データベースシステム 116－10
（1998. 7. 8）

# シェアドナシング並列OODBMSにおける並列ナビゲーションのパフォマンス評価

ムテンダ ローレンス、大津金光、吉永努、馬場敬信
mutenda@tkl.iis.u-tokyo.ac.jp
東京大学生産技術研究所

{kim,yoshi,baba}@lynx.infor.utsunomiya-u.ac.kp
宇都宮大学工学部情報処理工学科

**Abstract**

並列オブジェクト指向言語である A-NETL に基づく並列 OODBMS を提案した。この OODBMS は、シェアドナッシング環境に対して設計した。本稿では並列オブジェクト指向環境に対する並列ナビゲーションアルゴリズムに述べる. このアルゴリズムはシステムノードのロードバランシングを含む.OO7 ベンチマークに基づいた評価により 16 ノードで 1.7 スッピドアップを得た.

# Evaluation of a Parallel Navigation Algorithm for a Shared-Nothing Parallel OODBMS

Lawrence Mutenda, Kanemitsu Ootsu, Tsutomu Yoshinaga and Takanobu Baba
mutenda@tkl.iis.u-tokyo.ac.jp
Institute of Industrial Science, Tokyo University.

kim,yoshi,baba@lynx.infor.ustunomiya-u.ac.jp
Department of Information Science
Utsunomiya University
Utsunomiya 321, JAPAN
Phone/fax: 028-689-6284

**Abstract**

Object oriented database management systems provide rich facilities for the modelling and processing of structural as well as behavioral properties of complex application objects. However due to their inherent generality and the high performance demand in many application domains, efficient parallel algorithms are needed to meet the requirements of such systems. In this paper we evaluate an algorithm for navigating complex object assemblies in parallel in a shared-nothing environment. Evaluation based on the OO7 benchmark adapted for a parallel environment achieves 1.7 speedup over 16 nodes.

# 1 Introduction

The object oriented data model is expected to be among the most prominent models for representing and manipulating data. Business application oriented relational systems cannot provide the functionality required by newer application areas, like CAD, CAM and GIS, which require more powerful structural and behavioural facilities. OODBMS are a viable alternative to relational databases in these new areas [6] [4] . Much research has been done for OODBMSs and a number of systems have become commercially available, for example Gemstone, Vbase, ObjectStore and Versant [7].

Due to the inherent generality of OODBMS and the high performance requirements of corresponding applications, high performance implementation of system algorithms is required [3], the obvious candidate to obtain such performance gains being parallel processing. The past several years have seen the significant commercial success for relational databases, demonstrating convincingly that parallelim is a highly effective tool for providing high performance processing[4]. In addition, powerful algorithms have been designed like the Grace-Hash algorithm [10], to improve the performance of the join function, an expensive operation in relational database systems. Some work has been done on parallelism in OODBMS which has demonstrated that parallel processing can be effectively applied in such environments [3] [4] [9] [5]. As in relational databases, in these research projects, data is partitioned into a number of disks, in a parallel shared-nothing environment, to remove the I/O botteneck and then apply processing to the data in an MIMD fashion.

Similar to the way in which the join function is an important operation for relational database, our work tackles the problem of traversing links in an object graph to identify the objects that make up a complex object.In a parallel database, I/O bandwidth, a major bottle neck for databases, is increased by introducing multiple disks. Traversing the links of an object stored in one disk may require accessing a second disk. Ghandeharizadeh et al [6] introduce a way partition data in an OODBMS to limit inter-disk references. Without paying much attention to careful object placement, we present an algorithm for speeding up the traversal of an object graph in a shared-nothing OODBMS. The shared nothing paradigm has been the system of choice for parallel systems since it facilitates scalability. We present experimental evaluation of the algorithm based on the 007 benchmark [2].

The rest of this paper is organized as follows. Section 2 briefly describes related work. The design of the algorithm is covered in section 3. Experimental evaluation are described in section 4. Finally we discuss results, further work and conclude in section 5.

# 2 Related Work

Research in parallelism for OODBMSs is relatively new, compared to that for parallel relational databases. Kim[9] describes sources of parallelism in an OODBMS. Here path parallelism, node parallelism and class-hierachy parallelism are identified as three types of parallelism possible in a OODBMS. All three schems are based on a object query graph. Path paralelism is the one that most resembles the algorithm evaluated in this paper. Kim defines path parallelism as the explotaion of all different paths in the query graph such that all the nodes in the different paths are processed in parallel. The difference with our work is the we use the path expression based on the notion of a set-valued attribute. Here one path in a query graph becomes multiple paths if that source node follows that path from a set-valued attribute. Each of the object elements of the set-valued attribute becomes an indepedent path that can be processed in parallel.

The algorithm we describe assumes that the OODBMS uses phyical OIDs (Object Identifiers), so that the node and disk page on which that particular object is stored can be found by examining its OID. In this respect our work is similar to the parallel pointer-based join techniques described by

Lieuwen et al [11]. Lieuwen describes pointer-based joins for 2 classes where one class links to a second class using a set-valued attribute. We use the same idea here but we also apply it to a path-expression of infinite length. We also use dynamic load balancing to even out the load in the case where some nodes process more objects than other.

The work described in [4] uses par-sets to parallelize traversals of objects in a Parallel OODBMS. We share goals with this work, i.e, speeding up the navigation of objects related by links but the methods used are different. Chen et al [3] introduce elimination and identification based techniques, that use the concept of multi-wavefronts. Again the methods used differ from our work.

All parallel databases systems use data partition across multiple disks to increase I/O bandwidth. Ghandeharizadeh [6] explores the efficient partition of objects across disks in a shared-nothing environment to maximise parallelism and minimise interference from excessive message passing. Our parallel Navigation Algorithm does not assume particular partitioning scheme. However we intend to investigate the effect of data partioning that results in low locality of reference. Our evalution basically assumes that we have 90 percent locality of reference and 10 percent external references.

# 3 Algorithm Description

Having described work related to our own we will describe the parallel navigation with dynamic load-balancing algorithm evaluated in this paper.

Every object has a set of attributes which describe its composition. Some of these attributes are set valued, with either absolute values or object-identifier (OID) values . For example an object such as a three-dimensional prism may have an attribute, *sides*, which may be a collection of OIDs dereferencing the sides of the object. Set-valued objects whose elements are OIDS are quite common in OODBMSs.

It is important to consider the parallel navigation of object pointers of such objects. Consider the object whose structure is shown in Fig. 1. The *professor* object has a set-valued attribute, *courses* which is a set of OIDs representing the courses that professor teaches. In turn each course has a set-valued attribute whose value is the set of *students* (OIDs) taking that course.

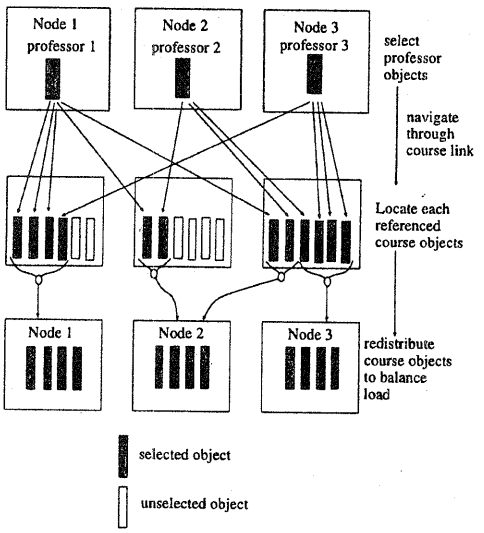


Figure 1: Navigating in Parallel with Load-Balancing

Fig. 1 also shows how such object may be traversed in parallel. The first stage in the navigation process involves selecting the professor objects at each node. The OIDs in the set-valued attribute courses are then traversed to select course objects in the second stage. At this point it is noted that node 3 will need to process 6 objects but node 2 only has two selected objects. The system then initiates transfer of two objects from node 3 to node 2 at the third stage, to balance the system load. Such transfer will be especially significant if each object requires a method to be executed.

## 3.1 Parallel Navigation with Dynamic Load Balancing

Our algorithm assumes a shared-nothing environment where each processing node in the system has

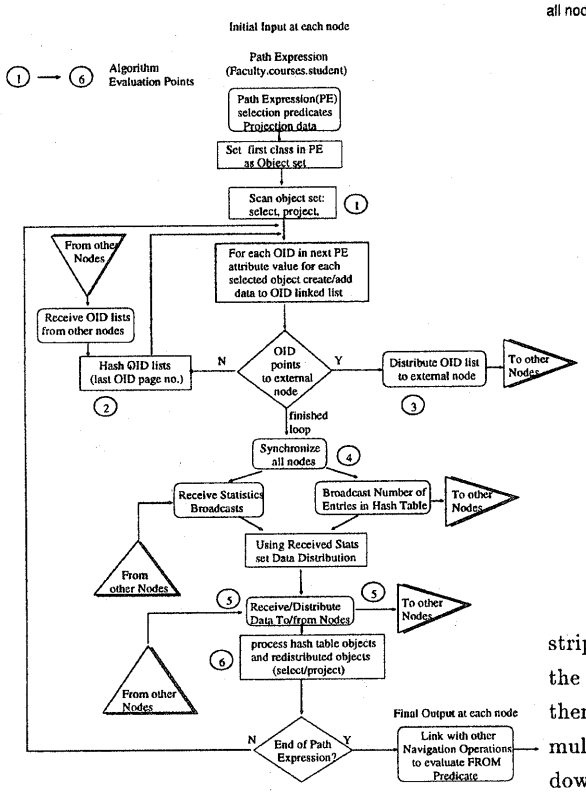


Figure 2: Parallel Navigation with Dynamic Load Balancing Algorithm

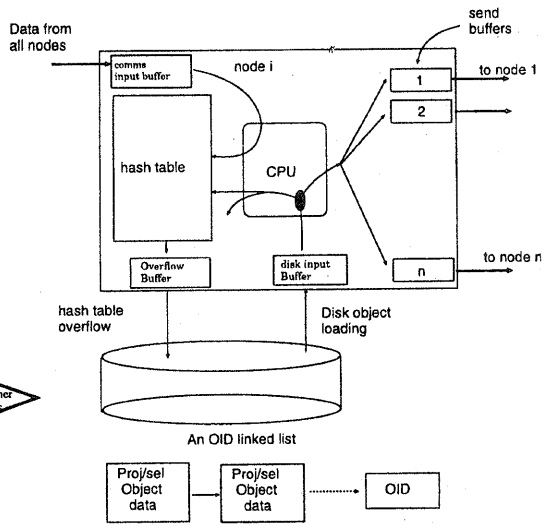


Figure 3: Data Distribution Diagram

access to its own disk and communicates with other nodes in the system via message passing.

Figure 2 is a flow chart of the algorithm that implements parallel navigation as described above. We assume that an object OID is physical and identifies the node and page on which an object is located and that objects are not shared. The input to the algorithm is a path expression (PE) of arbitrary length as well as any necessary selection predicates and projection requirements. We will briefly explain the algorithm below.

Take the *faculty.courses.student* as an example path expresssion. The algorithm executing at each system node will take the faculty class, scan the portion of the faculty class at that node, selecting and projecting as required by the query. The query relevant data from these objects is used to create a stripped down version of the object. The value of the next attribute in the PE , in this case *courses*, is then examined. For each object, every OID in this multi-attribute value is combined with the stripped down version of the object to form an OID linked list. This linked list as shown in, fig. 3, consists of the source object data and an OID from a multivalued attribute as the last element.

Each OID linked list is then transmitted to the node pointed to by its last element (a multi-valued attribute OID) . Lists bound for the node executing the algorithm are hashed on the page number of the last element. Lists going to other nodes are transmitted accordingly and each node receives lists sent from other nodes and inserts them into its hash table. The process is illustrated in Fig. 3. When all OID lists have been processed a synchronisation step then is executed. Dynamic load balancing is then initiated.

It is likely that when data is referenced via OIDs somes nodes may contain more query relevant objects than others. If such object data skew is severe it can potentially lead to load imbalance. The time to each synchronisation scheme is determined by the slowest node in the system. Our navigation algorithm implements where objects are relocated from lightly loaded nodes to heavily loaded nodes. We

Table 1: Parameter Definition List

| Parameter | Value |
|---|---|
| Size of a disk page | 8kbytes |
| Size of an object | 256bytes |
| Number of nodes | 2-16 |
| Send time(8kbytes) | 2.5ms |
| Receive time (8kbytes) | 7.2ms |
| Disk Access (8kbytes) | 20ms |
| Method execution | 10ms |
| Number of Objects | 200,000 |
| Number of links/object | 3 |
| Reference Locality | 90,70,50,30 % |

assume that the objects all need to have a method executed on them. Each method is assumed to take approximately the same amount of time for execution. Balancing the load is then a matter of sending some objects to lightly loaded nodes for execution of a method. Note however that since method is abitrary, execution time for a method is difficult to predict before hand. However in a real life situation, a database can determine the average execution time for frequently used methods and use those averaged times to determine wether load balancing should be executed. Dynamic load balancing in our algorithm does not eliminate but rather limit loads at all node to within 10 percent of the average system load.

After load balancing, the second level of the path expression is executed with the algorithm looping until all path expression elements are processed.

# 4 Algorithm Evaluation

We implementated our parallel navigation algorithm on the A-NET multi-computer in the parallel object oriented language A-NETL [1].

The A-NET multi-computer is a prototype machine with 16 processing nodes. Each node has one processing element and one router. each node is connected to the other nodes via a statically reconfigurable topology. However the machine cycle of the A-NET machine, a prototype, is low at 167ns, but the results we report here can be taken to be relative rather than absolute. The A-NET machine does not have user disk access facilities and therefore our evaluation employed a disk simulator.

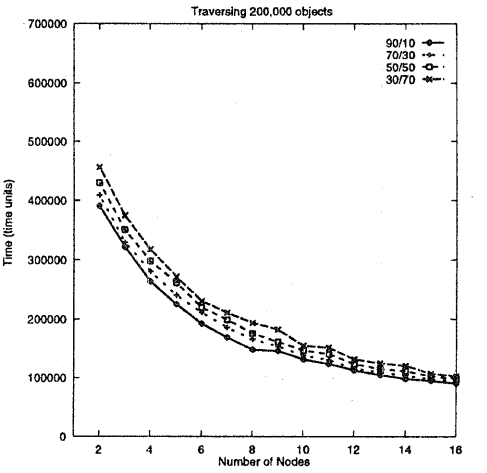


Figure 4: Response time with Varying Locality of Reference

A-NETL is a locally designed parallel object oriented language designed for use in defining parallel object-oriented programs. The language provides easy to use message passing functions, object definition facilities and efficient program synchronisation constructs. Objects in A-NETL communicate via message passing.

We based our evaluation on the OO7 [2] benchmark adapted for a parallel environment but took liberties to alter some of the provisions of the benchmark to suit our conditions. We report results on the traversal operation of the benchmark.

Table 1 shows the parameters used in the evaluation. The send and receive times indicate the actually send and receive times on the A-NET machine. For disk access we use the time normally used for such accesses [12].

Evaluation ignored the effects of memory buffers and assumed that all objects can fit in memory at the same time, a condition that can be fulfilled with today's large memories. We also evaluated traversal for the "cold" case where all needed objects have to be accessed from disk. To illustrate the effect of message passing in this case we also varied the ref-

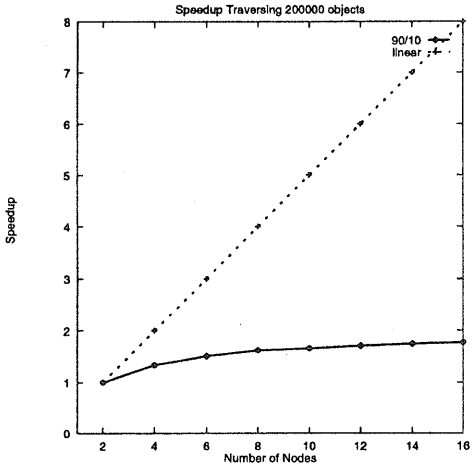


Figure 5: Speedup for 2-16 nodes

erence locality for each node, i.e, the percentage of OIDs pointing to external nodes and the percentage referencing local objects.

Fig. 4 shows the results of navigating 200,000 objects with varying locality of reference. All four graphs scale reasonable well when the number of nodes varies from 2 to 16. It can alo be deduced that when the locality of reference is high, execution time increases. This is due to the increase in message passing. A locality of reference of 30 % local and 70% remote gives the worst performance for the four cases. This is not unexpected and Ghandeharizadeh et al [6] attempt to reduce remote object access by locating objects on nodes with the high access frequency for an object.

Fig. 5 shows the speed up. The speedup from 2 to 16 nodes is about 1.7. This may be due to the dominace of disk access which is a sequential process. It would be instructive to examine speedup in the case of a main memory OODBMS.

# 5 Conclusion and Further Work

We have described the design outline of the parallel navigation algorithm and compared it with related work. We have also presented results from experimental evaluation of the algorithm on the A-NET multicomputer using the A-NETL language. Preliminary results show a speedup of 1.7 from 2-16 nodes. We are continuing the evaluation of the algorithm to improve performance. We will evaluate the dynamic load balancing scheme included in our algorithm. At the same time we will run the rest of the tests in the OO7 benchmark.

### Acknowledgments

The authors would like to thank other members of the A-NET lab for their helpful comments.

# References

[1] Baba T., Yoshinaga T., Furuta T.: *Programming and Debugging for Massive Marallelism:The Case for a Parallel Object-Oriented Language A-NETL*, Proc. France-Japan Workshop on Object-Based Parallel and Distributed Computation (OBPDC'95), Lecture Notes in Computer Science 1107, pp 38-58 (1996).

[2] Carey M.J., DeWitt D.J., Naughton J.F.: *The 007 Benchmark*, Univ. Wisconsin January (1994).

[3] Chen Y.H., Su S.: *Identification and Elimination-based Parallel Query Processing Techniques for Object-Oriented Databases*, Journal of Parallel and Distributed Computing, Vol. 18, No. 2, August (1995).

[4] DeWitt, D.J., Naughton J.F., Shafer J.C., Venkataraman S.: *ParSets for Parallelizing OODBMS Traversals: Implementation and Performance*, Univ. Wisconsin Tech Report (1995).

[5] Ghandeharizadeh S, Choi V., Ker C., Lin K. : *Design and Implementation of the Omega object-based system*, Proc. of the Fourth Australian Database Conference, (1993).

[6] Ghandeharizadeh S, Wihite, D. Lin,K. Zhao X.: *Object Placement in Parallel Object Oriented Database Systems*, Proc 10th International Conference on Data Engineering, pp253-262 (1994)

[7] Kempter A., Moerkotte G.: *Oject Oriented Database Management*, Prentice Hall, pp609-620, (1994).

[8] Khosafian S. Valduriez P Copeland G.:*Parallel query processing for complex objects*,Proc 4th Int'l Conf on Data Engineering, pp202-209, (1988)

[9] Kim K.C *Parallelism in Object-Oriented Query Processing*, Proc. 6th Int'l Conf on Data Engineering,pp. 209-217, (1990)

[10] Kitsuregawa M, Tanaka H. Motooka T.:*Application of Hash to Data Base and its Architecture* New Generation Computing No 1, pp62-74 (1983).

[11] Lieuwen, D.F., DeWitt, D.J., Mehta M. *Parallel Pointer-based Join Techniques for Object-oriented Databases*, Univ. Wisconsin Tech Report (1993)

[12] Walton C.B., Dale A.G., Jenevein R.M.: *A Taxonomy and Performance Model of Data Skew Effects in Parallel Joins*, Proc. of the Int. Conf. on VLDB, September 1991 pp537-548.