

Object Placement in Parallel OODBMS

Zhen He and Jeffrey Xu Yu

Department of Computer Science
The Australian National University
Canberra, ACT 2611

Abstract. Effective data placement is crucial to the performance of any parallel database system as it is an important lever for load balancing. In this document we describe and explain a two-phased approach to the placement of objects in a shared-nothing model of a parallel object oriented database with page grain caching. The first phase is a declustering phase in which a greedy similarity graph partitioning algorithm is used to assign objects into nodes of the shared-nothing system. The aim of the first phase is to minimise internode traversals and maximise parallelism. Once assigned to nodes, the second phase attempts to cluster objects residing at each node into pages with the aim of reducing occurrence of remote page requests. Therefore, objects that are likely to be accessed by another remote node are clustered together. Both phases use query frequency distribution information to decide the placement of objects. We implemented this two phased approach on the Fujitsu AP1000 with a slightly modified OO7 benchmark data set. The results presented in this paper demonstrate that this algorithm was able to reduce remote page loads while preserving a high degree of parallelism.

1 Introduction

Object-oriented data management systems (OODBMS) [Zdonik and Maier, 1990] provide rich facilities for the modelling and processing of structural as well as behavioural semantics associated with complex objects in many applications. Some of these applications include Computer-Aided Design (CAD), office information systems, spatial information systems, etc. Research into parallel object-oriented database systems demonstrates that parallelism is a highly effective tool for providing high performance [Thakore and Su, 1994, DeWitt et al., 1996, Kim, 1990]. Similar research into parallel relational database systems has also proved successful in extracting performance [Lu et al., 1994].

Proceedings of the Tenth Australasian Database Conference, Auckland, New Zealand, January 18–21 1999 Copyright Springer-Verlag, Singapore. Permission to copy this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or personal advantage; and this copyright notice, the title of the publication, and its date appear. Any other use or copying of this document requires specific prior permission from Springer-Verlag.

Effective data placement is crucial to the performance of any parallel database system since it is an important lever for load balancing [Padmanadhan and Baru, 92, Chen et al., 1995, Copeland et al., 1988]. In general, determining the optimal placement of data across nodes for performance is a difficult problem even in relational database systems. Object-oriented database systems introduce additional complexity by supporting complex data structures which results in the object base being representable as graphs. In parallel object-oriented databases on shared-nothing architectures, the object base needs to be declustered across the nodes of the system. The manner of declustering depends on the way queries are processed. For intra-operator parallelism in parallel object-oriented databases, it is important that the object placement strategy strikes a compromise between two conflicting objectives [Ghandeharizadeh et al., 1994]: a) the object placement strategy should assign all the relevant traversals by a query to a single node in order to minimise the number of inter-node references, and b) it should distribute the workload of an application evenly across the nodes in order to maximise the processing capability of the system. These objectives are in conflict because a) argues for assigning objects to a few nodes in order to group the relevant objects together, while b) advocates the distribution of the objects across as many nodes as possible.

We have based our object placement strategy on a similarity graph partitioning approach. [Fang et al., 1986] introduced similarity graph partitioning to decluster data across multiple disks of a single processor multiple disk system. They considered two groups of data items G_i and G_j similar if, for every point p_k in G_i there exists a point p_l in G_j in which p_k is a nearest neighbour of p_l , or p_l is a nearest neighbour of p_k . Minimal spanning trees and shortest spanning paths were proposed to divide a set of objects into two “similar groups”. [Liu and Shekhar, 1996] proposed a variation of this approach in which query frequency distribution information was used to define similarity. They proposed a general approach to declustering which they applied to grid files of a spatial database. Their aim was to maximise the chances that a pair of atomic data-items that are frequently accessed together by queries, are allocated to distinct disks. They defined similarity between two items as the weight of the edge joining two data items. The weight is assigned as the likelihood that the pair of data-items will be accessed together by queries in the query set of interest. They proposed two heuristic techniques to perform the similarity based declustering. The first is a global max-cut graph partitioning technique which is more expensive than the second incremental technique. The global max-cut graph partitioning technique initially partitions the data-items by some less optimal method like the incremental method and then repeatedly compares pairs of data items for possible improvements. The incremental method selects an unallocated data item and then retrieves all the data items that are in a window around the selected data item. The window around a data-item o_u is a set of data items which are likely to be accessed together with the data-item o_u . The algorithm iterates through the unallocated objects in the window in a greedy manner placing the data-items into disks based on the max-cut criteria. This method is more suitable for large

databases since it is less expensive. It is a variation on this second method that we have adopted to decluster our database.

In this document, we consider intra-operator parallelism for a parallel object-oriented database system on a shared-nothing architecture where each node has its own memory and disk. Page grain caching is also incorporated into the model. We focus on fully declustering which declusters objects across multiple nodes where each object only has a single copy in the object base. In addition, to reduced cache misses for requests by other nodes, we cluster onto the same page objects held at each node which have, similar remote access characteristics.

The remainder of this document is organised as follows. Section 2 gives notations and background information for this study. Section 3 and Section 4 discuss our declustering and clustering approaches and the heuristic techniques we used. Section 5 describes a slightly modified OO7 benchmark [Carey et al., 1993] on which we conduct our performance study. An overview of the test bed and performance results is given in Section 6 and Section 7. Finally, Section 8 concludes the document.

2 Preliminaries

We consider an object base as a directed graph. Each node represents an object and an edge (o_i, o_j) represents a reference from o_i to o_j . The weights of edges can be assigned either by dynamic reference counts or static reference counts. Dynamic reference counts are determined by the frequency of traversal of the edges. Static reference counts assign a weight to the edge joining an object with another object a value of 1 if a reference exists otherwise a value of 0 is assigned [Tsangaris and Naughton, 1991]. This makes object placement in an object oriented database a graph partitioning problem. For this study we use dynamic reference counts to represent weights on the edges of the graph.

Consider a shared-nothing system with N nodes, and a set of m queries Q_1, Q_2, \dots, Q_m . Also assume the queries are processed via intra-operator parallelism. We seek an object placement which will make $\sum_{i=1}^m Cost(Q_i)$ minimum where $Cost(Q_i)$ is the cost to execute query Q_i on the N node shared-nothing architecture.

The main issue here is how to balance the two conflicting goals as mentioned in [Ghandeharizadeh et al., 1994].

- The object placement strategy should assign all the relevant traversals by a query to a single node in order to minimise the number of inter-node references
- It should distribute the workload of an application evenly across the nodes in order to maximise the processing capability of the system.

3 Declustering

We used a variation on the similarity based approach proposed by [Liu and Shekhar, 1996] to decluster the objects in the data set. Unlike [Liu and Shekhar,

1996] who considered data placement in a single node multiple disk environment, we are targeting a share-nothing architecture having multiple nodes where each of node has its own CPU, memory and disk. Our algorithm differs from [Liu and Shekhar, 1996] in that we attempt to place objects that have a higher degree of similarity together in the disk of the same node in contrast to their goal of placing similar data items on different disks. Our definition of similarity states that two objects are more similar if they are accessed together in a *navigational* manner but less similar if the two objects can be accessed together in a *parallel* manner. This enables the algorithm to preserve a degree of parallelism while decreasing internode object references. A more formal definition of similarity for declustering is as follows:

$$S_d(o_i, o_j) = \alpha \left(w_{link}(o_i, o_j) + w_{link}(o_j, o_i) \right) - (1 - \alpha) w_p(o_i, o_j) \quad (1)$$

where α is a user defined weight that is assigned based on the relative importance of internode object references in reducing response time as compared to parallelism. A possible reason for increasing the value of this weight is the fact that in an environment where the communication overhead is very high you may place higher importance on reducing internode object references.

As the queries are run the appropriate frequency weights are incremented and stored for use by the placement algorithms. The frequency weights are pre-determined with respect to the algorithm. Therefore the queries are first run to train the system by incrementing dynamic frequency counters, then the information is used by the object placement algorithms when determining the placement of objects. The term $w_{link}(o_i, o_j)$ is used to denote the frequency of traversal of the pointer link from o_i to o_j . $w_p(o_i, o_j)$ denotes the frequency with which the two objects o_i and o_j can be processed on different nodes in parallel. Two objects are said to be accessed in parallel when the placement of the two objects determine the parallelism of query processing. In other words, in this study, $w_p(o_i, o_j)$ will be n if both o_i and o_j belong to the same typed set which is accessed n times. Otherwise, $w_p(o_i, o_j)$ will be treated as zero. The reason is for simplicity we have chosen to only consider partitioning of a set of objects belonging to the same type in this study. However w_p can be extended to consider objects belonging to the same set. The degree of declustering similarity of an object o_i with a partition P_j of objects is defined as follows:

$$S_d(o_i, P_j) = \alpha \sum_{o_k \in P_j} \left(w_{link}(o_i, o_k) + w_{link}(o_k, o_i) \right) - (1 - \alpha) \sum_{o_k \in P_j} w_p(o_i, o_k) \quad (2)$$

The declustering algorithm considers each object in the object base with a frequency counter greater than zero and places the object in the partition, P_j , that results in the largest value for $S_d(o_i, P_j)$.

Declustering Heuristic

The heuristic used in the declustering phase is a greedy approach in which the objects are first sorted into an array based on non-increasing frequency of reference. Then each object o_i is placed one at a time into the partition P_j which

has the largest declustering similarity value with the object o_i . In the case where a tie occurs, the object is placed in the partition on which the least number of ties have been assigned. The reason for this is that if objects involved in ties are distributed randomly or to a particular partition then there is potential for a bottle neck since nodes with abnormally large number of tied objects will get abnormally large number of remote page requests.

A simple method of determining the declustering similarity of an object with respect to a partition is to add the declustering similarity between each object in the partition and the currently considered object. This would not be feasible for large numbers of objects. We have developed a faster method that uses two hash tables to accumulate the information required to calculate the first summation in the declustering similarity partition equation. Both hash tables hash object identifier and partition number pairs. The way in which the two hash tables are used to determine similarity between an object and a partition is depicted diagrammatically in Figure 1. As shown in Figure 1, object O_5 is the object that is currently being considered for placement, while the other objects has been partitioned.

The first hash table indicates if an object exists in a partition. For each object o_i being considered the presence in partition P_j of each of the objects o_k referenced by o_i is determined by checking for the existence of an entry (o_k, P_j) in the first hash table. If the entry exists, then $S_d(o_i, P_j)$ is incremented by $w_{link}(o_i, o_k)$. The first term in the first summation of the declustering similarity equation is thus calculated.

The second hash table contains the set of object identifiers which is being referenced by objects already in the partition. The value field in this hash table is the accumulated frequencies of all objects o_k in the partition P_j which references the object o_i . This is the second term in the first summation of the declustering similarity equation defined in the previous section.

Pseudo code for the two hash table method of determining declustering similarity is presented in Algorithm 1.

4 Clustering

The clustering phase is composed of two steps. The first step clusters objects of the same type together. This benefits intra-operator parallelism since each node iterates through the portion of objects that reside in its own disk. Once clustered together based on type, the second step reduces remote page loads by clustering objects which are likely to be accessed by another node together.

The implementation of the first step is straight forward but the second is not as obvious. Our approach is to partition the objects to be clustered at a particular node into N partitions where each partition is reserved for a node. Each object is assigned to the partition which has the largest similarity value with respect to the node that the partition is clustered for. This requires a slightly different definition of similarity. The definition of clustering similarity between objects o_i and o_j is:

$$S_c(o_i, o_j) = w_{link}(o_i, o_j) + w_{link}(o_j, o_i) \quad (3)$$

Algorithm 1 A declustering algorithm

```
let  $N$  be the number of nodes;
let  $P_j$  and  $o_i$  be a partition and an object;
let  $type(o_i)$  be the base type of an object  $o_i$ ;
let  $load(P_i, T_j)$  be the load balancing factor for the node  $P_j$ 
for objects of type  $T_j$ ;
sort all objects with frequency greater than zero into non-increasing order
of frequency
for each object  $o_u$  in sorted order do
  for each partition  $P_j$ , for  $1 \leq j \leq N$  do
    for each object  $o_v$  that  $o_u$  references to do
      if  $(o_v, P_j)$  exists in hash table 1 then
         $S_d(o_u, P_j) \leftarrow S_d(o_u, P_j) + \alpha \times w_{link}(o_u, o_v)$ ;
      endif
    endfor
     $S_d(o_u, P_j) \leftarrow S_d(o_u, P_j) + \alpha \times$  value in hash table 2 for  $(o_u, P_j)$ ;
     $S_d(o_u, P_j) \leftarrow S_d(o_u, P_j) - (1 - \alpha) \times load(P_j, type(o_u))$ ;
  endfor
if no ties occurs then
  let  $P_k$  be the partition that has  $\max_{1 \leq i \leq N} (S_d(o_u, P_i))$ ;
else
  let  $P_k$  be the partition that has received least number of ties;
endif
 $P_k \leftarrow P_k \cup \{o_u\}$ ;
update hash tables;
update load balancing factors;
endfor
```

The term w_{link} is defined in the same way as in Equations 1 and 2. Note the term w_p present in Equations 1 and 2 is not required in this equation since clustering does not effect parallelism in the query processing techniques used in this document. The degree of clustering similarity of an object o_i with a partition P_j of objects is defined as follows:

$$S_c(o_i, P_j) = \sum_{o_k \in P_j} \left(w_{link}(o_i, o_k) + w_{link}(o_k, o_i) \right) \quad (4)$$

Clustering Heuristic

We reuse the two hash table technique developed for the declustering process. Since the clustering process follows declustering, the hash tables will have already been constructed. The algorithm for clustering is given in Algorithm 2. In this step we simply use the information in the two hash tables to calculate similarity and place the objects in the partition that returns the largest similarity. Note we do not need to break ties since placing objects involved in ties together in one partition is not necessarily worse than evenly distributing ties. Tied objects occur when more than one partition contain objects that either reference it or is referenced by it. One partition may reference many tied objects in which case

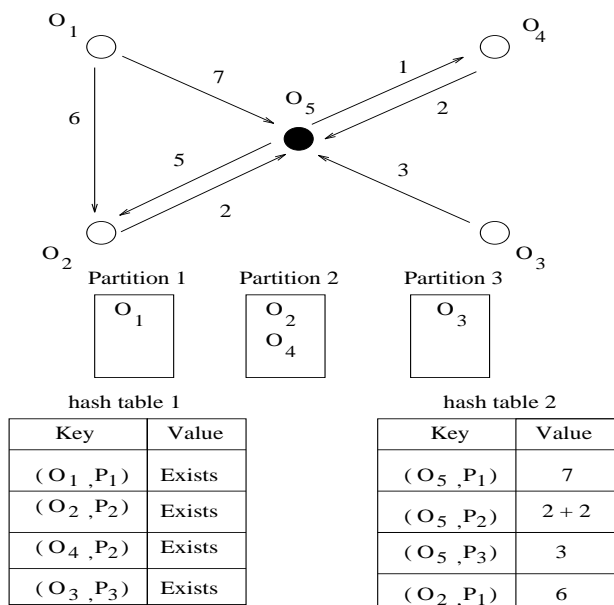


Fig. 1. Illustration of the two hash table technique

loosely grouping the tied objects together may be superior to evenly distributing the ties.

Once all the objects are clustered into partitions, the objects are stored into the database one partition at a time. If the similarity value of an object with respect to all other nodes is zero, then the object is placed in the partition reserved for the node that the object is on.

5 Query Processing

Intra-operator parallelism is used to process queries for our tests. Each node works on the objects which resides on its own disk. Thus the distribution of objects across the nodes defines parallelism. Four different queries were tested on the objects of a varying sized OO7 benchmark [Carey et al., 1993]. Varying the size of the OO7 benchmark allowed us to test the scalability of our algorithm by observing response time as the number of processors increase in proportion to problem size. As a base case a slightly modified medium sized OO7 benchmark was used for 32 nodes.

- **Query-1 (a range query):** The first query is a set based query which counts the number of atomic parts with build date above 1500.
- **Query-2 (a value-based join query):** The second query is based on query 8 of the OO7 benchmark. The query involves finding all pairs of documents and atomic parts where the document ID in the atomic part matches the ID of the document [Carey et al., 1993]. The query is processed by performing a value based hash join.

Algorithm 2 A clustering algorithm

```
let  $N$  be the number of nodes;
let  $n_j$  be node number  $j$ ;
let  $P_j$  be the partition for the  $j$ -th node;
for each node  $n_j$  for  $1 \leq j \leq N$  do
    for each object  $o_u \in n_j$  do
        for each partition  $P_k$  where  $P_k \neq P_j$  do
            for each object  $o_v$  that  $o_u$  references to do
                if  $(o_v, P_k)$  exists in hash table 1 then
                     $S_c(o_u, P_k) \leftarrow S_c(o_u, P_k) + \text{link}(o_u, o_v)$ 
                endif
             $S_c(o_u, P_k) \leftarrow S_c(o_u, P_k) + \text{value of the hash table 2 for } (o_u, P_j)$ 
            endfor
        endfor
    endfor
    if  $\max_{P_k \neq P_j} (S_c(o_u, P_k)) = 0$  then
         $P_j \leftarrow P_j \cup \{o_u\}$ ;
    else
        let  $P_k$  be the partition that has  $\max_{P_k \neq P_j} (S_c(o_u, P_k))$ ;
         $P_k \leftarrow P_k \cup \{o_u\}$ ;
    endif
endfor
endfor
```

- **Query-3 (a navigational query with sharing):** The third query is based on query 5 of the OO7 benchmark. Query 5 of the OO7 benchmark involves finding all base assemblies that use a composite part with a build date later than the build date of the base assembly. This query involves accessing shared objects. The OO7 benchmark is slightly modified for this query. The number of composite parts is increased from 500 (in original medium size OO7 benchmark) to 5000 and the size of the composite parts is enlarged. The reason for this is that when testing with only 500 small composite parts on 32 nodes with a 4 KB page size results in all composite parts fitting in 1 page per node. Remembering the object are first clustered by type. Therefore if you need at least one composite part object from every node then you end up loading all the pages in the system which contain composite parts. This is likely to happen when both the degree of sharing is high and a large proportion of the objects are selected by the query. Such a scenario would be difficult to optimise. We did test such a case (original medium sized OO7 benchmark) and the results showed our two phased approach did not offer any improvement in speed.
- **Query-4 (a navigational query without sharing):** The final query involves finding all composite parts with build dates earlier than its atomic parts build date. The OO7 benchmark was slightly changed here as well. The number of atomic parts per composite part was changed from 100 to 20. The reason for this change was the number of composite parts was increased ten fold thus the number of atomic parts needs to be decreased ten fold to keep the size of the database nearly the same size as the medium

sized OO7 benchmark. This query is a navigational query which does not share objects.

We call Query-1 and Query-2 set queries, and call Query-3 and Query-4 navigational queries. An important point to note is that there exists conflicts between the different queries. For Query-1 and Query-2, atomic parts need to be evenly placed on the nodes to allow a higher degree of parallelism but for optimal placement of atomic parts, Query-4 requires atomic parts to be placed together in order to decrease the number of internode traversals. Similarly optimal placement of objects for Query-3 requires composite part objects to be placed together but composite parts in Query-4 should be spread apart.

6 Overview of Test Bed

The hardware platform used is the 128 node Fujitsu AP1000 [Ishihata et al., 1990]. Each node has a 25 MHz SPARC CPU with 16MB of memory and 128 KB of direct-mapped cache memory. There are 16 disks in the system.

We have simulated a shared-nothing system on this hardware architecture. Each node of the system acts as both a client and server. The clients do the query processing and the server is responsible for satisfying remote page requests. When a client requires a page from another node it sends a request and the server on the remote node replies with the requested page. Due to the inability of the operating system to support threads, interrupt driven receive was not possible and instead an MPI [Sitsky et al., 1994] non-blocking receive was used to approximate an interrupt driven receive. The interrupt driven receive was simulated by the regular checking of the receive buffer to see if a request has been received. This may cause some inaccuracies since remote page requests may not be serviced immediately. However due to the small amount of computation between receive buffer checks the requests are delayed only for a short period of time and thus the inaccuracies are small.

The AP1000 uses a parallel file system, called HiDIOS [Tridgell and Walsh, 1995] which causes contention when two or more nodes attempt to read at once. This limitation can be overcome by de-stripping the disk. De-stripping involves writing fragments of the file for each node into particular positions in a file. The positions chosen cause the reading of a file for a node to be free from contention when the file is loaded onto the parallel file system. However this does not allow more than 16 nodes to be tested since there are only 16 disks on the AP1000. This second limitation was overcome by the adoption of an air-disk. The air-disk simulate reading from disk by first reading the entire file into a block of memory and then copying requested pages from the block into the application buffer on request. When a page is to be read from the air-disk, the system would delay for the amount of time equivalent to the disk I/O cost incurred when loading a page from disk. The disk I/O cost is derived by first running an experiment in which the queries (same as the queries to be used when running the experiments) are run on the disk de-stripped and recording response times of page loads. The

response times are then used to arrive at a model of the I/O costs involved in loading a page from disk into memory.

7 Performance Results

There are many factors effecting the performance of query processing. Object placement is only one among many. We have attempted to present the results in terms of comparisons between the various methods of object placement. The four queries outlined in Section 5 were first run to train the database, therefore to arrive at the values for the frequency counters required by the declustering and clustering algorithms. They were then run 10 times with the objects placed according to the various algorithms to obtain the results presented. When a query completes, the cache is flashed thus all results presented in this paper are derived from cold runs. Note: The labels on the graphs mean the following:

- RD: Random partitioning
- RR: Round Robin partitioning
- DN: Similarity declustering without page clustering (Algorithm 1).
- DC: Similarity declustering with page clustering (Algorithm 1 and Algorithm 2).

The value of α for DN and DC were both set to 0.9 since the queries run involved only a small amount of computation.

In addition to the results reported we have also conducted some other experiments which included varying training conditions and running conditions. The results also supported the findings presented below, namely the number of remote page loads stayed low while client workload skew also stayed low for DC with $\alpha = 0.9$.

Recently an enhancement has been made on the two phased approach outlined in this paper. This involves considering objects for placement which references or is referenced by the current object next instead of considering objects in order of decreasing frequency of traversal. This approach has yielded superior performance when compared to DC and DN. Unfortunately this enhancement was developed too late for this paper but will appear future publications.

At the start of every query a startup process is performed in which one server node sends a startup message to every client node. At the completion of every query a termination process is initiated in which every node sends a message to the server reporting the results of executing the query. Note both the startup and termination costs increases with the number of processors. Therefore the startup and termination processes are inherently non-scalable.

7.1 Response Time

Figure 2 depicts the total response time obtained from running all four queries together. Due to the fact the problem size is increased in proportion to the number of processors, a totally scalable system would be depicted by a constant response time with the increase in the number of processors. This is far from the case shown on Figure 2. The reason for this becomes evident when Figure 3

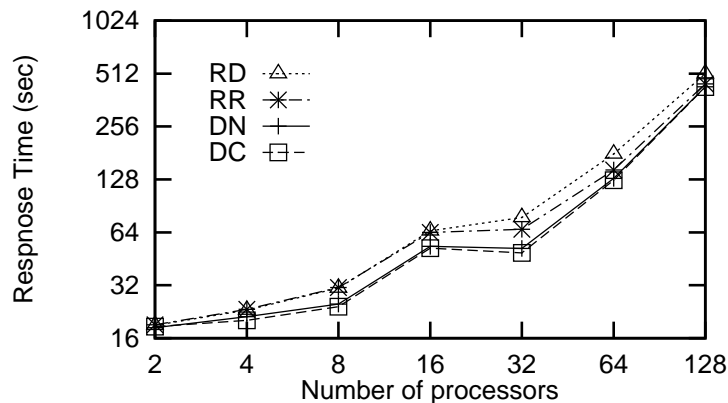


Fig. 2. Response time for running all four queries

is seen. Figure 3 depicts the individual execution times of the 4 queries run. It appears the value based join (query 2) is dominating the response time behaviour of the 4 queries being run. Query 2 involves all to all communication when the objects are hashed on there join attribute and redistributed. This process does not scale up well for the situation we tested due to the increase in communication costs when the number of processors is increased. For query 2 all declustering algorithms performed equally well. A reason for this is there is no navigational component in query 2 and therefore DC and DN only attempt to spread objects apart in-order to maximise parallelism. The round robin placement strategy already maximises parallelism since it evenly spreads objects across the nodes in a round robin manner. Another reason for the placement strategies to perform similarly for query 2 is the fact that it is the hash function which determines the amount of workload skew in the probing phase of the parallel value based join. Therefore it maybe concluded the object placement strategy employed has minimal effect on the response time of the value based join query investigated in this study. However for queries 3 and 4 where there is an navigational component to the query processing, DN and DC perform better relative to RD and RR. Also note in query 3 where there is sharing of objects the difference in performance between DC and DN is not large. This maybe due to the inability to cluster objects for more than one node when the object to be clustered, is shared by more than one node. However for query 4 where there is no sharing of objects, there appears to be noticeable difference between the performance of DN and DC. It is however encouraging to observe DC and DN did not perform noticeably worse than RR for queries 1 and 2 but did perform better than RR in queries 3 and 4 where navigation is present in the queries.

7.2 Number of Remote Page Loads and Client Workload Skew

As stated in the introduction the aim of the algorithms developed was to reduce the number of remote page loads and increase the degree of parallelism. The

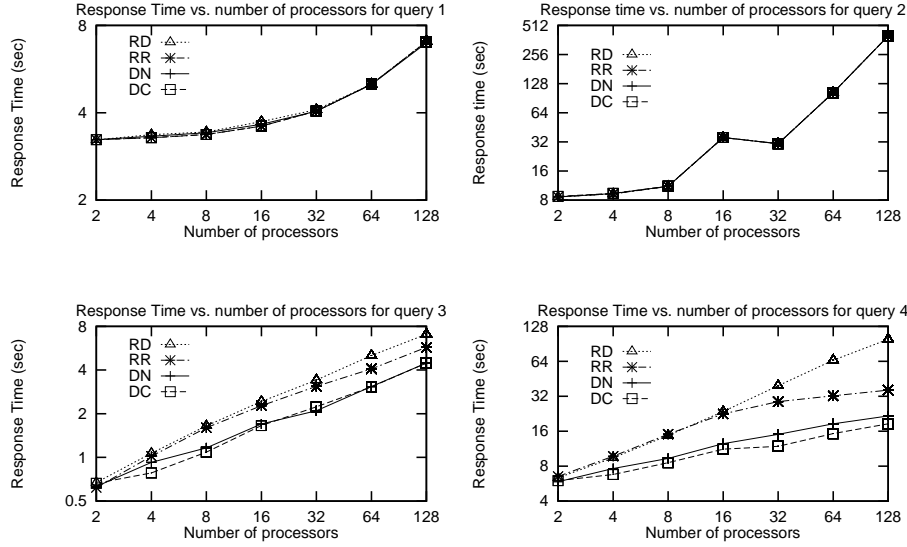


Fig. 3. Response time for individual queries

amount of client workload skew is a measure of the degree of parallelism. Figure 4 depicts how the algorithms performed for these two measures of placement quality. It appears DN and DC did reduce the amount of remote page loads by up to 10 times when compared to random and up to 5 times when compared to round robin. Also DC did perform noticeably better than DN which appears to indicate the clustering phase of the algorithm is potentially beneficial.

We defined client workload skew by:

$$\text{workload skew} = \frac{\max_{1 \leq i \leq N}(\text{work}(P_i)) - \min_{1 \leq i \leq N}(\text{work}(P_i))}{\max_{1 \leq i \leq N}(\text{work}(P_i))}$$

where $\text{work}(P_i)$ is the number of objects in the set of root objects processed for node P_i and N is the number of nodes in the shared nothing system. From the client workload skew graph depicted in Figure 4, DC and DN is shown to exhibit low client workload skew of around 0.005 which is considerably less than for random which is around 0.08. Also note DC and DN had the same amount of client workload skew. This can be explained by the fact that workload skew is only effected by the declustering phase of the algorithm.

These graphs indicates the declustering and clustering phases proposed in this paper does reduce remote page loads while preserving a high degree of parallelism.

8 Conclusion

Both the clustering and declustering of objects in a parallel object oriented database is important in reducing the number of remote page loads. The fact that the declustering plus clustering approach resulted in a large reduction in

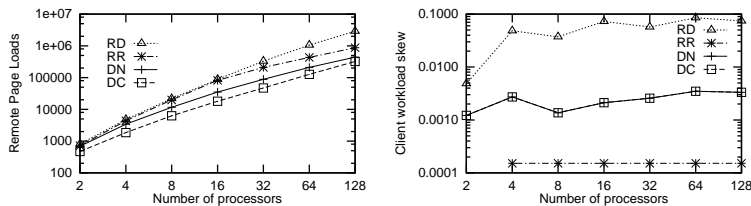


Fig. 4. Remote page load and client workload skew for running all four queries

the number of remote page loads while causing only a small amount of client workload skew is grounds for optimism. The object placement methods exhibit varying degree of scalability depending on the query being run. The relative performance of the different object placement algorithms is dependent on the particular type of query run.

Some areas of further study include investigation of different degrees of sharing of objects and how that effects the different object placement strategies, performance of algorithms with situations where network latencies are larger. Investigation of the performance of the algorithm with different query parallelisation methods. Testing the algorithm with more complex queries and data sets in the future is important for identifying situations where the algorithm does not perform well. When such situations are identified the algorithm may be altered in order to improve its ability to adapt to a wider range of query distributions.

Acknowledgement

We would like to acknowledge that this work was carried out within the Cooperative Research Center for Advanced Computational Systems established under the Australian Government’s Cooperative Research Centers Program. We would like to thank Steve Blackburn for his help with the design of the object store, help with the experimental setup and many useful and helpful suggestions. We would also like to thank Luke Kirby for helping in the implementation of the test bed and for providing helpful comments and suggestions, David Sitsky and David Walsh for their help in the implementation of the test bed, and Richard Walker for typographical assistance.

References

- [Carey et al., 1993] Carey, MJ, DeWitt, DJ, and Naughton, JF (1993). The OO7 benchmark. In *Proceedings of the 1993 ACM-SIGMOD Conference on the Management of Data*.
- [Chen et al., 1995] Chen, Ling Tony, Rotem, Doron, and Seshadri, Sridhar (1995). Declustering databases on heterogeneous disk systems. In *Proceedings of the 21st VLDB Conference*, pages 110–121.
- [Copeland et al., 1988] Copeland, George, Alexander, William, Boughter, Ellen, and Keller, Tom (1988). Data placement in bubba. In *Proceedings of the 21st VLDB Conference*, pages 99–108.

- [DeWitt et al., 1996] DeWitt, David J., Naughton, Jeffrey F., Shafer, John C., and Venkataraman, Shivakumar (1996). Parallelising OODBMS traversals: a performance evaluation. *The VLDB Journal*, 5(3):3–18.
- [Fang et al., 1986] Fang, M., Lee, R., and Chang, C. (1986). The idea of declustering and its applications. In *Proceedings of twelfth International conference on Very Large Databases*, pages 181–188.
- [Ghandeharizadeh et al., 1994] Ghandeharizadeh, S., Wilhite, D., Lin, K., and Zhao, X. (1994). Object placement in parallel object-oriented database systems. *Proceedings of the Tenth International Conference on Data Engineering*, pages 253–262.
- [Ishihata et al., 1990] Ishihata, H., Horie, T., Inano, S., Shimizu, T., and Kato, S (1990). CAP-II architecture. In *Proceedings of the First Fujitsu-ANU CAP Workshop*, Kawasaki, Japan. Fujitsu Laboratories Ltd.
- [Kim, 1990] Kim, K.C. (1990). Parallelism in object-oriented query processing. In *Proceedings of the Sixth International Conference on Data Engineering*, pages 209–217.
- [Liu and Shekhar, 1996] Liu, D. and Shekhar, S. (1996). Partitioning similarity graphs: A framework for declustering problems. *Information Systems*, 21(6):475–496.
- [Lu et al., 1994] Lu, Hongjun, Ooi, Beng-Chin, and Tan, Kian-Lee (1994). *Query Processing in Parallel Relational Database systems*. IEEE Computer Society Press.
- [Padmanadhan and Baru, 92] Padmanadhan, Sriram and Baru, Chaitanya K. (92). Data placement in shared-nothing parallel database systems. Technical report, The University of Michigan.
- [Sitsky et al., 1994] Sitsky, David, Walsh, David, and Johnson, Chris (1994). An efficient implementation of the message passing interface (MPI) on the Fujitsu AP1000. In *Proceedings of the Third Parallel Computing Workshop*, Kawasaki, Japan. Fujitsu Laboratories Ltd.
- [Thakore and Su, 1994] Thakore, Arun K. and Su, Stanley Y.W. (1994). Performance analysis of parallel object-oriented query processing algorithms. *Distributed and Parallel Databases*, 2(1):59–100.
- [Tridgell and Walsh, 1995] Tridgell, Andrew and Walsh, David (1995). The HiD-IOS file system. In *Proceedings of the Fourth Parallel Computing Workshop*, pages 52–63, London, UK. Fujitsu Laboratories Ltd.
- [Tsangaris and Naughton, 1991] Tsangaris, M. M. and Naughton, J. F. (1991). A stochastic approach for clustering in object bases. In *Proceedings of the ACM SIGMOD conference on Management of Data*, pages 12–21.
- [Zdonik and Maier, 1990] Zdonik, Stanley B. and Maier, David (1990). *Readings in Object-Oriented Database Systems*. Morgan Kaufmann.