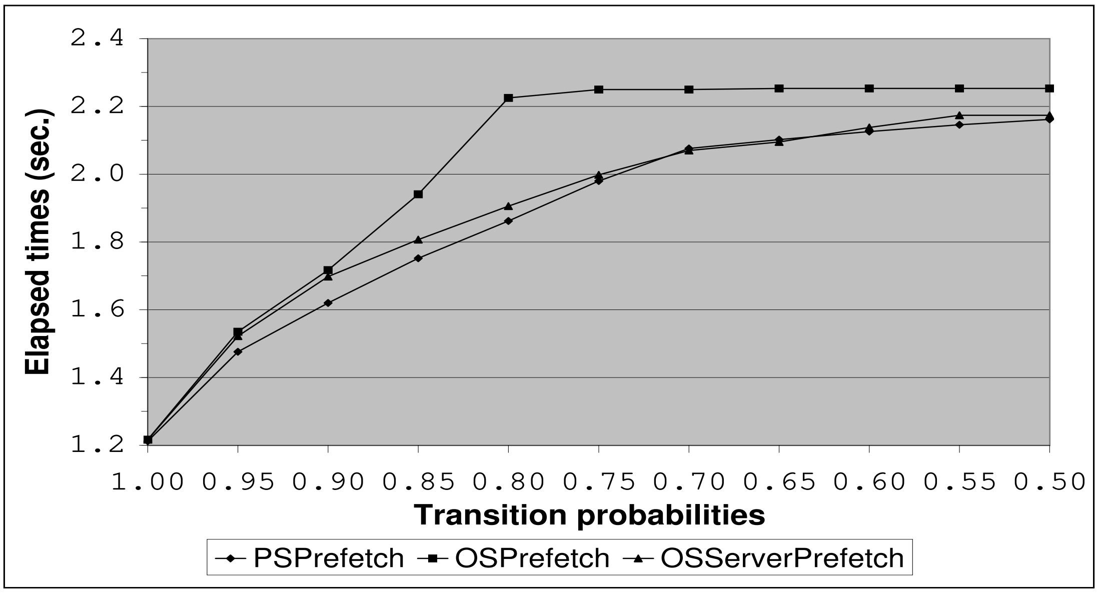

Figure 10: Server prefetching

to the prefetch information from a client. OSServerPrefetch starts a disk requests after the service of a prefetch request. The server prefetches the highest probability page of an additional lookahead that is not in the current request. Server prefetches have an extra low priority queue at the disk. If the queue still has requests from the same client it will delete all old requests. Fig. 10 demonstrates that OSServerPrefetch perform much better than OSPrefetch and very close to PSPrefetch. Of course, PSPrefetch could also do server prefetching. At the moment OSServerPrefetch fetches only one page at a time from the disk. Fetching multiple pages could provide in an even better result. Fig. 11 shows that most of the savings can be achieved at higher transition probabilities because these are the best prediction possibilities. Also the applications with the shorter distances achieve the best savings. The distance 10 application cannot achieve any improvement.

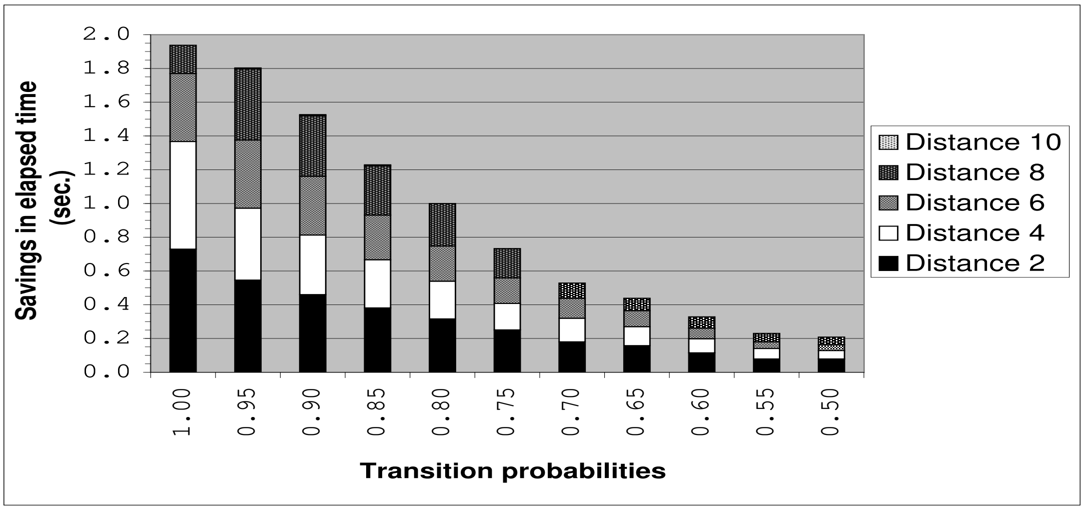

Figure 11: Server prefetching improvements for threshold 0.0

Another performance optimisation for the object server is the separate sending of pages to the client after receiving them from disk. This disk access is the most expensive part of the object fetch. The client could be waiting already for objects of that page and therefore it would make sense to start a network transfer after the disk receipt. Fig. 12 presents the result of this test. OSServerSendEarly shows a good improvement between the transitions probabilities of 0.65 and 0.85 but cannot achieve the result of PSPrefetch. The best improvement is at 0.85. Table 7 shows the differences of an 0.85 version. The prefetch wait time of the send early version is much lower because this version does not wait for the complete service at the server. The disadvantage of the send early application is that it increases the network service time with a fixed cost for every message. The server time is also increased because of the encoding of the message.

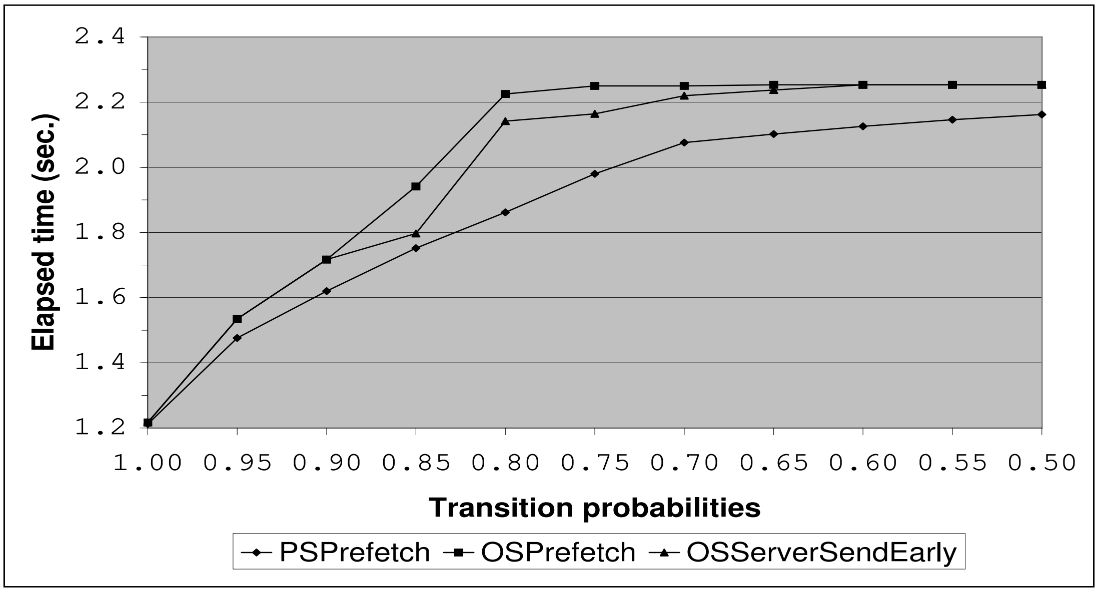

Figure 12: Server send early

Table 7: Object prefetching - server send early (transition probability: 0.85 / prefetch distance: 10 / threshold: 0.1)

Parameter	Normal object prefetch (sec.)	Send early (sec.)
Prefetch wait time	0.154	0.009
Total network time	0.049	0.068
Total server time	0.424	0.462

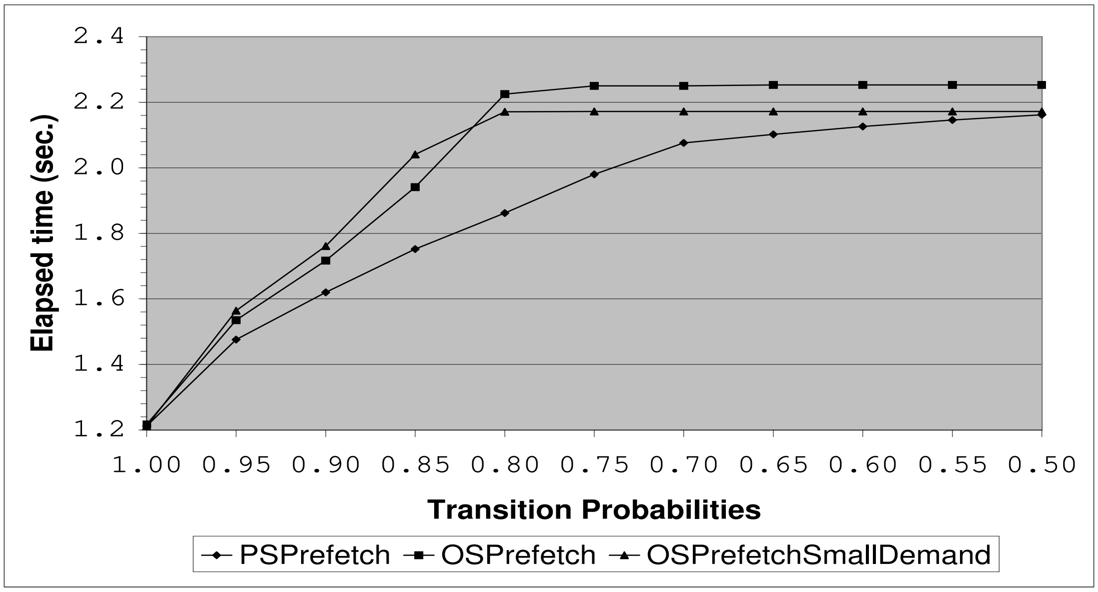

Figure 13: Small demand fetch

Fig. 13 shows the result of OSPrefetchSmallDemand application. It performs worse than OSPrefetch until the transition probability of 0.85 but then performs better than OSPrefetch. Before the probability of 0.8 the demand fetch can predict the next objects over the demand page accurately and therefore fetches more useful objects in one request and reduces the network load. After 0.8 prediction is not so accurate and the demand fetch requests many unnecessary objects which can be avoided with OSPrefetchSmallDemand. It cannot achieve the performance of PSPrefetch.

6 Conclusions and Future Work

We compared the performance of a prefetching page server with prefetching object server. We simulated the object access pattern by assigning transition probabilities to the object relationships. According to the transition probabilities we compute the access probability of pages and objects. The general result of our simulation is that the page server performs better than the object server if pages are not resident at the server. If all the pages are resident at the server the object server performs best. We presented some performance optimisations for the object server. The most effective technique was to do server prefetching. We still have to combine the optimisations for a version to compare with a page server.

In the future we will enhance this study by using different clustering levels. If object clustering is good then the page server has a performance advantage, otherwise the object server has an advantage. We would also like to test the effect of buffer replacement on the architectures.

References

[1] M.J Carey, D.J. DeWitt, M.J. Franklin, N.E. Hall, M.L. McAuliffe, J.F. Naughton, D.T. Schuh, M.H. Solomon, C.K. Tan, O.G. Tsatalos, S.J. White, and M.J. Zwilling. Shoring Up Persistent Applications. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 383–394, Minneapolis, Minnesota, May 1994.

[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] M.J. Carey, D.J. DeWitt, and J.F. Naughton. The OO7 Benchmark. Technical Report Technical Report 1140, Computer Science Department, University of Wisconsin - Madison, January 1994.

[4] M.J. Carey, J.J. DeWitt, and J.F. Naughton. The OO7 Benchmark. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 12–21, Washington, USA, May 1993.

[5] M.J. Carey, M.J. Franklin, and M. Zaharioudakis. Fine-Grained Sharing in a Page Server OODBMS. *SIGMOD Records*, 5:359–370, May 1994.

[6] E.E.-L. Chang. *Effective Clustering and Buffering in an Object-Oriented DBMS.* PhD thesis, Computer Science division, EECS department, University of California at Berkeley, 1989.

[7] 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.

[8] GemStone Servio Logic Corporation. Product Overview, February 1991.

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

[10] D.J. DeWitt, D. Maier, P. Futtersack, and F. Velez. A Study of Three Alternative Workstation-Server Architecture for Object-Oriented Database Systems. In *Proc. of the Sixteenth Int. Conf. on Very Large Data Bases*, pages 107–121, Brisbane, Australia, 1990.

[11] C.A. Gerlhof. *Optimierung von Seicherzugriffskosten in Objektbanken: Clustering und Prefetching.* PhD thesis, Faculty for Mathematics and Computer Science, University of Passau, 1996.

[12] U. Hohenstein, V. Pleßner, and R. Heller. Evaluating the Performance of Object-Oriented Database Systems by Means of a Concrete Application. In *Proc. of the 8th Int. Workshop on Database and Expert Applications*, Toulouse, France, September 1997.

[13] A.L. Hosking and J.E.B. Moss. Object Fault Handling for Persistent Programming Languages: A Performance Evaluation. In *Proc. of the Conf. on Object-Oriented Programming Systems, Languages and Applications*, pages 288–303, Washington, DC, September 1993.

[14] 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.

[15] 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.

[16] N. Knafla. Analysing Object Relationships to Predict Page Access for Prefetching. In *Proc. of the Eighth Int. Workshop on Persistent Object Systems: Design, Implementation and Use (POS-8)*, Tiburon, California, August 1998.

[17] P. Krishnan. *Online Prediction Algorithms for Databases and Operating Systems*. PhD thesis, Department of Computer Science, Brown University, 1995.

[18] C. Lamb, G. Landis, J. Orenstein, and D. Weinreb. The Objectstore Database System. *Communications of the ACM*, 34(10):50–63, October 1991.

[19] B. Liskov, A. Adya, M. Castro, M. Day, S. Ghemawat, R. Gruber, U. Maheshwari, A.C. Myers, and L. Shira. Safe and Efficient Sharing of Persistent Objects in Thor. In *Proc. of the ACM SIGMOD/PODS96 Joint Conf. on Management of Data*, pages 318–329, Montreal, Canada, June 1996.

[20] 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, Newcastle upon Tyne, NE1 7RU, UK, July 1993.

[21] Objectivity/DB Technical Overview, 1994.

[22] ONTOS, Inc. ONTOS Object Integration Server (ONTOS OIS) Integrating Objects with Relational Databases. Technical Overview, 1995.

[23] 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.

[24] Versant. ODBMS Performance Issues - A Versant Technical Overview, 1994.