An Analytical Tool for Predicting the Performance of Parallel Relational Databases

M H Williams, E W Dempster, N T Tomov, C S Pua, H Taylor, A Burger, J Lü and
P Broughton

M H Williams, E W Dempster, N T Tomov, C S Pua, H Taylor and A Burger are with the Department of Computing and Electrical Engineering, Heriot-Watt University, Riccarton, Edinburgh, Scotland, EH14 4AS, UK

J Lü is with the School of Computing, Information Systems and Mathematics, South Bank University, Borough Road, London, SE1 0AA, UK

P Broughton is with International Computers Limited, High Performance Technology, Wenlock Way, West Gorton, Manchester, England, M12 5DR, UK

Keywords: Performance prediction, parallel DBMS, analytical model, validation.

SUMMARY

The uptake of parallel DBMSs is being hampered by uncertainty about the impact on performance of porting database applications from sequential to parallel systems. The development of tools which aid the system manager or machine vendor could help to reduce this problem. This paper describes an analytical tool which determines the performance characteristics (in terms of throughput, resource utilisation and response time) of relational database transactions executing on particular machine configurations and provides simple graphical visualisations of these to enable users to obtain rapid insight into particular scenarios. The problems of handling different parallel DBMSs are illustrated with reference to three systems - Ingres, Informix and Oracle. A brief description is also given of two different approaches used to confirm the validity of the analytical approach on which the tool is based.

I. INTRODUCTION

In recent years there has been general agreement that the main commercial application of parallel computer systems in the future will be that of databases. The inherent parallelism in relational databases is well suited to parallel computer technology and commercial interest in the use of parallel computers for running relational database systems has been growing [1]. A number of well known commercial DBMSs produced by vendors such as Oracle [2], Informix [3], Ingres [4], Sybase [6] and DB2 [5] have been adapted to run on parallel systems and are now available on SMP and/or MPP machines.

Despite this, the uptake of parallel database systems by end users has been disappointingly slow. This reticence may be partly due to past experience in moving to new technology too quickly and acting as the guinea pigs for vendors to debug and refine their systems. It may also be partly attributable to uncertainty in the effects on performance of moving particular applications from sequential to parallel systems. Certainly, commercial institutions which have applications which might warrant the move to parallel systems need reassurance that such systems will deliver both performance and reliability.

Part of this problem can be addressed through the development of tools to predict the performance of parallel database systems. Such tools could be used to assist a user in determining whether a given hardware/software configuration will meet their performance requirements. They could be used to determine how data might be distributed to obtain good performance or how performance will vary as changes occur in the volume of data or the balance of queries. They could also be used to tune the performance of systems once installed.

In addition to the ability to predict the performance of a parallel database system, such tools rely on usable graphical interfaces which provide simple visualisations of different aspects of the performance of the system to enable the user to understand rapidly what is happening in any particular scenario.

This paper describes a tool called STEADY which has been developed to assist the user in these various tasks. One aim in its development was to produce a generic tool which could handle different DBMSs running on different parallel platforms. Thus far the work has been focused on a single platform, the ICL GoldRush Megaserver [17], and the three parallel DBMSs which it supports - Ingres, Informix and Oracle.

The approach used by STEADY is an analytical one. Given a particular workload running on a specified configuration with a specific data distribution, it determines the performance characteristics in terms of maximum throughput, resource utilisation and response time. In order to do this, a detailed knowledge of each DBMS was required as the accuracy of the tool depends upon how closely it models the actual working of the system. However, as some of the algorithms used are commercial secrets of the DBMS vendors, some approximations of these have had to be made with the help of ICL, the vendor of the parallel platform used.

The remainder of the paper is organised as follows. The next section provides a brief background to related work in developing tools to estimate the performance of parallel relational database systems. A description of the design and architecture of STEADY is given in section III. Section IV considers some aspects of the three DBMSs which STEADY currently handles and illustrates the considerable differences between the three systems. Section V describes two approaches that have been employed to confirm the validity of the analytical approach used by STEADY and discusses the results obtained. Section VI provides a summary and conclusion.

II. Related Work

There are a number of performance tools for parallel relational databases in the market today. Many of these are concerned with performance measurement. These tools monitor the DBMS operation and provide information on the way in which the system is being used. Examples of these tools include Digital's ECO Tools [7], the Patrol DB-Log master by BMC software [8] and the Ingres DB_Maximiser by DB LAB PTY Ltd [9]. On the other hand, there are relatively few tools which predict the performance of DBMSs running on different platforms. Those that exist vary in complexity from a simple set of cost formulae to a detailed simulation of the DBMS.

The DB2 Estimator [10] project at IBM has produced an analytical performance estimation tool, designed specifically for the relational database system DB2 for OS/390 V5. It runs on a PC and calculates estimated costs using formulae obtained from an analysis of real DB2 code and performance measurements and is reasonably accurate in its predictions.

The Oracle System Sizer V3.0 [11] project at Oracle has, in conjunction with HP and Dell, produced an analytical tool which sizes hardware configurations for Oracle database applications. At present there are two versions, one which predicts configurations for HP NetServers and the other for Dell PowerEdge servers, both running Windows NT. Oracle are planning versions for additional hardware types in the future.

SMART(Simulation and Model of Application based Relational Technology) [12] is a tool developed by Ifatec for predicting the performance of relational database applications using simulation. This tool is currently being superceded by its re-engineered successor, SWAP. SMART/SWAP is a sophisticated and versatile tool which is able to model complex real applications running on a variety of different platforms. Currently it models the performance of Oracle7 and is in the process of being extended to model Oracle8.

BEZPlus [13] is a tool for monitoring and predicting the performance of NCR Teradata and Oracle environments on MPP machines. The monitoring of the DBMS is carried out by the 'Investigator' tool, which monitors resource utilisation to highlight potential bottlenecks. The 'Strategist' evaluates hardware and software alternatives in order to identify the effect on performance and workload of business growth.

Metron Technology [14] [15] are developing a capacity planning and performance management system for Oracle. It aims to predict the system performance parameters with a reasonable level of accuracy, but so far the published details of the tool give no indication as to how it will model mixed workloads and complex queries. At present they have an Oracle performance and resource consumption monitoring tool called VIEWDB [16], which is part of their general computer monitoring tool suite, Athene.

III. STEADY

Originally STEADY was developed to model an Ingres Cluster [4]. It has been extended to model the Informix OnLine Extended Parallel Server (XPS) [3] and the Oracle7 Parallel Server [2] with the Parallel Query Option. Two different parallel DBMS architectures (shared disc and shared nothing) are represented by these latter two DBMSs. The underlying hardware platform for both systems consists of a number of processing elements (PEs) on which instances of the DBMS are running. PEs communicate using an interconnect. The platform which STEADY currently supports is the ICL GoldRush machine [17].

STEADY is designed to predict maximum transaction throughput, resource utilisation and response time given a transaction arrival rate. Fig. 1 illustrates the overall architecture. From this it can be seen that the input to STEADY consists of four sets of information: a description of the database (relations), the data placement strategy to be used, the DBMS/platform configuration, and the execution plans of SQL queries represented as annotated query trees. Within STEADY there are a number of modules which are grouped into four parts or layers. After executing all of these, the system produces as output details of the performance in terms of maximum throughput, resource utilisation and response time.

The four layers used are as follows:

A. The Application Layer

This is concerned with the characteristics of the user's data and the operations which the queries perform on the data. It contains two basic tools , the Profiler and DPTool.

The Profiler takes the input information provided on the base relations and uses it to generate profiles on the base relations. It also generates information on intermediate relations, such as number of tuples, resulting from particular data operations performed in the queries.

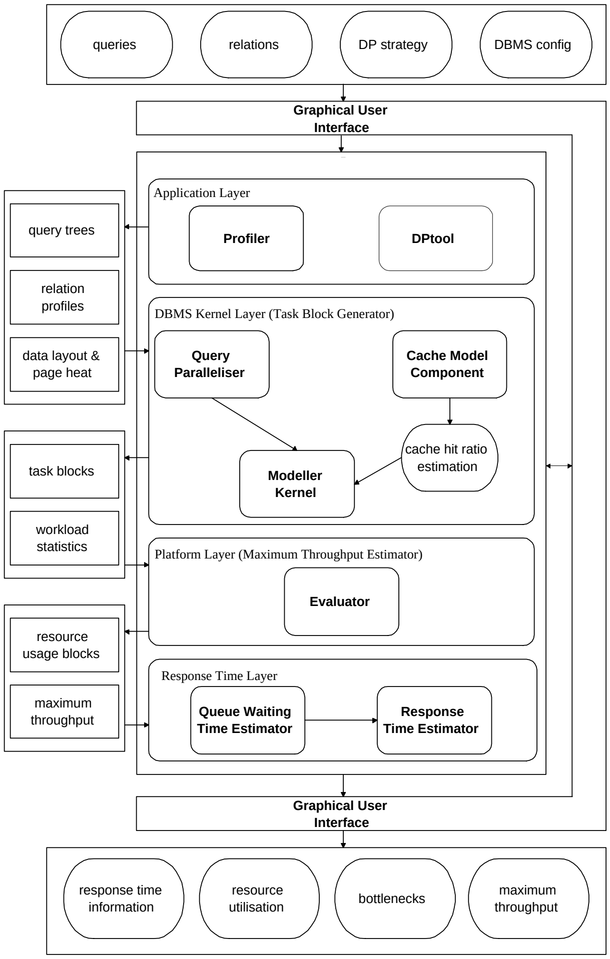


Fig. 1. STEADY architecture

The DPTool is used to determine how the data is distributed within a parallel database. The user selects the strategies to be used and given these, DPTool takes the information about the relations and the operations to be performed on them and determines how the relations should be fragmented and allocated to the different PEs, and then to individual discs attached to each PE. DPTool supports a number of data placement strategies including Hua's strategy [18] and the Bubba strategy [19] as well as simpler ones such as hash and round-robin. By experimenting with different strategies [20] the data placement tool can be used to evaluate the effect on data distribution and hence on overall performance of the system. DPTool also estimates the access frequency of different pages in each relation which is required by the Cache Model Component in the next layer.

B. The DBMS Kernel Layer

This is concerned with the particular DBMS and how it handles the user's queries on the data distribution determined by the previous layer. It consists of three modules: the Cache Model Component, the Query Paralleliser and the Modeller Kernel.

For the Cache Model Component a characterisation of the behaviour of the cache is employed which takes information on the access frequency of pages determined by DPTool and uses this to estimate the cache hit ratio for pages from different relations [21].

The Query Paralleliser estimates how queries are fragmented into basic operations and what parallelism strategies are employed. This it does by transforming the query tree into a *task block* structure, a directed graph structure in which each node is a task block. A task block represents one or more basic actions to be carried out in the process of executing the query. One may view this as a chunk of code which may be sent to one or more PEs to be executed. For example, a query which scans one relation and joins the resulting tuples with those of another relation.

Associated with each task block is information on the relationship between the task block and any predecessors in the directed graph. For example, if a task block is flagged as independent, it has no predecessors; if it is fully dependent then it is required that the previous block completes before the next one can start. The other form of dependency is pipeline dependency where a sending and a receiving block communicate tuples through a pipeline. In addition, the task block tree structure captures the inter- and intra-operator parallelism within the query.

The Modeller Kernel takes various files as input. They are: the relation profiles and query tree which are supplied by the user, data layout and page heat which are produced by the Application layer, estimated cache miss ratios and the task block profile of the query which is produced by the other modules in the DBMS Kernel layer. It fills in the details of the task blocks by expanding the execution phase within each block into corresponding sequences of basic operations. For example, an update of a tuple is an execution phase which is broken down into the following sequence of operations: wait for an exclusive page lock; obtain the lock; update the tuple, write to the logical log, write to the physical log and release the exclusive lock. The Modeller also produces a set of workload statistics in terms of the numbers of basic operations performed in the course of a transaction.

C. The Platform Layer

This layer consists of one module, the Evaluator. It contains a hardware platform model which is produced from a set of analytical formulae. It takes the workload statistics and task block profiles of the queries as input and maps them into a resource usage representation. From this representation, the bottleneck resource is highlighted and an upper limit for throughput is calculated. The details of the representation are described in Section V, where a detailed example is presented.

D. The Response Time Layer

This layer consists of two modules: the Queue Waiting Time Estimator and the Response Time Estimator. They compute the response time for each transaction. This is done by first converting the resource usage representation of the transactions into an equivalent open multi-class queueing network. Details of this process are the subject of a further paper [22].

IV. MODELLING DIFFERENT DBMSs

One difficulty with developing a tool aiming to model different DBMSs is that major parallel DBMSs, such as Ingres Cluster [4], Oracle 7 Parallel Server [2] and Informix OnLine XPS [3] adopt very different approaches to handling queries. Not only do they employ different degrees of parallelism, but they also use different caching, locking and logging mechanisms. This is further complicated by differences in the number and functionality of background processes. In this section, a brief comparison is given of some issues relating to performance modelling of these three major parallel DBMSs currently ported to the ICL GoldRush Megaserver [17] and shows how STEADY is able to model these features. It is also representative of its class of database servers: a number of SMP nodes connected via an interconnection network.

A. System Architecture

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

An Oracle Parallel Server configuration consists of a number of loosely coupled processing elements on which individual database servers are running, sharing a common database at the disc level. There is a distributed lock manager (DLM) which maintains the status of distributed locks, thereby co-ordinating access to resources required by different database servers. An Oracle instance on one PE can also access data stored in discs attached to other PEs through an underlying distributed file system. Data placement is transparent to the users. Oracle 7 Parallel Server utilises inter-query parallelism, while its parallel query option also supports intra-query parallelism including both inter- and intra-operator parallelism under some restrictions.

The Informix OnLine XPS on GoldRush utilises different forms of parallelism to divide time-consuming query trees into a number of subtrees which 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 traffic. Again, each PE runs its own instance of the database (a *co-server*) which consists of basic OnLine XPS services for managing its own logging, recovery, locking and buffer management.

B. Parallelism

GoldRush Ingres Cluster does not support parallelisation of queries. It simply distributes transactions across multiple PEs.

Oracle 7 Parallel Server has a parallel query option which breaks a query into sub-tasks and executes these in parallel. The *server* processes are co-ordinated by a *master* process which divides up the work

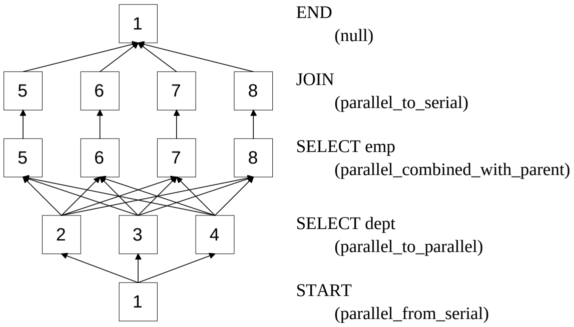


Fig. 2. Server allocation

and combines the results. When properly configured, the servers will access different sections of a table on different physical discs. If the required data is remote to a server (which it typically is), then it is transferred across the interconnect. All Oracle parallel queries must begin with a full table scan.

Informix OnLine XPS uses SQL operators to divide a query tree into subtrees that can be executed in parallel across co-servers. An *exchange* operator takes the results of two or more instances of an SQL operator and initiates another set of operators to process the next SQL operation. To do so, it takes the data provided, repartitions the data, and distributes the partitioned results along with instructions for executing the next SQL operations on multiple co-servers. The exchange operators support pipelining of the intermediate results so that the co-server does not need to wait for an operation to complete before sending the results to the next operator. All the scan operations are partitioned in such a way that an instance of a scan operator always covers the scanning of the data stored locally. Instances of join operators are always spread over all PEs.

To illustrate how the various types of parallelism are modelled in Oracle, consider the following example:

```
SELECT dept.name, dept.type, emp.name, emp.age
FROM   dept, emp
WHERE  dept.type = "Science" AND dept.dno = emp.dno
```

Fig. 2 shows the server allocation for the example query. The default number of servers for the dept table is three and for the emp table it is four. Three servers (2,3,4) scan the dept table and broadcast the tuples, where the type is "science", to the four servers (5,6,7,8) who are scanning the emp table. Matching emp and dept tuples are joined and the results are sent to the query master. The type of parallelism is shown in brackets. *Parallel_from_serial* indicates the operation is processed on a single server and the results are supplied to multiple servers. *Parallel_to_parallel* indicates the operation is processed on multiple servers and the results are passed to different multiple servers. *Parallel_combined_with_parent* indicates the operation is processed on the same servers as the next operation. *Parallel_to_serial* indicates the operation is processed by multiple servers and the results are all passed to a single server. *NULL* or *Serial* indicates the operation is processed on a single server only.

Fig. 3 shows the steps of the whole query and how the task blocks fit together. The query is initiated in the Start block. The home is selected at random, in this case PE3. The start block activates the scan_dept block, which scans the smaller of the two tables. The probability of a server being run on a PE is 0.6 as there are 5 PEs available to do the scan and 3 servers. As PE3 is the master it is not involved in the scan. At the end of the loop, tuples are sent to the join block in a pipeline. As the join is a nested one, all qualifying dept tuples are broadcast to the four servers which perform the join with the emp table.

Once the selected tuples from dept have been broadcast to all the servers, the emp table is read by four servers. The probability of a server being run on a PE is 0.8 as there are 5 PEs available to do the scan and 4 servers. First, the tuples sent by scan_dept are received in a pipeline, after which the scanning of the emp table begins. The join block also carries out the join, in the same loop which reads the emp table and by the same servers. As soon as a tuple is read it is joined with the selected dept tuples. The end block has PE3, the master server, as its home. It receives result tuples from the join block in a pipeline. Once all the results have been received, the locks for reading the pages are released.

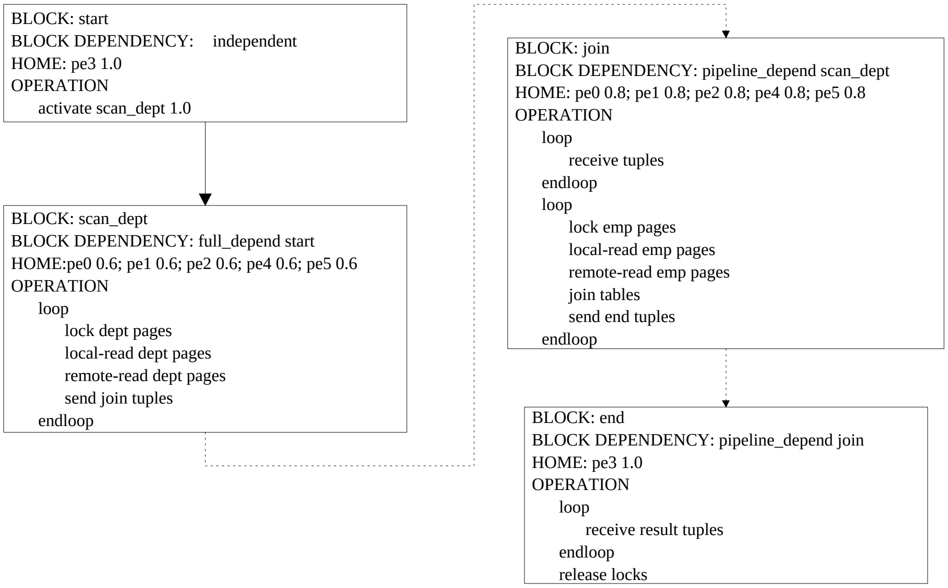


Fig. 3. Task block profile

This Oracle example can be usefully contrasted with Informix XPS by looking at how Informix handles the same query:

SELECT dept.name, dept.type, emp.name, emp.age
FROM dept, emp
WHERE dept.type = "Science" AND dept.dno = emp.dno

Intra-operator parallelism is used when the smaller relation is scanned and the tuples from it are sent to appropriate co-servers based on the value of a hash function applied to the join attribute. By default in Informix all co-servers participate in the hash join, so all of them will get tuples from the smaller relation to build a hash table in memory. Pipeline parallelism is used between these two execution phases — scanning and building a hash table — so that they are running in parallel. This continues until the smaller relation has been completely consumed. Then the other relation is scanned in a similar manner. Intra-operator parallelism is employed, and tuples are sent to appropriate PEs (to probe the generated hash tables) according to the same hash function used in the scanning of the smaller relation. There is pipeline parallelism between the scanning of the second relation and the probing phase of the execution of the join.

C. 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 locks: table and page level. All lock requests, regardless of their levels, are processed through the DLM. Locks are released when a COMMIT statement is issued, when a rollback aborts a transaction, or when an Ingres session ends. Cache coherency is achieved through the process in which one PE reads the pages modified by other transactions running on other PEs from disc through the distributed file system.

Oracle 7 Parallel Server has a more complicated locking mechanism which provides a high degree of data concurrency using parallel cache management. It guarantees cache coherency among the buffer caches of instances running on different PEs 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 that require access to the cache block. A PCM lock is acquired when an instance needs access to the data blocks the PCM lock covers and is retained until another instance requires it. It can be released and re-acquired multiple times during the execution of a transaction.

As Informix OnLine XPS uses the principle that data operations are always executed where data resides, no distributed lock management is needed. Lock management is only at the local level of a co-server. Cache management is also much like that in a conventional DBMS, as no cache coherency is required. However a transaction executes on the shared-nothing architecture in two phases: in the first phase locks are obtained on different co-servers. This phase has to complete before the next one can begin. It is a type of two-phase locking, but across multiple nodes. In the second phase, the data that was locked in the previous phase is updated across the nodes. Once this has been done the transaction can commit.

Two examples are presented that illustrate the modelling approach. In Oracle and Ingres the remote lock management and disc I/O jobs are performed on PEs which are different from the PE on which the transaction is executed. This is reflected in the following example, where a transaction running on PE1 requests a lock from the DLM instance running on PE2 and hence an extra task block is added for lock request processing by the DLM instance. v is the probability that the page that the lock request is for, is not held in the local cache. The additional *dlm* block represents the DLM handling a lock request from the transaction running on PE1. The owner of the page may be any of the four PEs; thus each has a probability of 0.25 of owning the page lock. lck_req is the message size transmitted from PE1 to PE2 to request a lock. lck_rep is the message size transmitted from PE2 to PE1 to approve a lock.

Block Name : start
Block Dependency : independent
Home : pe1 1.0
Operation
. . .
send dlm lck_req v

Block Name : dlm
Block Dependency : full_depend start
Home : pe1 0.25; pe2 0.25; pe3 0.25; pe4 0.25
Operation
obtain_lock;
send end lck_rep {pe1:1.0} 1.0

Block Name : end
Block Dependency : full_depend dlm
Home : pe1 1.0
Operation
. . .

An example of locking in Informix is given in Fig. 4, showing the execution of a transaction that updates three tables: account, branch and teller. The transaction executes under the control of a two-phase locking mechanism. Page locks are obtained as the relations are scanned and are released in the end block. The details of the individual task blocks are not given.

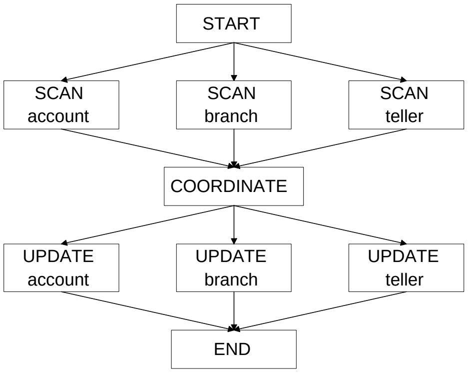


Fig. 4. Informix example

D. Disc I/O, Logging, and Background Processes

Ingres Cluster on GoldRush only supports *fast-commit* and *write-behind* in a single node configuration. In multi-node configurations, *group commit* is used to minimise the frequency of disc write operations. Checkpoints are usually performed in an exclusive time slot.

The buffer cache managed by PCM in Oracle reduces the number of reads required in a transaction and the writes conducted by the DBWR process are deferred for I/O optimisation. Apart from checkpoints, a full cache or too large a number of dirty blocks in the cache, DBWR also performs disc write operations when a dirty cache block held by one instance is requested by another instance for access. Oracle does not issue a commit confirmation until it has recorded the transaction in the redo log file and DBWR always signals the LGWR process to flush the buffer before it writes blocks back to the database.

Informix OnLine XPS uses dedicated page cleaner threads to manage the writing of dirty buffers to disc 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. The before-images of modified pages are flushed to disc before the modified pages themselves.

For both Oracle and Informix, a background process is representable as a query which uses elements of the task block syntax to perform its actions. For example, modelling the DBWR background process is carried out by estimating the number of pages in the buffer and the page size (both set by the user), and then using the cache miss probability, the number of pages in the buffer cache can be estimated. From the buffer size (set by the user), the row length of all of the tables and the query frequencies, the rate of filling of the buffer can be estimated. To model the writes that occur due to the expiry of the time interval, the cost can be added at the end of the transaction. The block representations of these background processes are similar and have the form:

Block Name : dbwr
Block Dependency : independent
Home : pe1 1.0
Operation
in_parallel{write {disc1(x:p_1),disc2(x:p_2)} 1.0}

The LGWR block is identical except that the value of x is equal to a third of the size of the log buffer instead of the number of modified buffers in the cache.

V. CHECKING THE VALIDITY OF THE ANALYTICAL APPROACH

The credibility of any complex modelling exercise is underpinned by the ability to demonstrate that the model accurately captures the way the modelled system behaves. Validation of the model by comparing its predictions with those of the measured behaviour of the real system is an obvious way to do this. However, the high cost of parallel database systems can make it difficult to obtain sufficient dedicated access to be able to use this method alone. Thus in order to check the validity of the approach, this process may be supported by less costly and more organisationally acceptable means which may be carried out in parallel with calibration. For the STEADY system, two methods of this kind have been employed, in parallel with calibration, to establish the model's accuracy — simulation and formal methods (process algebra).

A. TPC-C Example

TPC-C [23] is a transaction processing benchmark which exercises the database components necessary to perform tasks associated with transaction processing environments emphasising update-intensive database services. The workload is centred around the activity of a wholesale supplier which includes operations

Relation:	No. Tuples	Relation Size(bytes)
Orderline	2,000,000	100,000,000
History	1,800,000	90,000,000
Customer	200,000	80,000,000
Order	200,000	10,000,000
Stock	20,000	400,000
Parts	3,143	157,140
Items	1,429	142,860
New-Order	2,000	40,000
District	200	10,000
Warehouse	20	2,000

TABLE I

The TPC-C Relations

such as verifying and updating customer credit, receiving customer payment and controlling the inventory of multiple warehouses. Ten relations are involved: Warehouse, District, Stock, Items, Parts, Customer, Order, New-Order, Orderline and History. In this example, only Customer, Warehouse, District and History are used, although all ten relations are placed on the discs. It is assumed that there are twenty warehouses, ten districts and two hundred thousand customers for each warehouse. The key attribute in each relation has an index placed on it. Table I summarises the relations employed.

The platform modelled is an ICL GoldRush machine running Ingres. The number of PEs is varied from 1 to 13 with one disc attached to each PE. Each relation is hashed into twenty fragments on the warehouse id and then placed on PEs by a size placement strategy [20]. Since the number of fragments remains constant while the number of PEs changes, the number of fragments placed on each PE is different for each test.

Both validation methods use the Payment transaction, shown in Fig. 5 (a). The transaction takes as input a given warehouse, customer and district, as well as an amount, a date and extra information. Firstly, the transaction returns the customer's balance. Next, the customer's balance is updated with the given amount. Similarly, the district's total and the warehouse's total are updated. Finally, the details of the transaction are inserted into the History table.

B. Validation by Simulation

Part of the process of checking the validity of the analytical approach used in STEADY is to determine the level of accuracy of results that can be achieved using the approximation algorithm described in the previous section. One way of achieving this is by finding solutions for the model using discrete event simulation and then comparing these with the figures that are obtained using STEADY. The simulation is based on the resource usage profiles of transactions, generated by STEADY's platform layer. Transactions are implemented as simulation processes which access shared resources according to the specifications in

```
Inputs
                warehouse_id, customer_id, district_id,
                trans_amount, date_time, info
End_Inputs

SELECT          c_balance
FROM            customer
WHERE           c_w_id = warehouse_id AND
                c_d_id = district_id AND
                c_id = customer_id

UPDATE          customer
SET             c_balance = c_balance + trans_amount
WHERE           c_w_id = warehouse_id AND
                c_d_id = district_id AND
                c_id = customer_id

UPDATE          district
SET             d_ytd = d_ytd + trans_amount
WHERE           d_w_id = warehouse_id AND
                d_id = district_id

UPDATE          warehouse
SET             w_ytd = w_ytd + trans_amount
WHERE           w_id = warehouse_id

INSERT INTO             history
VALUES          ( customer_id, district_id,
                warehouse_id, date_time,
                trans_amount, info)
```

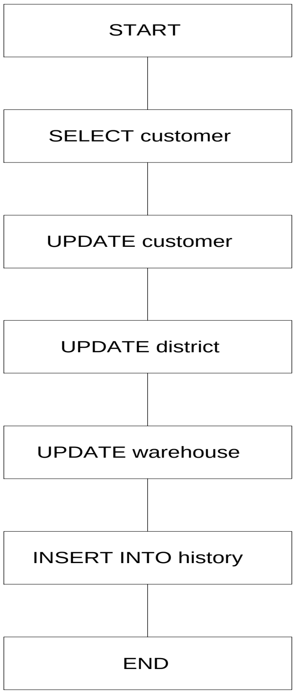


Fig. 5. Payment Transaction: (a) SQL. (b) Task block profile.

these profiles. New transactions are generated according to the overall arrival rate distribution. Simulation output includes transaction throughput, response time and statistics on resources (including utilisation).

To produce the resource usage profiles of the example transaction, STEADY first transforms it into a set of task blocks. The example transaction's task block profile is shown in Fig. 5 (b). This profile is produced by the Query Paralleliser which creates a number of task blocks for the transaction, linking them with appropriate dependency relations. Each block in Fig. 5 (b) corresponds to a phase of execution of the original relational operators within the transaction. Having been analysed by the Modeller Kernel and the Evaluator the blocks are then given a resource usage representation.

Fig. 6 (a) shows the task block and Fig. 6 (b) the resource usage representation for the update customer part of the transaction, where the number of PEs is eight. The block is fully dependent on the completion of the select customer block. The home of this block can be any one of the eight PEs. Following each PE is the probability of it being the home, which depends on the placement of the customer table. Since in this example PEs 0-3 each have two fragments of the customer relation while the remaining PEs each have three, the home probability of the PEs is 0.1 and 0.15 respectively. The resource block's first group, not present in the task block, is the cost for receiving the activation message from the select customer block. The rest of the two blocks follow the same pattern. The shared lock for reading the data page is waited for and then obtained. The page is read from one of the PEs. The numbers at the end of the read are the cache miss probabilities of each PE. Next, the page, now in the cache buffer is checked for the

```
BLOCK: update_customer
BLOCK DEPENDENCY: full_depend select_customer
HOME:   pe0 0.1; pe1 0.1; pe2 0.1; pe3 0.1; pe4 0.15;
        pe5 0.15; pe6 0.15; pe7 0.15
OPERATION_DEFINITION






mean_shared_lock_waiting_time;

obtain_lock;

read{disc0(1.0)}    pe0:0.99&pe1:0.99&pe2:0.99&pe3:0.99&
                    pe4:0.99&pe5:0.99&pe6:0.99&pe7:0.99;





cpu 874;

group {
    mean_exclusive_lock_waiting_time;

    conv_up;

    cpu 1;
    cpu 1;
    cpu 1
};

write disc0(1.0);



activate insert_blk






END_DEFINITION
```

```
BLOCK: update_customer
BLOCK DEPENDENCY: full_depend select_customer
HOME:   pe0 0.1; pe1 0.1; pe2 0.1; pe3 0.1; pe4 0.15;
        pe5 0.15; pe6 0.15; pe7 0.15
RESOURCE_TIME
group{
    ssu 80;
    pu 56
}   pe0:0.75;pe1:0.75;pe2:0.75;pe3:0.75;pe4:1.0;pe5:1.0;
    pe6:1.0;pe7:1.0;

mean_shared_lock_waiting_time;

pu 40.0;

group {
    pu 2551;
    ssu 40.0;
    disc0 15.5
}   pe0:0.99&pe1:0.99&pe2:0.99&pe3:0.99&
    pe4:0.99&pe5:0.99&pe6:0.99&pe7:0.99;

pu 874;

group{
    mean_exclusive_lock_waiting_time;

    pu 20.0;

    pu 1.0;
    pu 1.0;
    pu 1.0;
};

pu 2551;
ssu 40.0;
disc0 15.5;

group{
    pu 32;
    ssu 60;
    net 146
}   pe0:1.0;pe1:0.0;pe2:1.0;pe3:1.0;pe4:1.0;
    pe5:1.0;pe6:1.0;pe7:1.0

END_TIME
```

Fig. 6. Update Customer: (a) Task block. (b) Resource usage representation.

correct values. Once the correct row has been identified, an exclusive lock is waited for and placed on the page containing the row. The logical and physical logs are then updated and the row is updated. The updated page is then written to disc. Finally, an activation message is sent to the next block. All locks are released in the final block when the transaction is committed.

A typical set of results obtained from the simulation, for the cases where there are eight and ten PEs respectively, is given in Table II. When used in a parallel database system, the typical maximum utilisation of a GoldRush disc was taken to be 45%. Using this, simulation figures were obtained for utilisations up to 45% of the disc (bottleneck resource) and these are compared with the maximum throughput predicted by STEADY. The results show good agreement.

Simulation studies in this manner verify the accuracy of the analytical model over a range of different queries, workloads, database and platform configurations. Any significant discrepancies would have invited

	8 PEs		10 PEs	
	Bottleneck Resource Utilisation(%)	Throughput (tps)	Bottleneck Resource Utilisation (%)	Throughput (tps)
Simulation	10	4.031973	10	7.662665
	20	8.009048	20	15.34658
	30	12.03732	30	22.44061
	40	15.73204	40	29.98539
	45	17.81342	45	33.56250
STEADY Max Throughput(tps)	45	18.1433	45	34.3249

TABLE II

System performance of Simulation and STEADY: throughput

further investigation to determine whether inaccuracies in the model or limitations of the modelling process were at stake.

C. Process Algebra

Process algebras are mathematical theories which model communication and concurrent systems. The process algebra which has been selected for use in validating the analytical approach is the Performance Evaluation Process Algebra (PEPA) [24]. It is a stochastic method which has been developed to investigate the impact of the compositional features of process algebras upon performance modelling.

In PEPA, a system is expressed as an interaction of components which engage in activities. The components correspond to parts of the system or events in the behaviour of the system. Each component has a behaviour which is defined by the activities in which it engages. Every activity has an action type and an associated duration (which is represented by a parameter known as the activity rate), and is written as (α, r) where α is an action type and r the activity rate.

PEPA has a small but powerful set of combinators that are used for modelling. The sequential composition, (α, r).P, is the basic mechanism by which the behaviour of a component is constructed. This specification means that the component will perform activity (α, r) and behaves as P on completion.

The selection or choice composition, P + Q, represents a system which may behave either as P or Q but not P and Q at the same time. Both components are enabled. The co-operation or parallel composition, P <L> Q, denotes the fact that P and Q can proceed independently and concurrently with any activity whose action type is not contained in L. However, for any activity whose action type is included in L, P and Q must synchronise to achieve the activity. In this case one component may be blocked waiting for the other one. Finally, the constant A is a component whose meaning is given by the defining equation (A=P) which gives the component A the behaviour of the component P.

PEPA models suffer from state-space explosion [24] when the models become complicated. For this reason, a decompositional approach that combines the compositional strong equivalence [24] and flow-equivalent aggregation method [25] has been adapted in the study in order to overcome this problem.

As an example consider a simple database system that consists of a single PE with a disc attached to it. The PE comprises a transaction manager (TM), a concurrency control unit (CCU), a buffer manager(BM) and a distributed lock manager(DLM). When a transaction request arrives at a PE, the TM passes the request to the CCU which will send a lock request to the DLM. The DLM may grant the lock or force the lock request into a queue and reply accordingly. Once the lock is granted, the CCU sends a message to the BM which will read from the disc and return the desired data. The code for the specification of the PE follows:

```
#Q0 = (tm2ccu, infty).(ccu2dlm, r_sgn0).Q0;
#Q1 = (dlm2ccu, infty).(begin_trans, r_sgn).Q1;
#Q = Q0 <> Q1;
#P = (begin_trans, infty).(ccu2bm, r_sgn0).P;
#R = (bm2ccu, infty). (ccu2dlm0, r_sgn0).(ccu2tm, r_commit). R;
#CCU = Q <begin_trans> P <> R;
#DLM0 = (ccu2dlm, infty).(dlm2ccu, r_gnt).DLM0;
#DLM1= (ccu2dlm0, infty).(release, r_sgn).DLM1;
#DLM = DLM0 <> DLM1;
#BM = (ccu2bm, infty).(deliver, r_deliver).(bm2ccu, r_sgn0).BM;
#TM0 = (request, r_req).(tm2ccu, r_sgn0).TM0;
#TM1 = (ccu2tm, infty).(reply, r_reply).TM1;
#TM = TM0 <> TM1;
#MODEL = (TM<>BM<>DLM<>)
<tm2ccu, ccu2tm, bm2ccu, ccu2bm, ccu2dlm, ccu2dlm0, dlm2ccu> CCU;
MODEL
```

Table III summarises the results obtained from PEPA and STEADY. Both PEPA and STEADY have been used to model a system that supports the customer payment transaction of TPC-C as previously described. Although both PEPA and STEADY are capable of modelling larger systems, it is felt sufficient for comparison purposes to look at a few significant points where the system performance varies.

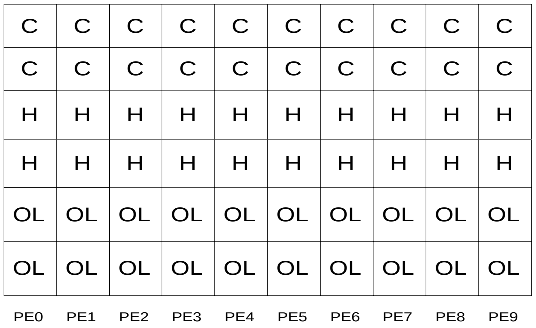
For the tests carried out, the size placement strategy orders the fragments of all relations into decreasing order of fragment size, and places each fragment in turn onto the PE that has the most space available, in an attempt to achieve a balance of data across all PEs. Since for each relation the 20 fragments are of equal size, a perfect balance will be achieved for 2, 4, 5, 10 or 20 PEs (i.e., divisors of 20). Fig. 7 shows the balanced placement of the fragments of the Orderline (OL), History (H) and Customer (C) relations for the 10 PE case. The fragment sizes are drawn proportionally to their sizes in bytes (Table I), i.e. OL = 5 Kbytes, H = 4.5 Kbytes and C = 4 Kbytes.

If the number of PEs is not a divisor of 20, the placement of the fragments of the largest relation (Orderline) introduces an imbalance, i.e., at least 1 PE will have 1 less fragment than the rest. This imbalance is subsequently compensated for by the placement of the fragments of the other relations.

	Max Throughput Predicted By	
No. of PEs	PEPA	STEADY
5	17.1505	17.1625
6	15.3174	15.3427
8	17.8823	18.1433
10	34.3010	34.3249
11	21.8771	22.0392
12	19.0884	18.7383

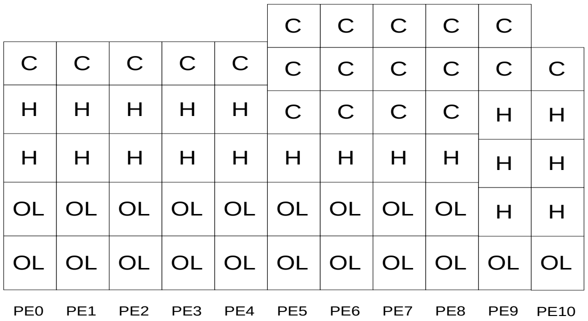
TABLE III

Maximum throughput (in tps) for the customer payroll transaction of TPCC as predicted by PEPA and STEADY for varying numbers of PEs



PE0	PE1	PE2	PE3	PE4	PE5	PE6	PE7	PE8	PE9
C	C	C	C	C	C	C	C	C	C
C	C	C	C	C	C	C	C	C	C
H	H	H	H	H	H	H	H	H	H
H	H	H	H	H	H	H	H	H	H
OL	OL	OL	OL	OL	OL	OL	OL	OL	OL
OL	OL	OL	OL	OL	OL	OL	OL	OL	OL

Fig. 7. Size placement for the customer payment transaction of TPC-C (10 PEs)



PE0	PE1	PE2	PE3	PE4	PE5	PE6	PE7	PE8	PE9	PE10
					C	C	C	C	C	
C	C	C	C	C	C	C	C	C	C	C
H	H	H	H	H	C	C	C	C	H	H
H	H	H	H	H	H	H	H	H	H	H
OL	OL	OL	OL	OL	OL	OL	OL	OL	H	H
OL	OL	OL	OL	OL	OL	OL	OL	OL	OL	OL

Fig. 8. Size placement for the customer payment transaction of TPC-C (11 PEs)

Fig. 8 shows the placement of fragments of the Orderline (OL), History (H) and Customer (C) relations for the 11 PE case. It can be seen that there are 3 fragments of the Customer relation on PEs 5 to 8 inclusive. As this relation is accessed in the example query, these 4 PEs become the bottlenecks and cause a reduction in throughput for the transaction. Similarly, placement imbalance accounts for the reduction in throughput when moving from a 5 to a 6 PE machine. The results estimated by PEPA are in good agreement with those of STEADY.

VI. Conclusions

STEADY is a performance prediction tool for parallel relational databases that predicts the throughput and response time for both complex queries and mixed workloads. In developing STEADY the aim has been to produce a generic tool which is capable of modelling different parallel databases running on different platforms. Thus far, the work has focused on three commercial DBMSs (Ingres, Informix and Oracle) running on the ICL GoldRush Megaserver [17]. The latter is representative of a class of database servers comprising a cluster of SMP nodes connected via an interconnection network.

In the future it is planned to extend the model to incorporate additional platforms. Originally the N-cube machine was considered but currently the ICL Xtra-server is being studied. An enhanced graphical user interface has been designed and implemented in Java. Other future work may include the modification of the tool to cater for future versions of the parallel DBMSs.

This paper describes some facets of the different DBMSs which the tool has had to deal with. It concludes with a brief description of two different approaches which have been used together with calibration to confirm the validity of the analytical approach used.

VII. Acknowledgements

The authors acknowledge the support received 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). They also wish to thank Arthur Fitzjohn and Monique Mitchell of ICL(Manchester) and Shaoyu Zhou of Microsoft Corp for their contribution to this work.

References

[1] M. G. Norman and P. Thanisch. "Parallel Database Technology" (1995) *Bloor Research Group, Netherlands, pages 1-546*

[2] Oracle Corp. "Oracle7 Parallel Server Concepts & Administration, Release 7.3", (1996) *Oracle, Server Products, California, USA*

[3] Informix Software Inc. "Informix-OnLine Dynamic Server & Administrator's Guide" (1994) *Informix Press, California, USA*

[4] The ASK Group Ltd. "ASK Open (INGRES) database administrator's guide" (1994) *The ASK Group Limited, Bury St. Edmunds, UK*

[5] IBM Corp. "DB2" (1998) *http://www.software.ibm.com/data/db2/*

[6] Sybase Inc. "Sybase" (1998) *http://www.sybase.com*

[7] Digital Equipment Corp. (1998) "EcoTools" *http://www.digital.com/alphaserver/products/internet/search/ecotools_3_3.html*

[8] BMC Software Inc. "PATROL DB-log Master" (1998) *http://www.bmc.com/products/trx/index.html*

[9] DB LAB PTY Ltd. "Ingres DB_Maximiser" (1996) *http://ingres.com.my/db_maximizer.html*

[10] R. Eberhard, IBM Corp. "DB2 Estimator for Windows" (1998) *http://www.software.ibm.com/data/db2/os390/db2est-home.html*

[11] Dell Computer Corp. "Oracle System Sizer" (1998) *http://www.dell.com/products/poweredge/partners/db/oracle/wp-sizer.htm*

[12] Ifatec S.A. "SMART2 User Guide, Release 2.1" (1994) *Ifatec S.A., 3 Rue Petigny, 78000 Versailles, France*

[13] BEZ Systems Inc. "BEZPlus for NCR Teradata and Oracle environments on MPP machines" (1997) *http://www.bez.com/software.htm*

[14] M. Garth. "Capacity Planning for Parallel IT Systems" (November 1996) *Computer Bulletin, pp. 16-18*

[15] T. Foxon, M. Garth and P. Harrison. "Capacity Planning in Client-Server Systems" (1996) *Journal of Distributed Systems Engineering, Volume 3, pp. 32-38*

[16] Metron Technology Ltd. "VIEWDB" (19986) *http://www.metron.co.uk/viewdb.htm*

[17] P. Watson and G. Catlow. "The architecture of the ICL GoldRush MegaSERVER" (July 1995) *Proceedings of the 13th British National Conference on Databases (BNCOD 13), Manchester, UK, pages 250-262*

[18] K. Hua, C. Lee, and H. Young. "An efficient load balancing strategy for shared-nothing database systems" (September 1992) *Proceedings of DEXA'92 conference, Valencia, Spain, pages 469-474*

[19] G. Copeland, W. Alexander, E. Boughter and T. Keller. "Data placement in Bubba" (1988) *Proceedings of ACM SIGMOD Conference*

[20] M. Howard Williams and Shaoyu Zhou. "Data Placement in Parallel Database Systems" (1997) *Parallel Database Techniques, IEEE Computer Society Press, in print*

[21] S. Zhou, N. Tomov, M.H. Williams, A. Burger, and H. Taylor. "Cache Modelling in a Performance Evaluator of Parallel Database Systems" (January 1997) *Proceedings of the Fifth International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, IEEE Computer Society Press, pages 46-50*

[22] N. Tomov, E. Dempster, M.H. Williams, P.J.B. King and A. Burger. "Approximate Estimation of Transaction Response Time" (1998) *Submitted for publication*

[23] Transaction Processing Performance Council (TPC). "TPC Benchmark C" (1992) *ITOM International Co, California, USA*

[24] J. Hillston. "A Compositional Approach for Performance Modelling", (1994) *PhD Thesis, University of Edinburgh, UK*

[25] K. Kant. (1992) "Introduction to Computer System Performance Evaluation" *McGraw-Hill Inc., Singapore*