Verifying a Performance Estimator for Parallel DBMSs

E W Dempster[1], N T Tomov[1], J Lü[1], C S Pua[1], M H Williams[1], A Burger[1], H Taylor[1] and P Broughton[2]

[1] Department of Computing and Electrical Engineering, Heriot-Watt University, Riccarton, Edinburgh, Scotland, EH14 4AS, UK
[2] International Computers Limited, High Performance Technology, Wenlock Way, West Gorton, Manchester, England, M12 5DR, UK

Abstract. Although database systems are a natural application for parallel machines, their uptake has been slower than anticipated. This problem can be alleviated to some extent by the development of tools to predict the performance of parallel database systems and provide the user with simple graphic visualisations of particular scenarios. However, in view of the complexities of these systems, verification of such tools can be very difficult. This paper describes how both process algebra and simulation are being used to verify the STEADY parallel DBMS performance estimator.

1 Introduction

Database systems are an ideal application area for parallel computers. The inherent parallelism in database applications can be exploited by running them on suitable parallel platforms to enhance their performance - a fact which has attracted significant commercial interest. A number of general purpose parallel machines are currently available that support different parallel database systems, including adaptations of standard commercial DBMSs produced by vendors such as Oracle, Informix and Ingres. For a platform based on non-shared memory, such systems distribute their data (and associated query processing) amongst the available processing elements in such a way as to balance the load [1].

However, despite the potential benefits of parallel database systems, their uptake has been slower than expected. While information processing businesses have a strong interest in obtaining improved performance from current DBMSs, they have a substantial investment in existing database systems, running generally on mainframe computers. Such users need to be convinced that the benefits outweigh the costs of migrating to a parallel environment. They need assistance to assess what parallel database platform configuration is required and what improvements in performance can be achieved at what cost. They need tools to tune the performance of their applications on a parallel database platform to ensure that they take best advantage of such a platform. They need help in determining how the system should be upgraded as the load on it increases, and how it should be reorganised to meet anticipated changes in usage of database capacities.

Application sizing is the process of determining the database and machine configuration (and hence the cost) required to meet the performance requirements of a particular application. *Capacity planning*, on the other hand, is concerned with determining the impact on performance of changes to the parallel database system or its workload. *Data placement* is the process of determining how to lay out data on a distributed memory database architecture to yield good (optimal) performance for the application.

All three activities require suitable tools to predict the performance of parallel database systems for the particular application. They determine whether a given hardware/software configuration will meet a user's performance requirements, how performance will change as the load changes, and how the user's data should be distributed to achieve good performance [2]. Such tools would rely on a model of transaction throughput, response time and resource utilisation for given system configurations, workloads and data distributions. To complicate matters further, since a parallel machine may host several different DBMSs, such tools should be capable of supporting models of different parallel DBMSs and of providing facilities for describing different features of applications running on these DBMSs.

This paper presents the results from an initial verification exercise of a parallel DBMS performance estimation tool called STEADY. The predictions produced by components of STEADY are compared with results from a simulation and with those derived from a process algebra model. The rest of the paper is organised as follows. Section 2 briefly describes STEADY. Sections 3 and 4 provide a comparison between predictions of components of STEADY and those obtained from a simulation and a process algebra model. Section 5 provides a summary and conclusion.

2 STEADY

STEADY is designed to predict maximum transaction throughput, resource utilisation and response time given a transaction arrival rate. The maximum throughput value is derived by analysing the workload and identifying the system bottlenecks. Given a transaction arrival rate, lower than the maximum throughput, the response time is derived using an analytical queuing model. During the process of calculating the response time, the resource utilisation can also be obtained.

The basic STEADY system originally worked with an Ingres Cluster model, and has been extended to model the Informix OnLine Extended Parallel Server (XPS) and the Oracle7 Parallel Server with the Parallel Query Option. These two systems are representative of two different parallel DBMS architectures: shared disk and shared nothing. 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 a fast interconnecting network. The platform which STEADY currently supports is the ICL GoldRush machine [3].

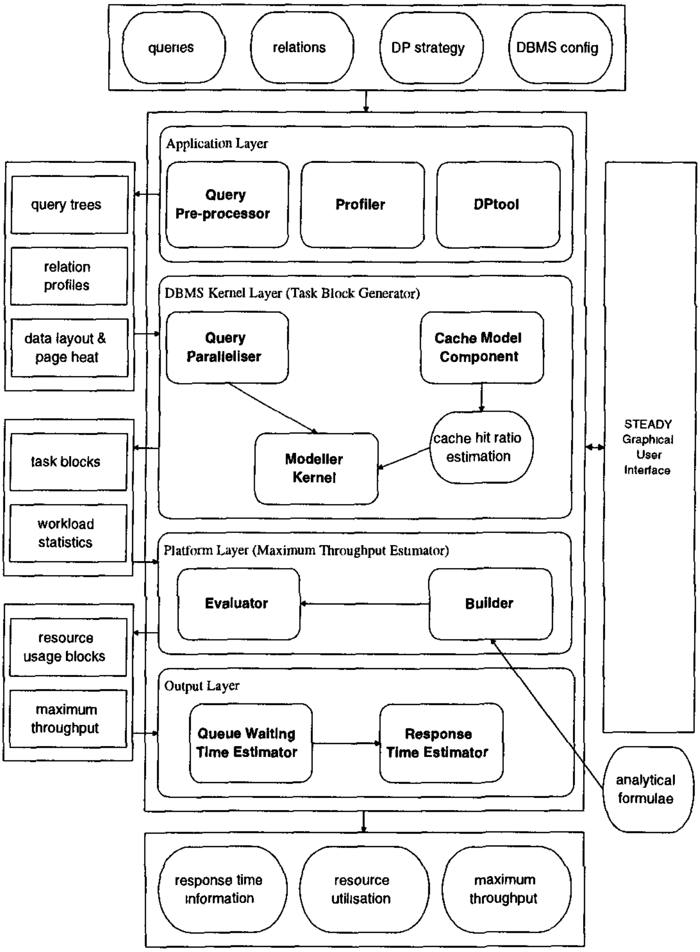

Fig. 1. STEADY Architecture

In a shared disk DBMS (Oracle 7), each of the PEs of the parallel machine may access and modify data residing on the disks of any of the PEs of the configuration. In a shared nothing DBMS (Informix XPS), the disks of a PE are not directly accessible to other PEs.

STEADY takes as input the execution plans of SQL queries, represented as annotated query trees. The 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. This process of transformation can be followed in Fig. 1 which illustrates the architecture of STEADY.

In addition to the graphical user interface the system comprises four parts. The application layer consists of the Profiler, DPTool and the Query Pre-Processor The Profiler is a statistical tool, primarily responsible for generating base relation profiles and estimating the number of tuples resulting from data operations. DPTool is used to generate data placement schemes using different strategies, and estimates the access frequency (heat) of different pages in each relation. DPTool provides the necessary heat information along with the generated data layout to the Cache Model Component. Both the Profiler and DPTool make use of the annotated query tree format of the original SQL queries.

The DBMS kernel layer consists of the Cache Model Component, the Query Paralleliser and the Modeller Kernel. The Cache Model Component estimates the cache hit ratio for pages from different relations [4]. The Query Paralleliser transforms the query tree into a *task block* structure. Each task block represents one or more phases in the execution of the relational operators within the query trees. Examples of phases are the following: building a hash table in the first phase of a hash join operator; merging two sorted streams of tuples in the last phase of a merge-sort join operator; performing a full table scan of a relation. The task blocks are organised as the nodes of a tree with dependencies among blocks represented by links. The dependencies specify ordering constraints among execution phases. One form of dependency is a pipeline dependency where a sending and a receiving block communicate tuples through a pipeline. Another form is full dependency which requires the previous block to complete before the next one can start. In addition, the task block tree structure captures the inter- and intra-operator parallelism within the query. The Query Paralleliser is able to construct the tree based on knowledge of the parallelisation techniques and the load balancing mechanism employed by the parallel DBMS.

The Modeller Kernel takes as input the relation profiles, data layout, estimated cache hit ratios and the task block profile of the query, produced by the Query Paralleliser. It produces workload profiles in terms of the numbers of basic operations which are to be executed on each PE in the course of a transaction. This results in a set of workload statistics. Together with this, the Modeller Kernel fills in the details of the task blocks by expanding each execution phase within the block into a corresponding sequence of basic operations. For example, a full table scan of a relation is an execution phase process which is represented by the following sequence of operations: wait for a page lock; obtain the page lock; read the page; select tuples from the page; send selected tuples to the next phase. This sequence will be executed once for each page of the relation. The obtained locks are released within the body of a different task block which represents the commit phase of the transaction the query belongs to.

The platform layer consists of the Evaluator and the Builder. The task block profiles of the queries are mapped by the Evaluator into sequences of resource usages. The Evaluator contains a hardware platform model which is generated by the Builder from a set of analytical formulae. This enables the Evaluator to map each basic operation into the appropriate sequence of resource usages. For example, a basic page read operation may be mapped to the following sequence, which makes explicit the order of usage and time consumption associated with the read:

```
cpu 32μs;
disk3 150ms;
cpu 50μs
```

Apart from producing these resource usage profiles, the evaluator also gives an initial estimation of the maximum throughput value.

A resource usage representation is then produced which captures the pattern of resource consumption of the original query by mapping the task blocks to resource blocks. The Queue Waiting Time Estimator and the Response Time Estimator of the output layer of STEADY work with this representation to estimate individual resource utilisation and response time and, from these, overall query response time. The details of these processes is the subject of a future paper.

3 Verification by Simulation

When estimating the performance of complex systems using analytical methods, two basic approaches are available: 1) to build a high-level abstract model and find an exact solution for the model, or 2) to build a fairly detailed model and find an approximate solution for it — finding exact solutions for detailed models is in general not feasible. In the case of STEADY, the second approach has been adopted.

Part of the verification of STEADY is to determine the level of accuracy of results that can be achieved using the approximation algorithm described in the previous section. This can be achieved by finding solutions, within error bounds, 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, which are generated by STEADY's platform layer. Transactions are implemented as simulation processes which access shared resources according to the specifications in these profiles. New transactions are generated according to the overall arrival rate distribution and the respective transaction frequencies. Simulation output includes statistics on resources (including utilisation) and transaction response times.

A number of examples have been selected and experiments conducted to compare the results obtained from simulation with those predicted by STEADY. One example is given in Figure 2 which shows the average utilisation of resources as predicted by simulation and by STEADY. By contrast Figure 3 provides a comparison of average transaction response times. In all the experiments results for average utilisation show very good agreement whereas those for response time are more variable. This is being investigated further.

4 Verification by Process Algebra

The second way in which the STEADY approach is being "verified" is through process algebras. A process algebra is a mathematical theory which models communication and concurrent systems. In the early versions of process algebras,

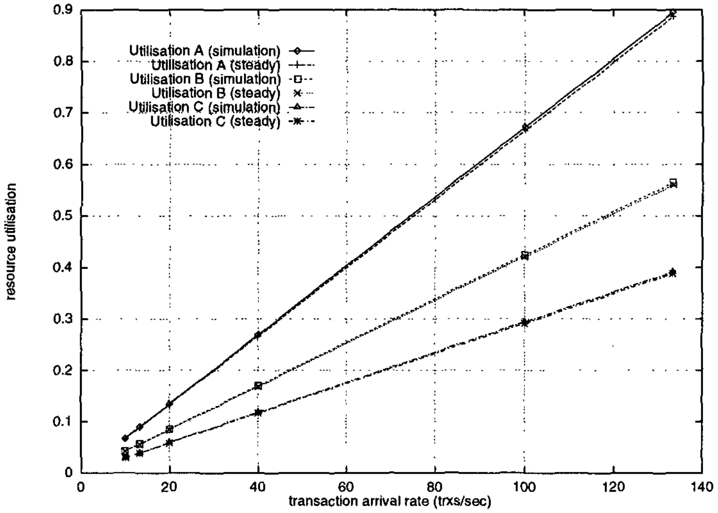

Fig. 2. Utilisation of resources A, B and C

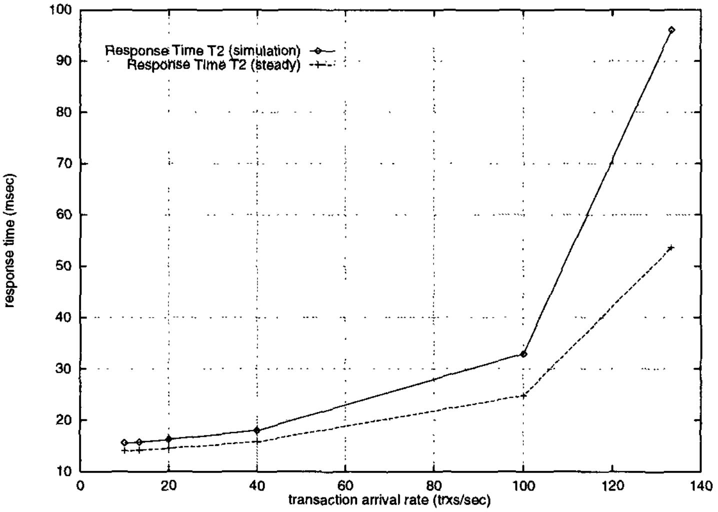

Fig. 3. Mean response time of t_2

time was abstracted away. A classical example of such an algebra is CCS (Calculus of Communicating Systems) [7].

The form of process algebra which has been selected for use here is known as a Performance Evaluation Process Algebra (PEPA) [6]. This is a stochastic process algebra which has been developed to investigate the impact of the computational features of process algebras upon performance modelling.

In PEPA, a system is expressed as an interaction of components, each of which engages in some specific activity. These components correspond to the parts of the system or the events in the behaviour of the system. The behaviour of each component is defined by the activities in which it can engage. 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 activity and r the activity rate.

To illustrate the PEPA approach, consider the simplest of models, comprising a single processing element with one processing unit and one disk (see fig 4). Suppose that transactions arrive at the transaction manager (TM) module, which on receipt of a transaction passes it to the concurrency control unit (CCU). The latter sends a message to the distributed lock manager (DLM) requesting a lock; the DLM grants the lock and replies accordingly. The CCU sends a message to the buffer manager (BM) which reads from disk and returns the data. Finally the CCU releases the lock and commits the transaction.

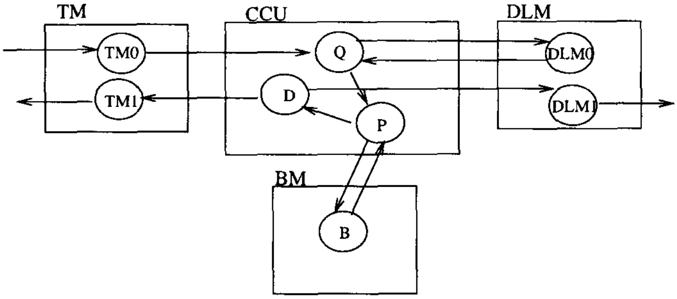

Fig. 4. Single Processing Element Configuration

This is expressed in PEPA notation as follows:

```
#Q0 = (tm2ccu,infty).(ccu2dlm,r_sgn0).Q0 ;
#Q1 = (dlm2ccu,infty).(lock_granted,r_sgn).Q1;
#Q = Q0 ⟨⟩ Q1;
#D = (release_lock,infty).(commit,infty).(ccu2dlm0,r_sgn0). (ccu2tm,r_sgn0).D;
#P = (lock_granted,infty).P1;
#P1 = (ccu2bm,r_sgn0).(bm2ccu,infty).(release_lock,r_sgn). (commit,r_commit).P;
#CCU = (Q ⟨⟩ D) ⟨ commit, release_lock, lock_granted ⟩ P ;
#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;
#PEPA = (TM ⟨⟩ BM ⟨⟩ DLM)
⟨ tm2ccu,ccu2tm,bm2ccu,ccu2bm,ccu2dlm,ccu2dlm0,dlm2ccu ⟩ CCU;
PEPA
```

Here the notation $(\alpha, r).REST$ is the basic mechanism by which the behaviour of a component is expressed. It denotes the fact that the component will carry out activity $(\alpha,\ r)$ which has action type α and a duration of mean $1/r$, followed by the rest of the behaviour (REST). The notation $P\langle L\rangle Q$ where P and Q are components and L is a set of action types, denotes the fact that P and Q 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, components P and Q must synchronise to achieve the activity. These shared activities will only be enabled in $P\langle L\rangle Q$ when they are enabled in both P and Q , i.e. one component may be blocked waiting for the other component to be ready to participate.

For this very simple configuration, a very simple benchmark has been adopted in which a single relation with tuple size of 100 bytes is scanned by a query performing a simple search. The page size is set to 1024 bytes and each page is assumed to be 70% full. Each fragment of the relation is assumed to consist of a single page. The average time to perform a disk read is taken as 41.11 milliseconds.

Results for this very simple example are given in Table 1.

Query arrival rate	Throughput (tps)			
	data in 1 frag	data in 2 frags	data in 3 frags	data in 4 frags
5.0	4.996	4.962	4.816	4.512
10.0	9.919	9.015	7.339	5.858
25.0	20.418	11.924	8.058	6.059
100.0	24.019	12.089	8.077	6.063
STEADY Max throughput(tps)	24.324	12.162	8.108	6.081

Table 1. System performance of PEPA model and STEADY: throughput (in tps)

In order to solve such a model, the method reduces the model to a set of states. In the case of this very simple example, the number of states generated is 1028. However, when one considers slightly more complex models, such as configurations involving more than one processing element, the number of states increases rapidly (approx 810000 states for two processing elements) and the computation required to solve it becomes too large.

In order to handle such cases we are currently experimenting with the flow equivalent aggregation method [8] to decompose the models into manageable sub-models and find solutions to these. The results so far are encouraging although no complete solution is yet available.

5 Conclusions

This paper has discussed the verification of an analytical model (STEADY) for estimating the performance of parallel relational DBMSs. Two different methods of verification were investigated. The first approach was to apply simulation to a set of problems. The same problems were also solved using STEADY. The results of the comparison between STEADY and the simulation were mixed, as some compared very favourably whereas others require further investigation. Although the simulation alone is not sufficient to verify all aspects of STEADY, it has proved to be valuable in analysing, choosing and developing approximation algorithms for mathematical solutions of detailed analytical models.

The second, more theoretical, method of verification that was investigated was a process algebra system known as Performance Evaluation Process Algebra (PEPA). This system was used to verify the components of STEADY which produce both the resource usage profiles and the first estimation of maximum system throughput. The system used to verify STEADY was a simple system as PEPA becomes unmanageable when more complex configurations are considered. Further work on PEPA is being carried out to handle more complex configurations, whilst keeping the system manageable.

The processes described are intended as an initial stage in the verification of STEADY, being performed in parallel to validation against actual measured performance.

6 Acknowledgments

The authors acknowledge the support received from the Engineering and Physical Sciences Research Council under the PSTPA programme (GR/K40345) and from the Commission of the European Union under the Framework IV programme, for the Mercury project (ESPRIT IV 20089). They also wish to thank Arthur Fitzjohn of ICL and Shaoyu Zhou of Microsoft Corp. for their contribution to this work.

References

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

2. M. B. Ibiza-Espiga and M. H. Williams. “Data placement strategy for a parallel database system” *Proceedings of Database and Expert Systems Applications 92, Spain, 1992. Springer-Verlag, pages 48-54*
3. P. Watson and G. Catlow. “The architecture of the ICL GoldRush MegaSERVER” *Proceedings of the 13th British National Conference on Databases (BNCOD 13), Manchester, U.K., July 1995, pages 250-262*
4. S. Zhou, N. Tomov, M.H. Williams, A. Burger, and H. Taylor. ”Cache Modelling in a Performance Evaluator of Parallel Database Systems” *Proceedings of the Fifth International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, IEEE Computer Society Press, January 1997, pages 46-50*
5. L. Kleinrock. ”Queueing Systems, Volume 1: Theory” *John Wiley & Sons, Inc., Canada, 1975*
6. J. Hilston. “A Compositional Approach for Performance Modelling”, *PhD Thesis, University of Edinburgh, 1994.*
7. R. Milner. “Communication and Concurrency” *Prentice Hall International, UK, 1989*
8. K. Kant. “Introduction to Computer System Performance Evaluation” *McGraw-Hill Inc., Singapore, 1992*