

A TOOL FOR SUPPORTING THE TEACHING OF PARALLEL DATABASE SYSTEMS

E W Dempster, M H Williams, N T Tomov, A Burger, N Vassilopoulos and H Taylor.

E W Dempster, M H Williams, A Burger and H Taylor are with the Department of Computing and Electrical Engineering at Heriot-Watt University, Riccarton, Edinburgh, Scotland.

N T Tomov is with Infobrokers, Melbourne, Australia.

N Vassilopoulos is with the Greek Army, Athens, Greece.

This work received funding from the UK Engineering and Physical Sciences Research Council (EPSRC) under the PSTPA programme (GR/K40345) and from the Commission of the European Union under the Framework IV programme (Mercury project).

A TOOL FOR SUPPORTING THE TEACHING OF PARALLEL DATABASE SYSTEMS

E W Dempster, M H Williams, N T Tomov, A Burger, N Vassilopoulos and H Taylor.

Abstract– Parallel database systems are complex entities. As part of a course in a limited time scale, it is difficult to provide useful practical experience on these systems that provide deep insight into their behaviour and operation. This paper describes a tool for performance prediction which has been developed to aid the visualisation of parallel database systems and which is currently being used to support teaching. It enables students to experiment with different hardware and software configurations and to view the effects of changes on the performance of the system. It provides insight into how data can be placed among the nodes of a parallel machine according to predefined strategies, as well as manually, and provides feedback on the effect of these on throughput and response time. It is able to provide a good appreciation of the concepts in a relatively short period of time.

Index Terms – Parallel systems, parallel databases, performance prediction, DBMS.

I. INTRODUCTION

Parallelism and parallel computer systems form an important component of some courses in software engineering and computer science. Parallel database systems are the most important commercial applications running on parallel computers today, and present a particular challenge since these systems are complex and their behaviour is difficult to predict.

Not only has one to deal with the traditional problems of database design [1] to produce a set of normalised relations that provide efficient performance for a given set of queries, but one has also the additional problems associated with parallel systems. In particular, the relations need to be divided into fragments and distributed amongst the processing elements in such a way as to provide optimum (or at least good) performance. The latter process, of determining how best to fragment and lay out data on a parallel database configuration, is referred to as *data placement*. This can be done in various ways with a view to achieving a balance of activity on different processing elements, an even distribution of data across the disks or minimising the traffic between processing elements. Issues such as the way in which the system is configured, the sizes of caches and the sensitivity to changes in the balance of queries need to be considered. One may even need to revisit the original database design. A useful introduction to data placement can be found in [2].

This paper describes a tool called STEADY (System Throughput Estimator for Advanced Database sYstems) which we have developed to aid visualisation of the complex processes involved and support understanding of the factors affecting the performance of such systems. This tool is based on a collection of analytical modelling techniques and provides estimates of performance in terms of throughput, utilisation and response time for specified system configurations, workloads and data distributions.

The platform which STEADY currently models is the ICL Goldrush Megaserver [3]. This is a back-end parallel database server which is typical of shared nothing parallel systems. The Goldrush system hosts three different DBMSs – Parallel Ingres [4], Oracle 7 [5] and Informix XPS [6][7] – and these are all catered for in STEADY. All examples used in this paper use Informix XPS, except for those in section V which use both Informix XPS and Oracle 7.

This tool has been used as an adjunct to teaching part of a course on parallel databases and, hence, knowledge of parallel architectures [8] is assumed throughout this paper. STEADY enables students to experiment with the various factors and obtain a better understanding not only of the workings of such systems but also of the measures of performance, the sensitivity to change and the difficulties of system tuning. The next section briefly describes the design of the architecture of the tool. Section III presents an example transaction that is used throughout the rest of the paper. Section IV discusses data placement and some of the strategies that can be employed to distribute data across nodes and disks and

their effects on performance. Section V discusses some of the differences between the three DBMSs that are modelled by STEADY. This is followed by some conclusions.

II. STEADY

STEADY [9][10] is an analytical tool for estimating the performance of parallel relational database systems. Its architecture is shown in Fig. 1. It takes as input various sets of parameters including the hardware and DBMS configuration, details of the relations, the data placement strategy to be used in distributing the relations amongst the processing elements (PEs) and the execution plans of SQL queries, represented as annotated query trees. These query trees capture the order in which relational operators are executed and the method for computing each operator. Within STEADY, the annotated query trees undergo several transformations until they reach a form (the resource usage representation) which allows the prediction of response time and resource utilisation.

Apart from a graphical user interface, STEADY consists of eight major modules divided into four layers as follows:

- 1) **Application Layer.** This is concerned with the data relating to the specific database application under study and how it is distributed within the system. It consists of two modules: Profiler and DPTool. The Profiler is responsible for maintaining information on the properties of the relations, including base table profiles and estimates of numbers of rows resulting from operations. The DPTool determines how each table is fragmented and how the fragments are assigned to the disks attached to the different PEs of the system.
- 2) **DBMS layer.** This models the processes specific to the database system and comprises three modules: Query Paralleliser, Cache Model and Modeller. The first of these determines how queries are decomposed into basic operations and what parallelisation strategies are to be applied. This follows the mapping of a query to sub-queries which are sent to appropriate sets of PEs by the DBMS. The Cache Model Component models the behaviour of the cache and provides an estimate of the cache hit ratio for pages from different tables (further details can be found in [11]). Finally, the Modeller uses the output of these various modules to produce a workload profile in terms of the basic operations to be executed on each PE.
- 3) **Machine Layer.** This consists of a single module, the Evaluator, which uses the information on the basic operations to be performed on each PE together with a hardware platform model to determine resource usage, system bottlenecks and estimated maximum throughput.
- 4) **Response Time Layer.** This comprises two modules, Queue Waiting Time Estimator and Response Time Estimator, which determine the queue lengths at the various resources involved and from these the response times for transactions at particular transaction arrival rates. Further details can be found in [12].

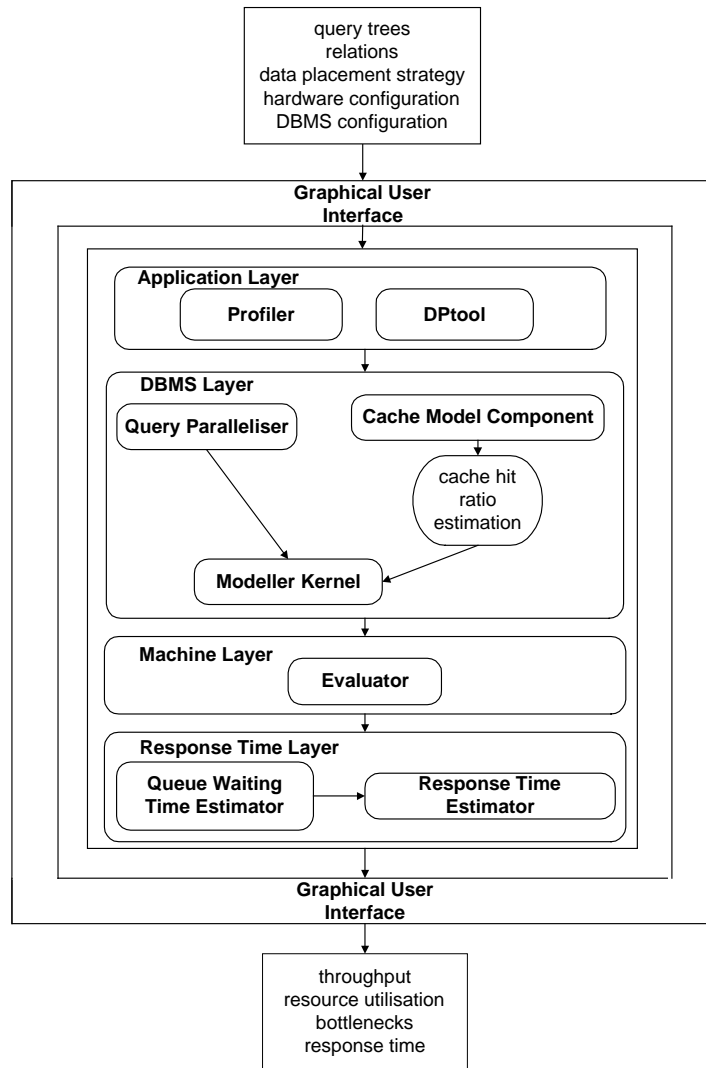


Fig. 1 STEADY architecture

III. EXAMPLE

To illustrate the features of the tool, a simple example will be used. The example employs a standard (TPC-C benchmark [13]) set-up, involving 10 warehouses with 3000 customers per warehouse. The query is selecting the customer identity number (c_id), district identity number (d_id) and the warehouse identity number (w_id) of all rows (records) in the *customer* table that match the customer identity number, district identity number and the warehouse identity number that are input to the query by the user. The tables are fragmented by hashing on the chosen column (attribute). For example, the *customer* table is fragmented into 10 fragments by hashing on the w_id column. The query used is:

```
int C_ID, D_ID, W_ID;
SELECT  $c\_id$ ,  $d\_id$ ,  $w\_id$ 
FROM customer
WHERE  $c\_id = C\_ID$  and
       $d\_id = D\_ID$  and
       $w\_id = W\_ID$ ;
```

This simple query employs a full table scan of the *customer* table. Knowledge of the *customer* table fragmentation is used to determine which fragments need to be scanned. The parameters C_ID, D_ID and W_ID may be varied for each query fired.

IV. DATA PLACEMENT AND PERFORMANCE

Given a set of relations and a set of queries to be executed against these relations with an appropriate relative frequency associated with each query, the problem of data placement is that of deciding how to fragment the relations and distribute them amongst the disks of the system in order to produce optimal (good) performance.

A number of techniques have been developed to help to automate this process. There are three main issues to placing data. Firstly, one has to decide how the relations are to be fragmented into chunks (partitions). Secondly, one must allocate these chunks to the nodes of the parallel machine in such a way as to obtain optimal (or at least good) performance. Finally, one may need to determine how to redistribute the data after insertions and deletions have unbalanced the original set up.

The first process, fragmentation or declustering, can be performed in various ways. One of the most common methods is a hash-based scheme on the key attribute. This involves applying a hash function to the key attribute value of each row and then allocating each row to the appropriate chunk on the basis of its hash value. Hashing can be done on other attributes and combinations of attributes and, of course, other functions can be used to decide how to create the chunks.

Placement and redistribution are performed by applying some rules or functions to the chunks of data in order to decide which nodes to assign them to. There are many strategies for doing this. The ones we will look at are:

- Size – The basic idea of this approach is to distribute the data in such a way that equal amounts of data are assigned to each node. For each table a list of fragments to be allocated to nodes is sorted in order according to size. Then the fragment at the top of the list of the largest table is removed from the list and assigned to the node with the most free disk space available. This is repeated until the list is empty. This process is then repeated for each table in the same way until all fragments have been allocated.
- Full, Semi and Abstract Round-Robin – In a round-robin approach the items to be distributed are allocated one at a time to nodes so that each node ends up with a similar number of items. In the case of abstract round-robin the items allocated are the rows of data and the result is that each node has about the same number of rows. In full and semi round-robin it is the chunks that are allocated so that each node ends up with about the same number of chunks. The only difference between full and semi round-robin lies in where they start and in which order the chunks are placed.
- Apers [14] – This is a network traffic based strategy which is aimed at minimising the delay incurred by network communication for transferring data between nodes. This approach can either be applied to rows of data (unit apers) or to relation fragments (fragment apers). It uses a

combination of a greedy algorithm and a heuristic approach applied to a network in which each fragment or unit is treated as a node. Each edge represents the cost of transmission between two nodes. The pairs of nodes are evaluated on the basis of transmission costs between them and the pair of nodes with the highest utilisation is selected and united. This is repeated until the network is unified with the actual network of the parallel machine.

- Bubba [15] – This is an access frequency based strategy which aims to balance the load on the nodes. The frequency of access to a table is defined as the heat of the table and the quotient of the table's heat divided by its size is referred to as the temperature of the table. The heat is used to determine the number of nodes across which to spread the tables and the temperature is used to determine whether a table should be resident in the cache. The strategy then assigns chunks of tables to nodes taking account of each node's temperature and heat. It puts relations in a list ordered by their temperatures and assigns the chunks in the list to nodes using a greedy algorithm to ensure that the heat of each node is about the same.
- Hua [16] – This is a size based strategy which attempts to balance the loads of nodes during query processing. Tables are partitioned into large numbers of cells using a grid file structure. These cells are then assigned to the nodes using a greedy algorithm to balance the data load on each node. To begin, all the cells of all the tables are placed in a list in size order. The cell at the top of the list is allocated to the node with the most space remaining on its disks and this cell is removed from the list. This is repeated until the list is empty. The difference between this strategy and the size strategy described above is that all of the cells (chunks) of all tables are placed on the list in Hua's approach whereas each table is dealt with in turn in the size-based strategy.
- EDS [2] – This is a variant of Apers placement method. It pre-assigns the clusters of tables (each cluster contains tables which need to be located) to groups of nodes, taking into account the space left on the nodes, so that each cluster is mapped to a set of nodes. The idea is to find a balance between the work load of the nodes and an equitable distribution of space. Then the chunks within each cluster are assigned to individual nodes in the set using the Apers method.

The widely differing effects of these strategies can be seen using STEADY by selecting different data placement strategies and viewing the resulting placement and the overall performance.

For the TPC-C example in the previous section, Fig. 2 shows two screen shots of the DPTool monitor that illustrate two data placements for the same machine configuration (7 nodes). (a) shows the result for a size placement strategy while (b) gives the result for a full round-robin placement. Each relation is represented by a separate colour and the fragments of each relation are numbered. Since the sizes of fragments are very variable, the fragments cannot be drawn to scale linearly as otherwise larger fragments would obscure smaller ones. On the right hand side of the screen a PE may be selected, and the details of the fragments for that PE will be displayed. If a particular fragment is selected, details of that fragment will be presented beneath this. In the top figure the *customer* table is number 5 (white), which has four fragments on PE2 and one fragment on each of the others. The highlighted fragment is from PE2, table 5 (*customer*) and the fragment number is 3. It has 3000 tuples and, as can be seen, it

was fragmented at the PE level by employing a hash function on the *c_w_id* attribute. The placement used was size and all of its *c_w_id* values are 3.

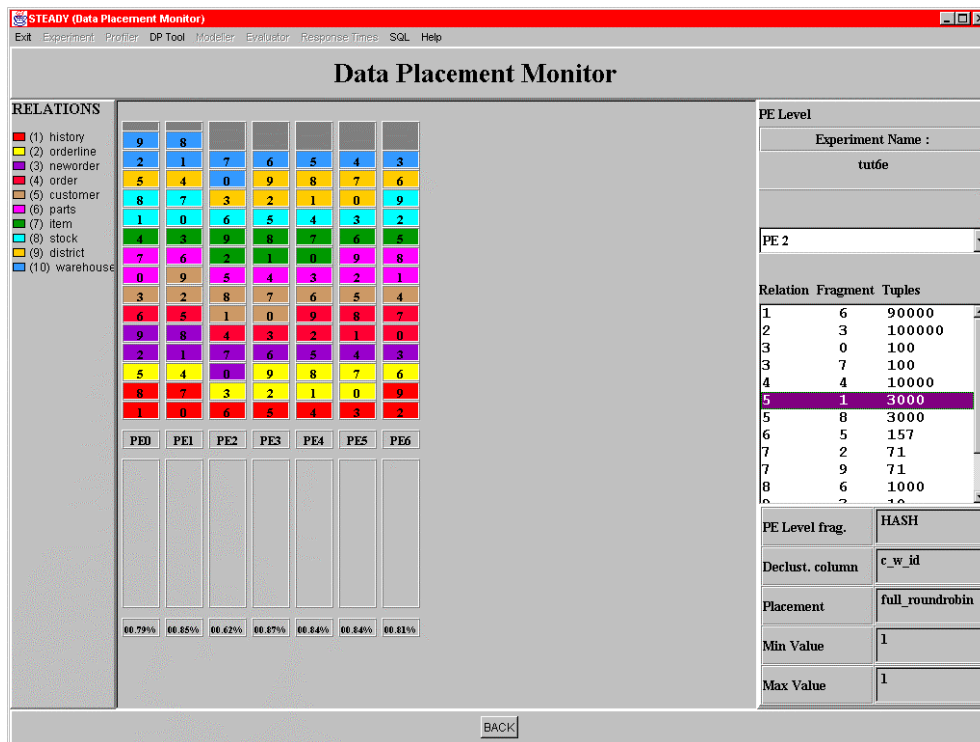
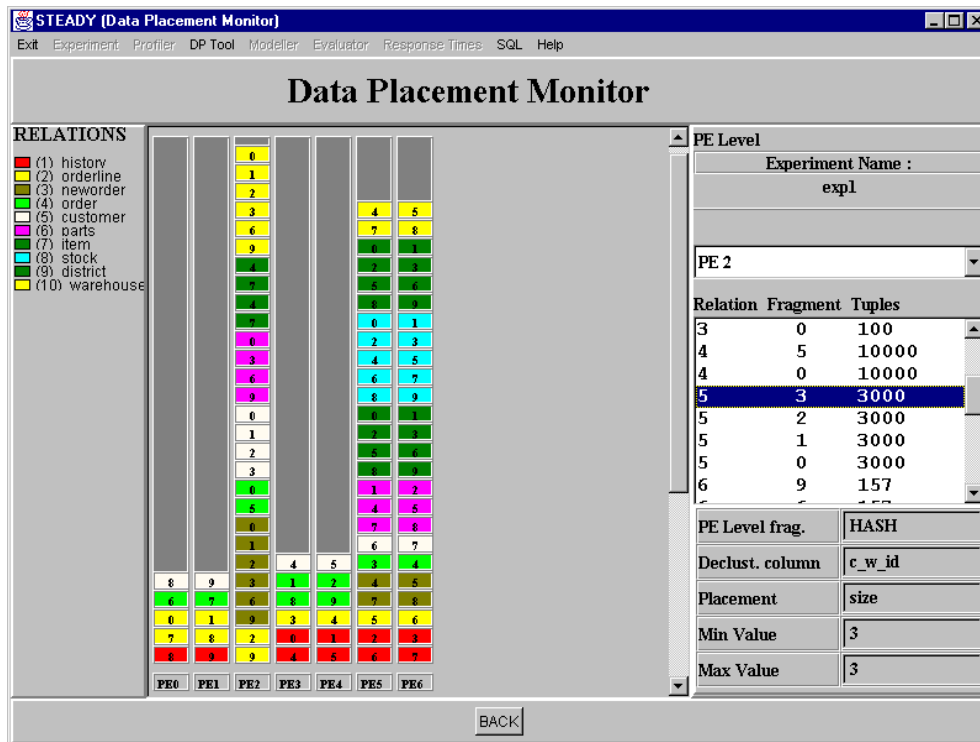


Fig. 2 Data placement comparison a) size b) full round-robin

The details of any fragment can be viewed by highlighting it. There is also a disk placement viewer available in the tool that shows how the node fragments are distributed across the disks of each node. In this example the fragments on each node are distributed across its disks according to the size strategy. The lower figure shows the DPTool monitor for the same experiment except that the data placement is

full round-robin rather than size placement. In this case the placement appears more balanced. However, the full round-robin strategy does not consider the sizes of the chunks being placed. Thus although the number of fragments on each PE is more balanced, the total amount of data on each PE is not.

The results viewer enables one to inspect the performance of the system in terms of the maximum throughput achievable, the bottlenecks and the resource utilisation of PEs or disks. This may be used to view results from a single experiment or to compare the results for two different experiments so that one can compare what is happening when different data placements are used for the same application. Fig. 3 illustrates the comparison between the two experiments whose data placements were shown in Fig. 2. Fig. 3 shows that PE2 disk 2 in experiment 1 has a utilisation that is twice that of experiment 2 which would lead one to expect that the throughput of experiment 2 would be much better than that of experiment 1. This is not the case as can be seen from the figures on the right hand side of the figure, where the round-robin placement (experiment 2) provides a marginally better maximum throughput (6.61 transactions/sec) than the size placement (6.5 transactions/sec). The reason that this is not much better is that although the size placement in experiment 2 has improved the loading on disk 2 of PE2, there are other disks on other PEs which have to do more work in experiment 2 than in experiment 1. This is down to the size of the *customer* table fragments which when fragmented mean both experiments end up with disks that have two fragments and are the bottlenecks. The minor difference in throughput between the two experiments is due to the slight difference in the size of the *customer* table fragments.

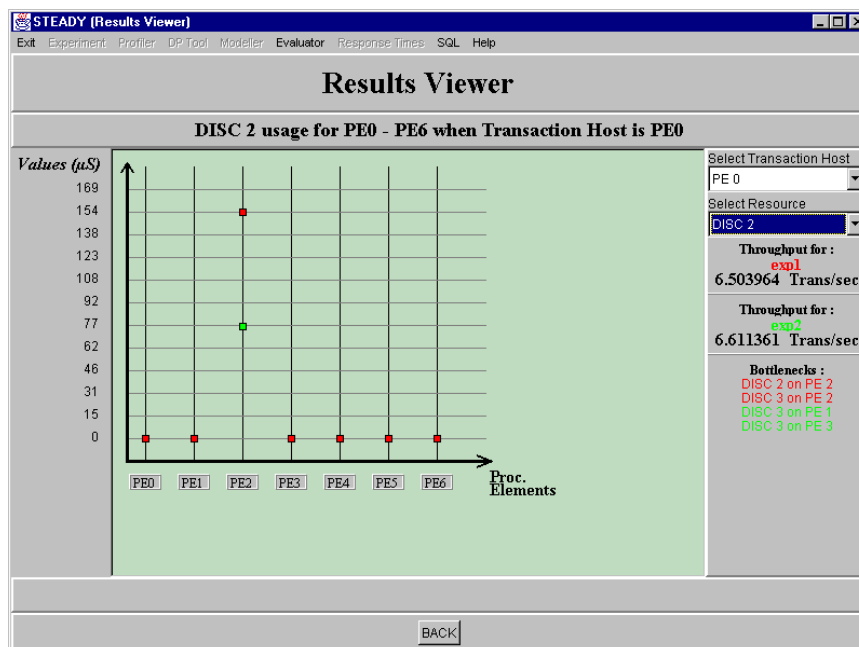


Fig. 3 Results viewer

As previously discussed there are many data placement strategies. Fig. 4 shows a graph of throughputs for the data placement strategies discussed earlier applied to the configuration in experiment 1.

Placement Strategy

- 1 = Size
- 2 = Full Round-Robin
- 3 = Abstract Round-Robin
- 4 = Semi Round-Robin
- 5 = Fragment Apers
- 6 = Unit Apers
- 7 = Bubba
- 8 = Hua
- 9 = Eds

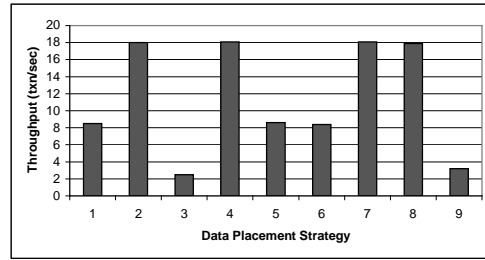


Fig. 4 Throughputs for various data placements

As can be seen from Fig. 4, the throughput can vary dramatically depending on the choice of data placement strategy selected. Strategies 2,4,7 and 8 all give a throughput of about 18 transactions per second, whereas strategy 3 gives a throughput of only 2.5 transactions per second. In addition to these standard placement strategies, the user may create any desired placement by manually redistributing fragments between nodes by using the facility provided in STEADY.

Another aspect of performance is the response time obtained for different queries. The response time viewer provides estimates of the average response times for different queries for different arrival rates. Fig. 5 illustrates the response time curves of two queries. The left-hand graph shows the response time in seconds for the query in experiment 1 and the right-hand graph shows the response time for a different query. The user may select what to show on the graphs from the check boxes at the top left-hand of the screen (any combination of points, lines and actual values).

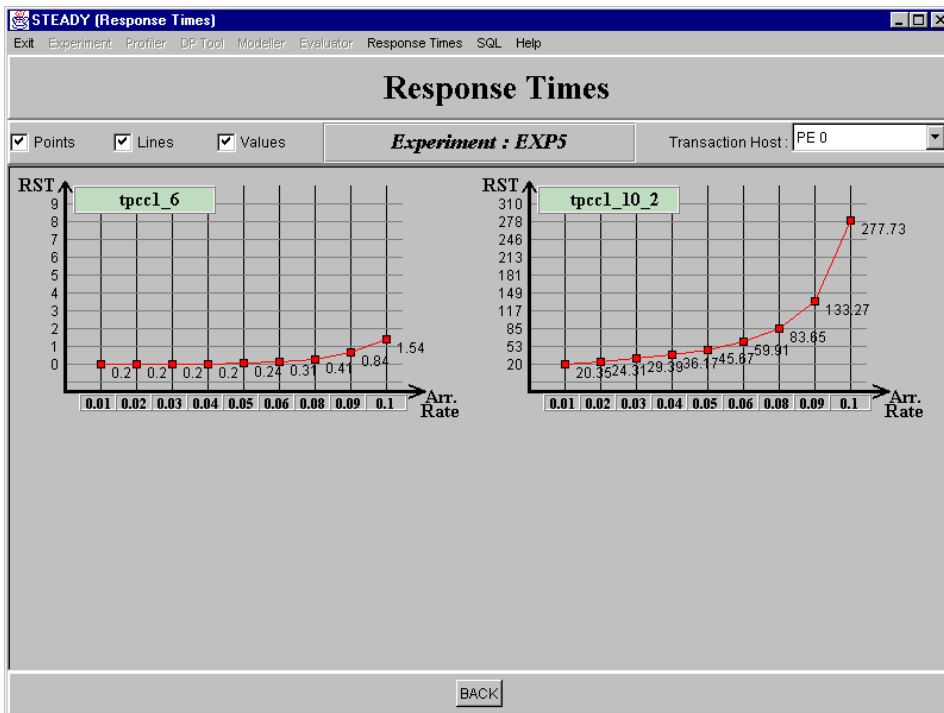


Fig. 5 Response time viewer

By changing the input parameters one can experiment with the system configuration, such as the architecture of the machine (size and number of disks, PEs and memory), the fragmentation of the tables involved (including disk fragmentation), the data placement strategies used and the workload itself (including tables, queries and the query mix). To summarise, one has a complex system with many parameters that may be varied in order to determine an appropriate architecture or configuration together with a data placement for a particular application.

V. COMPARISON OF DIFFERENT DBMSs

Another factor affecting performance is the actual DataBase Management System (DBMS) used. At present STEADY models the three major parallel DBMSs currently ported to the ICL Goldrush Megaserver. These three DBMSs are Ingres Cluster, Oracle 7 Parallel Server and Informix Online XPS. To choose an appropriate DBMS it is necessary to understand some of the differences between them.

Ingres Cluster utilises inter-query parallelism. Each processing element (PE) can run a set of Ingres Cluster processes, which include DBMS server process, recovery and archiver processes. A transaction executed on one PE can access the data stored on disks attached to other PEs through a distributed file system.

Oracle 7 Parallel Server also utilises inter-query parallelism, although its parallel query option provides intra-query parallelism including both inter- and intra-operator parallelism under some restrictions. An Oracle instance on one PE can also access data stored in disks attached to other PEs through an underlying distributed file system. Data placement is transparent to the users. Each Oracle instance has several background processes including the DBWR which manages database buffer cache by writing all dirty buffers to the data files using an LRU algorithm.

Informix Online XPS utilises inter- and intra-operator parallelism by dividing time-consuming query trees into a number of subtrees that are then sent to different PEs for executing. It takes a shared-nothing approach to managing data to minimise operating system overhead and reduce network I/O. Data operations are executed on the PEs where the required data is stored.

The three parallel DBMSs also use different types of access methods.

A. *Cache and Lock Management*

Ingres Cluster utilises the Goldrush Distributed Lock Manager (DLM) facility to handle concurrency control for a multiple PE configuration. It has two levels of lock: table and page levels. Cache coherency is achieved through the process in which one PE modifies pages and then writes these to disk on committing and another PE reads these modified pages from disk through the distributed file system. The pages cannot be read until the transaction commits and the locks are released.

Oracle 7 Parallel Server has a more complex locking mechanism using a technology called *Parallel Cache Management*. PCM takes advantage of the DLM to co-ordinate the accesses to various resources. It guarantees cache coherency by ensuring that the master copy of a data cache block in one instance's buffer has identical copies in the buffers of other instances. It also has row locks which operate independently of PCM locks. A PCM lock can be released by an instance, while row locks on the same data blocks are still held by the same instance.

As Informix Online XPS uses the principle that data operations are always executed where data resides, the Goldrush DLM is not involved. Lock management is only at the local level of co-server. The cache management is also similar to that in a single instance DBMS, as no cache coherency is required.

B. Disk I/O and Logging

The current version of parallel Ingres supports *fast-commit* in a single node configuration. In multi-node configurations, *group commit* is used to minimise the frequency of disk write operations.

In Oracle, apart from checkpoints, a full cache or too large a number of dirty blocks in cache, DBWR also performs disk write operations when a dirty cache block held by one instance is requested by another instance.

Informix Online XPS has dedicated page cleaner threads that manage the writing of dirty buffers to disk when the number of dirty buffers reaches a threshold, when there is no unmodified buffer available in the cache, or when a checkpoint is initiated.

C. Query Parallelisation

Goldrush Ingres Cluster does not support parallelisation within queries. It simply distributes transactions across multiple PEs.

Oracle 7 Parallel Server has a parallel query option that decomposes a query into sub-tasks and executes these in parallel.

Informix Online XPS uses SQL operators to divide a query tree into subtrees that can be executed in parallel across co-servers. It takes the data provided, repartitions it, and distributes the partitioned results along with instructions for executing the next SQL operations on multiple co-servers for processing.

D. Example

As an example of the difference in performance between DBMSs, consider an experiment run on STEADY for both Informix and Oracle. The transaction used was a form of the TPC-B benchmark [17] in which a customer performs a transaction at a bank. This involves updating the teller's and the branch's information, inserting a record (row) into the history table and updating the customer's account as well as returning their balance at the end of the transaction. The transaction is:

Inputs

Account_id, Branch_id, Teller_id, Delta, Time_stamp

End_Inputs

```
SELECT          account_balance
FROM            account
WHERE           account_id = Account_id

UPDATE          account
SET             account_balance = account_balance + Delta
WHERE           account_id = Account_id

UPDATE          teller
SET             teller_balance = teller_balance + Delta
WHERE           teller_id = Teller_id and branch_id = Branch_id

UPDATE          branch
SET             branch_balance = branch_balance + Delta
WHERE           branch_id = Branch_id

INSERT INTO     history
VALUES          Account_id, Branch_id, Teller_id, Delta, Time_stamp
```

Fig. 6 shows the response time curves for this transaction run on Informix and on Oracle over the same transaction arrival rate range. Here the response times for Oracle are only slightly greater than those for Informix since the dominant cost in this transaction is the reading in of the rows to search for the correct tuples to update. The reading costs for both DBMSs are similar and the difference is mainly due to the larger writing costs for Oracle. The difference is small at the low end of the range but becomes more pronounced as the rate increases and queues start to build up in the system.

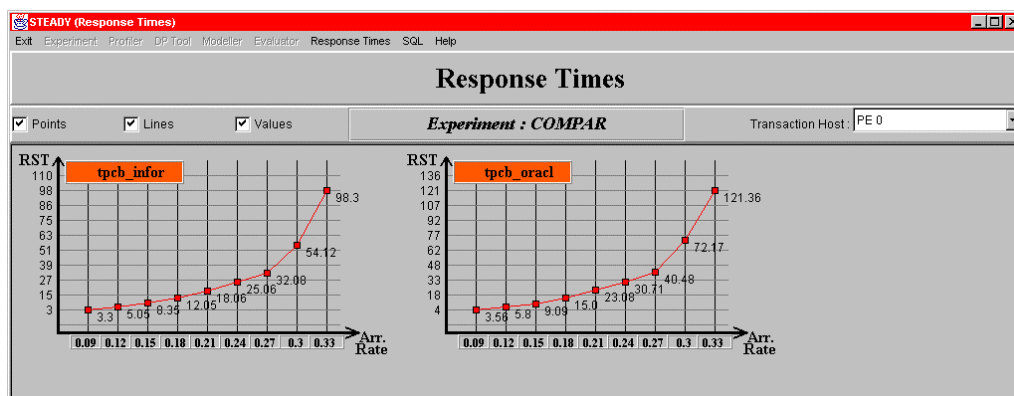


Fig. 6 Informix and Oracle comparison

VI. CONCLUSIONS

STEADY is a performance prediction tool for parallel relational databases that provides a simple but effective means of experimenting with these complex systems and of visualising the effects of different

decisions on the performance. It predicts the throughput, utilisation, bottlenecks and response time for both complex queries and mixed workloads.

This paper describes some of the facilities and illustrates how the tool can be used as an adjunct to teaching to assist students to understand the factors that influence the performance of a parallel database system and the effects that they can have. An insight into the importance of data placement is presented along with a brief comparison of DBMSs. The tool is designed to illustrate concepts in application sizing, capacity planning and performance tuning.

It has been used in teaching at final year/postgraduate level for two years to courses of about forty students. Students were enthusiastic about using STEADY but comparison to previous years is not possible as there was no practical element to the teaching of parallel databases prior to using STEADY. Measuring the difference in students performance is also complex as the examinations involved have an element of choice for the student in the questions tackled and not all elements of the course are examined in every examination but the overall impression is that the understanding of parallel databases and their concepts have improved with the introduction of using STEADY in practical classes.

To assist the students and any other interested parties, in their understanding of the STEADY tool, a number of materials are available for downloading at <http://www.cee.hw.ac.uk/Databases/down.html>. The items available are an instruction document (Word) on how to install STEADY on a PC, two movies (avi) which show the user how to start using the tool, the STEADY tool itself, some tutorial files to run on STEADY and a user manual (Word). Online versions of the tutorial handouts and user manual are also available at the same location.

ACKNOWLEDGEMENTS

The authors wish to thank Prof. Phil Broughton, Arthur Fitzjohn and Monique Mitchell of ICL for their support and assistance in developing STEADY.

REFERENCES

- [1] R. Elmasri and S. B. Navathe, "Fundamentals of Database Systems", 3rd Edition, Addison Wesley, California, 1999.
- [2] M. H. Williams and S. Zhou, "Data Placement in Parallel Database Systems", Parallel Database Techniques, IEEE Computer Society Press, 1997.
- [3] P. Watson and G. Catlow, "The architecture of the ICL Goldrush MegaServer", ICL Systems Journal, vol. 10, no. 2, <http://www.icl.com/sjournal/v10i2/v10i2a1.html>, November 1995.
- [4] The ASK Group Ltd., "ASK Open (INGRES) database administrator's guide", The ASK Group Limited, Bury St. Edmunds, UK, 1994.
- [5] Oracle Corporation, "Oracle7 Parallel Server Concepts & Administration", Release 7.3, Oracle Server Products, California, USA, 1996.
- [6] Informix Software Inc., "Informix-OnLine Dynamic Server & Administrator's Guide", Informix Press, California, USA, 1994.
- [7] Informix Software Inc., "Informix-OnLine extended parallel server for loosely coupled cluster and massively parallel processing architectures", Informix White Paper, <http://www.informix.com>, 1998.
- [8] J. L. Hennessy and D. A. Patterson, "Computer Architecture A Quantitative Approach", 2nd Edition, Morgan Kaufmann Publishers, Inc. San Francisco, California, 1996.

- [9] M. H. Williams, E. W. Dempster, N. T. Tomov, C. S. Pua, H. Taylor, A. Burger, J. Lu and P. Broughton, "An analytical tool for predicting the performance of parallel relational databases", *Concurrency: Practice and Experience*, Sept. 1999.
- [10] S. Zhou, M.H. Williams and H. Taylor, "Practical throughput estimation for parallel databases", *Software Engineering Journal*, vol.11, no.4, 255- 263, Jul 1996.
- [11] M. H. Williams, S. Zhou, H. Taylor, N. Tomov, A. Burger, "Cache Modelling in a Performance Evaluator for Parallel Database Systems", *Proceedings of the Fifth International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, IEEE Computer Society Press, 46-50, 1997.
- [12] N. Tomov, E. Dempster, M. H. Williams, P. King and A. Burger, "Approximate estimation of transaction response time", *The Computer Journal*, Volume 42, No. 3, 1999.
- [13] F. Raab, "Overview of the TPC benchmark C: a complex OLTP benchmark", in J. Gray (ed.), *The benchmark handbook for database and transaction processing systems*, 2nd edition, 131-267, 1993.
- [14] P. Apers, "Data allocation in distributed database systems", *ACM Transactions on Database Systems*, vol. 13, no. 3, 263-304, Sept. 1988.
- [15] G. Copeland, W. Alexander, E. Boughter, and T. Keller, "Data placement in Bubba", *Proceedings of ACM SIGMOD Conference*, 1988.
- [16] K. Hua and C. Lee, "An adaptive data placement scheme for parallel database computer systems", *Proceedings of the 16th VLDB Conference*, Brisbane, Australia, 493-506, 1990.
- [17] J. Gray (ed), "The Benchmark Handbook for Database and Transaction Processing Systems", *Transaction Processing Performance Council (TPC)*, 2nd edition, Morgan Kaufmann Publishers, Inc. San Francisco, California, 1993.