An Evaluation of Precompilation and Interpretation in Distributed Database Management Systems

ELISA BERTINO
Istituto di Elaborazione della Informazione, Consiglio Nazionale delle Ricerche, Via S. Maria 46-56100 Pisa, Italy

In this paper we present a simple model for the performance evaluation of Distributed Database Management Systems (DDBMS). We first define a transaction-processing model. Then we provide a set of expressions to evaluate the impact of various factors on the performance. The performance is evaluated in terms of transaction response time. Finally we compare the pre-compilation and the interpretation approaches using these expressions.

Received March 1986

1. INTRODUCTION

The growing number of mature database applications as well as the rapidly developing telecommunication technology have created a situation of increasing demand for integration of data resources residing at different nodes of a computer network as well as integration of the existing information systems. The distributed database management (DDBMS) technology is emerging as a result of the above requirements.

The reasons that have motivated the development of DDBMS are several, both organizational and technical. From an organisational point of view there was the need for: (1) major flexibility in reflecting the structure of large, geographically dispersed organisations; (2) integration of pre-existing databases; (3) local autonomy, responsibility and ownership; (4) graceful growth of systems. From a technical point of view the potential advantages offered by a DDBMS were: (1) improved performance, since it is possible to increase the locality of references and exploit parallelism during the execution of database operations; (2) reliability and availability, since it is possible to replicate data; if a node crashes, the other nodes can continue to operate normally, making available copies of data.

In particular, the need to interconnect pre-existing databases has been demonstrated in many large organisations, which gather and maintain semantically related data stored in different DBMSs. It is therefore important to provide a uniform and integrated access to the various databases, since users cannot be expected to learn and use many different DBMSs.

An important requirement of DDBMS concerns the performance. In fact, if distributed database technology is to be viable,[12] the DDBMS must be built with careful attention to reasonable performances for typical user operations. It is therefore important to provide a model to evaluate the factors that most affect the DDBMS performance.

1.1 Previous Related Work

No methodology exists that addresses the problem of evaluating the performance of a DDBS in all its aspects, such as the one defined for centralised DBS.[4, 20] Instead, various aspects have been evaluated in isolation, that is, making simplified assumptions on the overall DDBMS functioning. In particular, an area which has received much attention concerns evaluation of concurrency control mechanisms for centralised and distributed database systems.[1, 7, 10, 13, 14, 17, 20, 25, 27] Here we survey briefly some of the studies that have focused on distributed database systems.

Ries[17] compared four concurrency control algorithms, two based on centralised control and two based on distributed control. The centralised methods were variations of centralised two-phase locking; the difference in the two algorithms was essentially concerning transaction scheduling. The distributed control methods differed in the way they solved the deadlock. In one, a deadlock detection algorithm was invoked periodically, in the other the 'wound-wait' model was adopted, which prevents deadlock situations. The simulation model used involved about twenty input parameters that described the database, the transactions, the sites and the network. Performance measures included I/O utilisation and average response time. The simulation results indicated that choice of the best algorithm in terms of the overall database system performance is application-dependent. However, the results indicated that when most transactions can be handled locally the distributed control leads to better performance, while if most transactions are non-local centralised control performs better.

Lin and Nolte[14] have first evaluated the two-phase locking in a centralised DBMS. In the simulation model, the application environment is characterised by the transaction size, and the system environment is characterised by the number of transactions running concurrently and the total number of lockable units. Performance measures include the probability of a lock involved in a conflict (PC) and deadlock (PD) respectively and the average waiting delay (WT). The simulation results indicated that the system behaved quite similarly for different access distribution; PC, PD, WT all increased more than linearly with the multiprogramming level and the transaction size. Lin and Nolte have also studied, for two-phase locking in a DDBMS, how the communication delay affects the blocking delay and system performance. The results showed that the communication delay has little effect on the probability of conflict and deadlock of lock requests.

Garcia-Molina[10] compared two concurrency control algorithms, one based on centralized control and the other on distributed control. The distributed algorithm

was the majority consensus algorithm due to Thomas[26]; the other one the centralised locking algorithm. He analysed and compared these two algorithms in the case of completely duplicated databases. Garcia-Molina used a simple model for the transactions in which read-sets and write-sets are predeclared and each transaction consisted of only a read phase, a processing phase and a write phase. Performance measures include CPU and I/O utilisation, average response time of updates and the number of messages sent per update. Garcia-Molina's results indicated that the centralised locking algorithm performs considerably better than the decentralised one, except in cases of extreme I/O utilisation.

Other studies that have been reported are based on direct measurement of existing DDBMS prototypes.[5, 23] In the work reported in Ref. 5, the performance analysis is described for a simplified model of the heterogeneous distributed system ADDS. ADDS[6] is an experimental heterogeneous distributed database system designed to provide a uniform interface to various pre-existing heterogeneous databases that are distributed among the various nodes of a computer network. The simplified model of this system, consisting of two remote sites, was evaluated by conducting 4,555 tests requesting data from IMS databases and receiving data as CMS files. The results showed that transmission time is the most significant factor in system response when large data files are transmitted through the network. In most cases, the transmission time accounted for 80–90% of the system response time. The work reported in Ref. 23 describes some results obtained by means of benchmarks on various configurations of the distributed INGRES.[22] The benchmarks were run on a VAX 11/780 along with 0, 1 or 2 VAX 11/750. All configurations used the local network Ethernet. The major conclusion drawn was that the factor that most affects the performance is the parallelism within the query execution. It is therefore important that the query strategy takes into account the processing speed at the various nodes, so that the load is equally distributed among the participant sites and the maximum parallelism is achieved. The authors also concluded that in a local network environment the network transmission time was not bearing on the performance in a considerable way.

1.2 Structure of the Paper

The remainder of this paper is organised as follows. In Section 2 we describe a transaction-processing model in a DDBMS. At a more general level the transaction processing can be decomposed in three steps:[15] preparation, execution and commit. For each step the main tasks are analysed from the point of view of the performance. An important issue also discussed concerns pre-compilation and interpretation, which are two different approaches to the preparation phase.

Using the analysis done in Section 2, a set of expressions have been derived in Section 3 to provide an estimate of the DDBMS response time when executing a transaction. These expressions provide a more precise idea of the factors that affect the system performance when executing a transaction. In Section 4 we discuss the impact of pre-compilation and interpretation and we compare the two approaches. Other results are presented in Ref. 2.

2. TRANSACTION PROCESSING MODEL

In both centralised and distributed DBMSs a transaction is defined as a set of data access and manipulation statements with the atomicity property.[11] The atomicity property ensures that either all updates are executed or none.

At the more general level the transaction processing can be divided into three phases:

- preparation step: during this step each query statement contained in the transaction is processed and the strategy for the query execution is generated;
- execution step: during this step the actual query execution takes place;
- commit step: during this step a decision is reached among the participating sites as to whether or not the transaction must be committed; if the transaction is committed, the transaction updates are installed permanently in the various local databases.

An important activity performed during the execution of the previous steps is the concurrency control. The goal of the concurrency control mechanism is to synchronise concurrent accesses to data. This prevents anomalies such as lost updates or dirty reads.

2.1 Preparation Phase

2.1.1 *Pre-compilation and interpretation*

A first important distinction for the preparation phase is pre-compilation versus interpretation.

Pre-compilation means that the query statements contained within the transaction are processed before the actual execution takes place. At the end of the compilation, an access module for the transaction is generated and stored in system catalogues.[8] The access module produced by the query compilation can be optimised and generally runs much faster, because the bindings represented in the access module bypass several levels of interpretation and authorisation that must be applied to the original query. Users may compile the transactions during periods of low system usage, and later execute the access modules any number of times. Because the majority of applications usually execute the same application programs repeatedly, the impact on the performance of the preparation phase is amortised over many executions. However, query compilation means that there may be a significant gap between the times of binding and execution. During this interval the characteristics of data might have changed, and thus the original execution strategy might have become inefficient. A system supporting pre-compilation is system R^*.[12, 15]

Interpretation means, instead, that each query statement is received from the application program as a string at run time, and is then parsed, validated and executed. In this case the preparation phase is not really separated from the execution phase. The preparation phase must be repeated each time the transaction is executed. As pointed out in Ref. 24, this phase has a significant impact on system performance. However, late binding allows the most updated information to be used in making the optimisation decisions. A system supporting interpretation is distributed INGRES.[22]

2.1.2 *Steps during preparation phase*

The input for this phase is a transaction. The statements contained within the transaction are expressed in terms

of the global query language. The data objects referenced by the statements are those defined in the Global Schema (GS).

In the following discussion we use the term Master Site to indicate the site where the query has been submitted, while the term Apprentice Site indicates a site storing a portion of the data referenced in the query.

The following steps are executed at the Master Site for each query statement within the transaction.

(1) *Parsing.* The query is parsed by a conventional parser. Parsing verifies that the query has correct syntax. The parser output is an internal parse tree that is augmented and modified by later steps. Usually the parsing phase does not require catalogue accesses.

(2) *Catalogue lookup.* During this phase information about the global schema is brought from system catalogues. In particular, in this step information is retrieved describing the mapping between the global schema and the various local schemas. Also information is retrieved which allows determination of the location of each data object referenced in the query. Here an important issue for the performance is represented by the catalogue architecture. In fact, if the global schema is not replicated at each site, several remote accesses may be necessary.

(3) *Execution plan generation.* If the query references more than one local database, the query is fragmented into subqueries that refer only to local schemas; then a distributed processing strategy is determined. The output of this step is a schedule for the execution of subqueries. The schedule defines which subqueries can be executed in parallel and which subqueries must precede others, as well as the relationships among the intermediate results. This step requires remote accesses to the sites involved in the query, to gather information such as data sizes or available access mechanisms. This information is not likely to be replicated for reasons of autonomy. However, cache mechanisms may be used.

(4) *Plan distribution.* At this step the Master Site sends to each apprentice one or more subqueries. The plan distribution involves remote communication.

At each Apprentice Site the following steps are executed.

(1) *Parsing.*

(2) *Authorisation checking.* For each object referenced in the subquery, the apprentice checks the authorisation table to determine whether the user issuing the query has the appropriate access privileges. This step involves accessing the system catalogue.

(3) *Local optimisation.* The subquery is optimised considering the local DBMS specifics. This step necessitates accessing system catalogues to gather information about access paths and data statistics.

(4) *Code generation for subquery execution (only for pre-compilation).* In this step a routine is generated that implements the strategy chosen by the optimiser. This routine will be loaded and executed at run-time. This is a step which is not executed in a system which interprets the query after the execution strategy has been determined.

(5) *Dependency recording (only for pre-compilation).* The access module at each participant site depends upon the continued existence and validity of database objects stored at that site. These dependencies are stored in system catalogues at that site. These catalogues are searched whenever the definition of a table or an access path is changed. Access modules dependent upon these changed objects are then invalidated. The next time an invalid access module is invoked, re-compilation is automatically executed. Recording of these dependencies requiring catalogue accesses.

Steps 2, 3, 4 and 5 are executed at the Master Site also if this site stores data referenced in the query.

When transactions are pre-compiled, an access module is stored in system catalogues at the master sites as well as at the apprentices at the end of the transaction compilation. The access module is retrieved at run-time to execute the transaction. This step is omitted when queries are interpreted.

2.2 Execution Phase

At transaction execution time, the following activities are executed.

(1) *Access Module Retrieval.* The access module for the transaction is retrieved from the system catalogues. This step is executed only when queries are compiled. As we pointed out before, for interpreted queries the preparation and execution phases are not separated. Also a check is performed to verify that the access module is valid. If the access module is not valid, the transaction is recompiled.

(2) Then for each query statement contained within the transaction the following steps are executed.

(i) *Subquery Execution.* In this step the various subqueries are executed. The execution schedule defined during the preparation phase is used to coordinate subquery execution. The query master site sends to the various apprentices the control messages needed to coordinate the query execution. Also each apprentice has to load the access module if the statement being executed is the first. For the subsequent statements it is not necessary to load the access module since it is already in memory. The main impact on the performance during this step is represented by the communication delays. Data movements among sites are in fact needed to execute joins and to gather the final data at the query master site. However, the impact of I/O and CPU processing may not be negligible. It has been shown that, given the frequency with which these operations are requested in a 'typical' query that joins two tables at different sites, CPU and I/O delays can both dominate message delays.[18]

(ii) *Result Integration.* At the query master site, the results are gathered and presented to the user. Again this involves I/O overhead due to temporary storage of data.

2.3 Commit Phase

We describe here the various steps for the basic 2-phase commitment (2PC) protocol.[11] However, many other variations of such protocols have been defined.[9, 16, 21] In the basic version there is a site which has the role of coordinator, while the other sites are indicated as participants. The basic idea of the 2PC is to determine a unique decision for all the participants with respect to committing or aborting all the local subtransactions. The protocol consists of two phases. The goal of the first phase is to reach a common decision; the goal of the second one is to implement this decision.

(1) *Phase One*. During the first phase, the coordinator asks all the participants to prepare for commitment; each participant answers READY if it is ready to commit. Before sending the first 'prepare for commitment' message, the coordinator records on stable storage a log record, in which the identifiers of all the participants are recorded. The coordinator also activates a timeout mechanism, which will interrupt the coordinator after a given time interval has expired. When a participant answers READY, it ensures that it will be able to commit the local subtransactions even if failures occur at its site. In practice, this means that each participant has to record on stable storage two items, as follows.

- All the information required for locally committing the subtransactions. This means that all the log records of the subtransactions must be recorded on stable storage.
- The fact that this subtransaction has been declared ready to commit. This means that a log record of type 'ready' must be recorded on stable storage.

The coordinator decides whether to commit or abort the transaction as a result of the answers which it has received from the participants. If all participants have answered READY, it decides to commit the transaction. If instead some participant has answered ABORT or has not yet answered when the timeout expires, it decides to abort the transaction.

(2) *Phase Two*. The coordinator begins the second phase of the 2PC by recording on stable storage its decision. This corresponds to writing a 'global_commit' or 'global_abort' record in the log. The fact that the coordinator records its decision on stable storage means that the distributed transaction will eventually be committed or aborted, in spite of failures. Then the coordinator informs all participants of its decision, by sending them a command message.

All the participants write a commit or abort record in the log, based on the command message received from the coordinator. From this moment, the local recovery procedure is capable of ensuring that the effect of the subtransaction will not be lost.

Finally, all participants send a final acknowledgement (ACK) message to the coordinator, and perform the actions required for committing or aborting the subtransaction. When the coordinator has received an ACK message from all participants, he writes a log record, called 'complete' record. After having written this record, the coordinator can forget the outcome of the transaction; thus all records related to this transaction can be taken offline after the next checkpoint.

The commit phase involves exchanging control messages among sites to reach the final agreement. Such messages are rather short, and the main overhead here is due to message initiation. Also a considerable amount of local processing may be required to execute actions such as transaction update logging.

3. ANALYSIS OF PERFORMANCE

In this section we present an analysis of the performance for a DDBMS that processes transactions according to the model described in the previous section. In the present analysis we evaluate the performance in terms of response time for transactions. The response time of a transaction is defined as the difference between the time of transaction end and the time of transaction initiation.

To keep the analysis fairly simple, we make the following assumptions.

The Global schema is replicated. This means that to retrieve information contained in this schema only local accesses are needed.

The query execution strategy is rather simple. In particular no distributed joins are executed. The various subqueries are executed in parallel at the participant sites and the results of subqueries are assembled at the query master site. Joins are executed at the master site. An extension of the model is planned to take into account different join strategies.

No cache mechanism is used. This hypothesis is expounded in a forthcoming paper.[3]

For the network we have considered two cases: (1) it is possible to send messages in parallel on the network; (2) it is not possible to send messages in parallel and therefore only one site at a time can transmit messages.

We have assumed three different message sizes: *short* (up to 200 bytes) – this type of message is used to send control information such as in the commit phase; *medium* (700–1000 bytes) – this type of message is used to send information such as catalogue information or query execution plans; *long* (more than 1000 bytes) – this type of message is used in general to transmit data in response to the user queries.

We assume that the statements within a single transaction are processed and executed in serial order. That is, given a transaction, the system processes a statement at the time from that transaction.

We assume that the catalogues are stored in the database as normal data. For instance in a relational DBMS, catalogue tables would be stored as relations.

In the evaluation that we present in the next subsections, we don't take into account the impact of the concurrency control mechanisms. In Subsection 4.6 the effect of the concurrency control on the overall performance is discussed.

We assume that the processing speeds and I/O access times are equal for all the sites.

3.1. Model Parameters

In this section we describe the parameters that we use in the analysis.

N	number of statements within a transaction
N'	average number of statements effectively executed at a transaction run ($N' \leqslant N$)
ND	average number of data objects referenced by a statement
NS	average number of remote sites storing data objects referenced by a statement
NRD	average number of remote data objects; by remote data object we mean one stored at a site different from the master site
$NRPS$	average number of remote data objects per site; $NRPS = NRD/NS$
NSU	number of sites where the transaction performs updates
VDU	number of data pages updated per site by the transaction. A page is the unit of data transferring when executing an I/O operation

VDA	average number of data pages accessed for a subquery execution
VDT	average volume of data transmitted as subquery result; this volume is expressed in bytes
P	probability that a transaction must be re-compiled on line; this parameter is significant only for the compilation case.
PS	average page size in bytes
I/O	page access time
TP	execution time for parsing a query
TDQ	execution time for distributed query optimisation
TLQ	execution time for local query optimisation
TCG	execution time for code generation
ST	startup time for sending a message; this time is independent of message size
TB	transmission time per byte
$TT(x)$	transmission time for an x bytes message; $TT(x) = ST + TB * x$
TS	transmission time for a short message $TS = ST + TB * 200$
TM	transmission time for a medium message $TM = ST + TB * 1000$

3.2 Estimation of the Preparation Phase

3.2.1 *Compilation*

Given a transaction T containing a number N of data access and manipulation statements, the response time for the preparation phase when queries are compiled is given by:

$$T(ETPP) = N * T(PP) + T(SAM)$$

where $T(PP)$ is the response time for the preparation of a single statement and $T(SAM)$ is the time to store the access module for the transaction.

The response time for the preparation phase of a statement is calculated as follows:

$T(PP) = T$ (parsing)
+ T (local catalogue lookup)
+ T (remote catalogue lookup)
+ T (generation of the execution plan)
+ T (plan distribution)
+ $MAX\{T_i = T$ (work at remote apprentice i)$\}$
+ (receiving answers from apprentices)

(1)

where

T (local catalogue lookup) $= ND * \text{I/O}$

We suppose that for each data object referenced by the statement, a disk access is performed to collect information about that object. Actually some of the catalogue pages might be present in the system buffers. In such a case the number of disk accesses is lower.

T (remote catalogue lookup) $= T$ (to send a number NS of short messages)
+ T (local catalogue lookup at each remote site)
+ T (to send a number NS of medium messages)

Here we distinguish between two cases:

(1) Parallel transmission of messages
T (remote catalogue lookup) $= TS + NRPS * \text{I/O} + TM$

(2) Non-parallel transmission of messages
T (remote catalogue lookup) $= NS * TS + NRPS * \text{I/O} + NS * TM$
T (plan distribution) $= T$ (time to send NS medium messages)

Again we distinguish between cases:

(1) Parallel transmission of messages
T (plan distribution) $= TM$

(2) Non-parallel transmission of messages
T (plan distribution) $= NS * TM$

Since the work at apprentices is carried out in parallel and all sites have the same processing speed the expression

$MAX\{T_i: T_i = T$ (work at remote apprentice i)$\}$

can be approximated by the expression

T (work at remote apprentice i), where
T (work at remote apprentice i) $= T$ (parsing)
+ T (local catalogue lookup for authorisation)
+ T (local catalogue lookup for local optimisation)
+ T (local optimisation)
+ T (code generation)
+ T (local catalogue access for dependency recording)

We obtain

T (work at remote apprentice i) $= TP + TLQ + TCG + 3 * NRPS * \text{I/O}$
T (receiving answers from the apprentices) $= T$ (to receive NS short messages)

Here we distinguish two cases:

(1) Parallel transmission of messages
T (receiving answers) $= TS$

(2) Non-parallel transmission of messages
T (receiving answers) $= NS * TS$

Substituting the previous expressions into the expression (1) we obtain the following expression for the execution of the preparation time.

(1) Parallel transmission of messages
$T(PP) = 2 * TP + TDQ + TLQ + TCG + \text{I/O} * (ND + 4 * NRPS) + 2 * TS + 2 * TM$

(2) Non-parallel transmission of messages
$T(PP) = 2 * TP + TDQ + TLQ + TCG + \text{I/O} * (ND + 4 * NRPS) + 2 * NS * TS + 2 * NS * TM$

The time for storing the access modules can be approximated as the time for executing an I/O operation, therefore

$T(SAM) = \text{I/O}$.

This activity is carried in parallel at the master and apprentices.

3.2.2 *Interpretation*

The response time for the preparation phase in case of interpretation is given by:

$T(ETPP) = N' * T(PP)$

For interpretation the preparation phase must be carried only for the statements which are effectively executed when the transaction is run. The average number of statements executed is indicated in our model by N' (see Subsection 3.1). In addition, the activity of code generation is not executed and the access modules and dependencies are not recorded.

We obtain the following expression for $T(PP)$:

(1) Parallel transmission of messages

$$T(PP) = 2*TP + TDQ + TLQ + I/O*(ND + 2*NRPS) + 2*TS + 2*TM$$

(2) Non-parallel transmission of messages

$$T(PP) = 2*TP + TDQ + TLQ + I/O*(ND + 2*NRPS) + 2*NS*TS + 2*NS*TM$$

3.3 Estimation of the Execution Phase

3.3.1 *Compilation*

The response time for the execution phase of a transaction T when queries are compiled is given by:

$$T(ETEP) = T(AMR) + N'*T(EP) \quad (2)$$

where $T(AMR)$ is the time needed to retrieve the access module at the master site and $T(EP)$ is the time for executing a single statement.

$$T(AMR) = I/O.$$

The execution time for a statement is calculated as follows:

T(EP) = T (send messages to participant to activate subquery execution)
+ $MAX\{T_i: T_i = T$ (work at remote apprentice i)$\}$
+ T (receiving results)
+ T (result integration)

where

T(send messages to activate subquery execution) = T(to send a number NS of short messages)

Here we distinguish between two cases:

(1) Parallel transmission of messages
T (to send a number NS of short messages) = TS

(2) Non-parallel transmission of messages
T (to send a number NS of short messages) = $NS*TS$

Since the work at apprentices is carried out in parallel and all sites have the same processing speed the expression

$MAX\{T_i: T_i = T$ (work at remote apprentice i)$\}$

can be approximated by the expression

T {work at remote apprentice i) where

T (work at remote apprentice i) $= T$ (access module retrieval) $+ T$ (query execution) $= I/O*(1 + VDA)$

if this statement is the first being executed of transaction T; else:

T (work at remote apprentice i) = T(query execution) $= I/O*VDA$

T (receiving results) = T (receiving VDT bytes on the network from NS sites)

Here we distinguish between two cases:

(1) Parallel transmission of messages
T (receiving results) = $TT(VDT)$

(2) Non-parallel transmission of messages
T (receiving results) = $NS*TT(VDT)$
T (result integration) = $I/O*[NS*(VDT/PS)]$†

Substituting the previous expressions into expression (2) we obtain the following expression for the response time of the execution phase.

(1) Parallel transmission of messages

$$T(ETEP) = I/O*(2 + N'*VDA + N'*[NS*(VDT/PS)]) + N'*TT(VDT) + N'*TS$$

(2) Non-parallel transmission of messages

$$T(ETEP) = I/O*(2 + N'*VDA + N'*[NS*(VDT/PS)]) + NS*N'*TT(VDT) + NS*N'*TS$$

3.3.2 *Interpretation*

The execution time for a statement in the case of interpretation is evaluated as in the case of compilation. The only difference is that the access modules are not retrieved from system catalogues. Therefore there will be less catalogue accesses.

We obtain the following expression for $T(ETEP)$:

(1) Parallel transmission of messages

$$T(ETEP) = I/O*(N'*VDA + N'*[NS*(VDT/PS)]) + N'*TT(VDT) + N'*TS$$

(2) Non-parallel transmission of messages

$$T(ETEP) = I/O*(N'*VDA + N'*[NS*(VDT/PS)]) + NS*N'*TT(VDT) + NS*N'*TS$$

3.4. Estimation of the Commit Phase

In this subsection we evaluate the execution time for the commit phase in a situation where there are no crashes. Given a transaction T, the execution time for the commit phase is evaluated as follows:

$T(CP) = T$ (store a log record containing subordinate names)
+ T (send a READY message)
+ T (remote subordinate work phase I)
+ T (receiving answers from subordinates)
+ T (record on stable storage decision about transaction)
+ T (send decision to subordinates)
+ T (remote subordinate work phase II)
+ T (receiving acknowledgements from subordinates)
+ T(store a 'complete' on stable storage) (3)

where

T (remote subordinate work phase I) = T (record on log transaction updates) + T (write a 'ready' record) = $I/O*(VDU + 1)$

T (remote subordinate work phase II) = T (write a 'commit' record) = I/O

Substituting the previous expressions into expression (3), we obtain the following expression for the execution time of the commit phase:

(1) Parallel transmission of messages

$$T(CP) = I/O*(VDU + 4) + 4*TS$$

(2) Non-parallel transmission of messages

$$T(CP) = I/O*(VDU + 4) + 4*NSU*TS$$

3.5 Estimate of Transaction Response Time

Given a transaction T containing a number N of data access and manipulation statements, such that a number $N'(N \leqslant N)$ of statements are executed at transaction

† $[x]$ indicates the smallest integer greater than or equal to x.

run-time, the transaction response time is given by the following expressions:

(1) Compilation

$T(ETT) = T(ETEP) + T(CP)$ with probability $q = (1-p)$

(q is the probability that the transaction does not have to be re-compiled)

$T(ETT) = T(ETPP) + T(ETEP) + T(CP)$ with probability p

(p is the probability that the transaction has to be re-compiled)

We obtain the following average value for $T(ETT)$, that we indicate as $T'(ETT)$:

$$T'(ETT) = T(ETEP) + T(CP) + p*(ETPP)$$

Parallel transmission of messages

$$\begin{aligned} T'(ETT) = {} & I/O*[N'*(VDA+[NS*(VDT/PS)]) \\ & + VDU+6+p+N*(ND+4*NRPS)*p] \\ & + TS*[4+N'+2*p*N] \\ & + TT(VDT)*N' \\ & + TM*[2*p*N] \\ & + p*N*[2*TP+TDQ+TLQ+TCG] \end{aligned}$$

Non-parallel transmission of messages

$$\begin{aligned} T'(ETT) = {} & I/O*[N'*(VDA+[NS*(VDT/PS)]) \\ & + VDU+6+p+N*(ND+4*NRPS)*p] \\ & + TS*[4*NSU+N'*NS+2*p*N*NS] \\ & + TT(VDT)*N'*NS \\ & + TM*[2*p*N*NS] \\ & + p*N*[2*TP+TDQ+TLQ+TCG] \end{aligned}$$

(2) Interpretation

$T(ETT) = T(ETPP) + T(ETEP) + T(CP)$

Substituting the expressions we have obtained in the previous subsections we obtain the following expressions:

Parallel transmission of messages

$$\begin{aligned} T(ETT) = {} & I/O*[N'*(VDA+[NS*(VDT/PS)]+ND \\ & +2*NRPS) \\ & + VDU+4] \\ & + TS*(3*N'+4) \\ & + TT(VDT)*N' \\ & + TM*2*N' \\ & + N'*[2*TP+TDQ+TLQ] \end{aligned}$$

Non-parallel transmission of messages

$$\begin{aligned} T(ETT) = {} & I/O*[N'*(VDA+[NS*(VDT/PS)]+ND \\ & +2*NRPS) \\ & + VDU+4] \\ & + TS*(3*N'*NS+4*NSU) \\ & + TT(VDT)*N'*NS \\ & + TM*2*N'*NS \\ & + N'*[2*TP+TDQ+TLQ] \end{aligned}$$

4. RESULTS

Using the expressions derived in the previous section, we have evaluated the transaction response time in various cases to compare the pre-compilation approach versus the interpretation. In all the evaluations presented in the present paper we have considered the case of parallel transmission of messages.

In all the evaluations performed we have used the following values for I/O and network parameters:[15]

- I/O = 23.48, milliseconds/IO

  This value is typical of an IBM 3380 disk drive; this time includes initiation, seek, latency and transfer time.
- PS = 4096 bytes
- ST = 11.54 milliseconds/message
- TB = 0.16 milliseconds/byte

These values for ST and TB are typical of a 50 kilobit/second communication line.

It is more difficult to determine average times for query parsing, optimisation, and code generation, because such times depend on the query being processed. In the present work we have assumed the following average values (milliseconds): TP 15; TDQ 70; TLQ 50; TCG 17 milliseconds.

To compare the compilation approach versus the interpretation, we have estimated the transaction response time for different values of the number of statements contained within the transaction. We have considered three cases regarding the size the data accessed (VDA) and transmitted per statement (VDT):

Case I, low values for VA, low values for VDT: VDA = 1 page and VDT = 4096 bytes.

Case II, high values for VDA, high values for VDT: VDA = 10 pages and VDT = 40,960 bytes.

Case III, high values for VDA, low values for VDT: VDA = 10 pages and VDT = 4096 bytes.

For the compilation case we have assumed three different values for the probability of re-compilation:

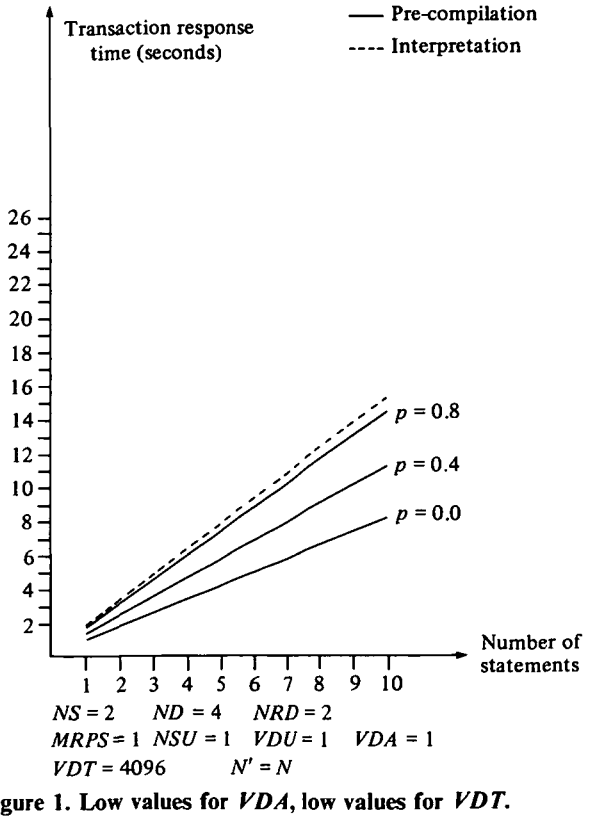

Figure 1. Low values for VDA, low values for VDT.

p = 0.0, 0.4, 0.8. The value p = 0.0 means that the transaction is never re-compiled on-line. In this case the re-compilation takes place off-line and therefore it does not impact on the transaction response time.

Finally we have considered two cases concerning the values of N and N'.

Case (a) $N' = N$ (figs. 1, 2, 3): in this case all the statements within the program are executed.

Case (b) $N' = [N/2]$ (figs. 4, 5, 6): in this case about half of the statements within the program are executed.

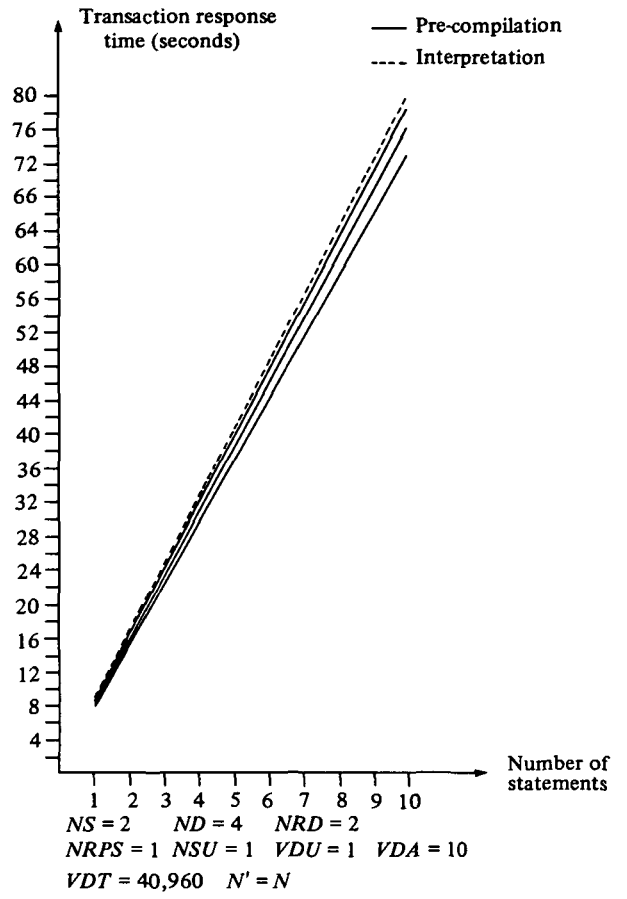

Figure 2. High values for *VDA*, high values for *VDT*.

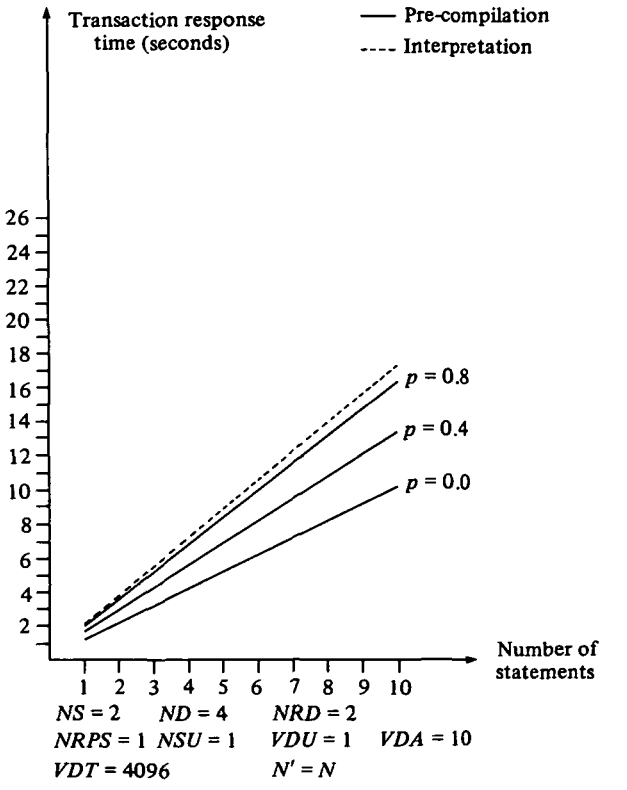

Figure 3. High values for *VDA*, low values for *VDT*.

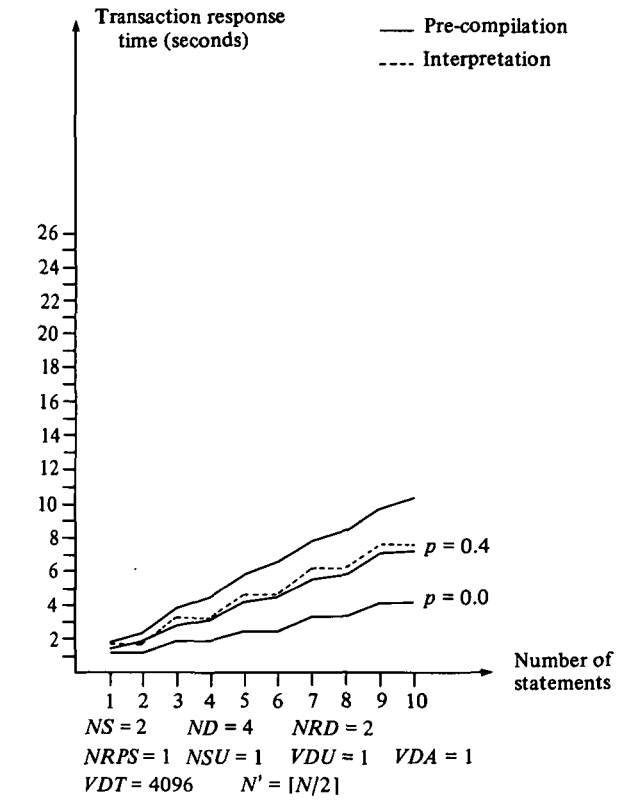

Figure 4. Low values for *VDA*, low values for *VDT*.

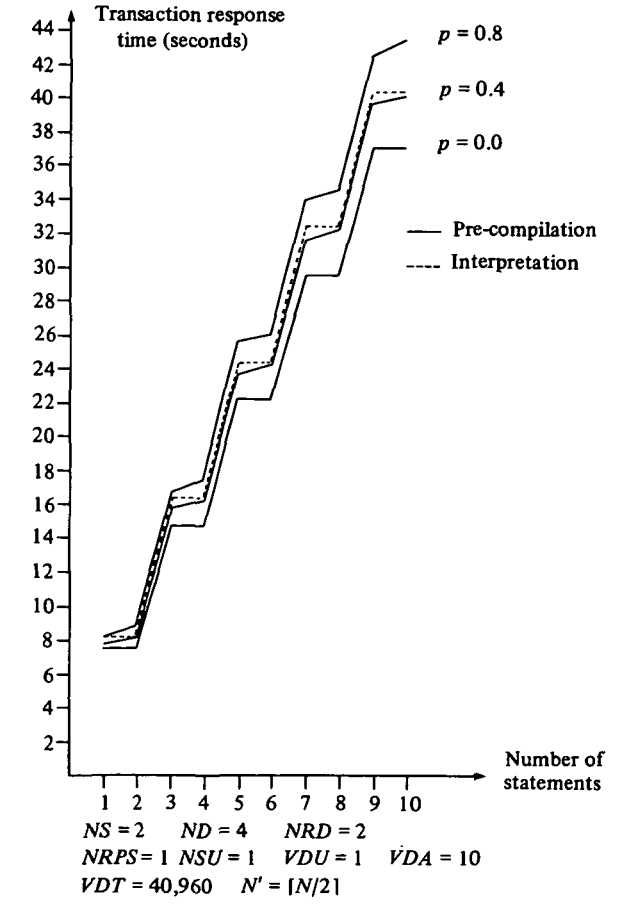

Figure 5. High values for *VDA*, high values for *VDT*.

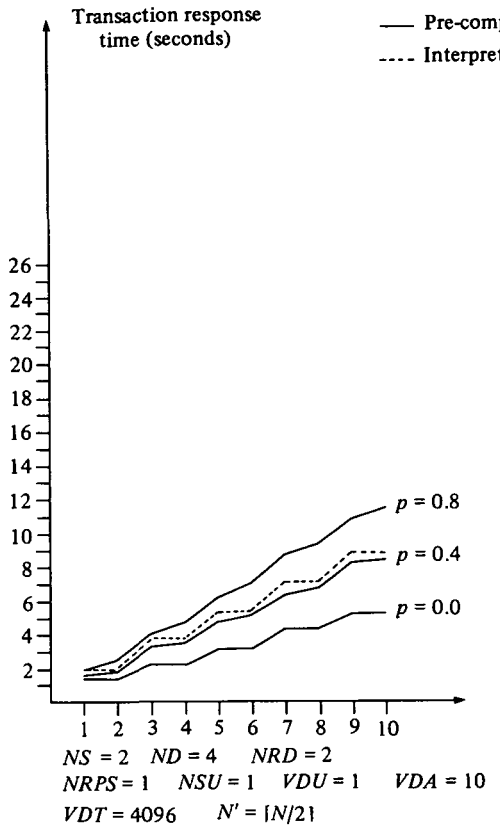

Figure 6. High values for *VDA*, low values for *VDT*.

The results reported in figs. 1, 2 and 3 show that when $N = N'$ pre-compilation and interpretation have the same performance when $p = 0.8$. The difference between the execution time for pre-compilation ($p = 0.0$ and $p = 0.4$) increases with the number of statements within the transaction. However, when statements involve large amounts of data transmitted (fig. 2) the difference in performance is less marked. For instance, in the case of $N = 10$ and $VDT = 40{,}960$ we obtain:

$$(T(ETT(\text{interpr.}) - T(ETT(\text{pre-comp. } (p = 0.0)))/T(ETT(\text{interpr.})) = 9.2\%$$

However, when $VDT = 4096$ we obtain:

$$(T(ETT(\text{interpr.}) - T(ETT(\text{pre-comp. } (p = 0.0)))/T(ETT(\text{intepr.})) = 45\%$$

The results reported in figs. 4, 5 and 6 show that when $N' = \lceil N/2 \rceil$ pre-compilation and interpretation have about the same performance when $p = 0.4$. The difference the previous case ($N' = N$) can be explained by observing that when a transaction is re-compiled, all the statements within the transaction are re-compiled again. In the interpretation case, instead, only the statements which are executed go through the preparation phase.

Therefore we conclude that the pre-compilation performs better than the interpretation if (1) the probability of re-compilation is less than $p = 0.8$ for $N' = N$ ($p = 0.4$ for $N' = \lceil N/2 \rceil$) and (2) the transaction contains a large number of statements that involve a small amount of data.

4.1. Concurrency Control

In the previous expressions to keep the analysis fairly simple, we have not taken into account the impact of the concurrency control mechanisms on the overall transaction execution time.

Here we discuss some aspects related to the impact of the concurrency control activity. To evaluate such an impact on the execution time of a transaction two factors must be considered.

- The overhead due to the execution of lock operations, such as lock-acquire and lock-release; these operations are executed by the lock manager.
- The delay caused by conflicts with other transactions accessing the same data.

It should be pointed out that the overhead due to the operations is independent from the degree of interference among transactions. This overhead is proportional, instead, to the number of data accessed by the transaction and to the lock granularity. In System R a lock–unlock pair takes the execution of 350 instructions.[11]

In particular, if the granularity is at page level and the CPU processing speed is 0.000769 milliseconds/instruction,[15] we obtain the following delay for accessing a number *VDA* of pages:

$$\text{Delay } (CC) = VDA * 0.269 \text{ milliseconds}$$

For instance, if each local subquery accesses 10 pages, the local overhead due to lock operations is 2.69 milliseconds. This impact seems to be rather slight compared to the transaction response times.

The second factor that should be considered is the delay introduced by conflicts among transactions. This delay depends on the degree of interference among transactions and on the transaction arrival frequency. The degree of interference states how much the data accessed by a transaction intersect with data accessed by other transactions. The delay due to transaction conflicts may not be negligible and it may increase considerably the transaction execution time.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have described a simple model to evaluate certain aspects of DDBMS performance. In particular we have addressed the problem of comparing the pre-compilation approach with the interpretation. The results presented here are part of a study aiming to determine the factors that most impact on the performance of a distributed database management system. In particular the model will be extended to evaluate different catalogue architectures and various query-processing algorithms.

REFERENCES

1. E. Bertino, C. Carlesi and C. Thanos, Performance evaluation of two-phase locking algorithms in a system for distributed databases. In *Proc. Third IEEE Symposium on Reliability in Distributed Software and DB Systems, Clearwater Beach, Florida, October 18–19, 1983.*
2. E. Bertino, *Identification of the Factors that Impact the Performance of a Distributed Database Management System.* I.E.I.–C.N.R. internal report.
3. E. Bertino, *An Evaluation of Catalog Architecture for Distributed Database Management Systems.* In preparation (1985).
4. H. Boral and D. J. DeWitt, A methodology for database

system performance evaluation. In *Proc. ACM-SIGMOD Conference, Boston, June 1984.*

5. Y. J. Breibart, L. F. Kemp, G. R. Thompson and A. Silberschatz, Performance evaluation of a simulation model for data retrieval in a heterogeneous database environment. In *Proc. International Trends and Applications Conference, Gaithersburg (Maryland), May 1984.*
6. Y. J. Breibart, L. R. Tieman, ADDS – heterogeneous distributed database system. In *Distributed Data Sharing Systems*, edited F. A. Schreiber and W. Litwin, pp. 7–24. North-Holland, Amsterdam (1985).
7. M. Carey and M. Stonebraker, The performance of concurrency control algorithms for database management systems. In *Proc. Tenth International Conference on Very Large Data Bases, Singapore, August 1984.*
8. D. Chamberlin, M. Astrahan, W. King, R. Lorie, J. Mehl, T. Price, M. Schkolnick, P. Selinger, D. Slutz, B. Wade and R. Yost, Support for repetitive transactions and ad hoc queries in system R. *ACM Trans. Database Systems* (1), 70–94 (1981).
9. D. Eager and K. Sevcik, Achieving robustness in distributed database systems. *ACM Trans. Database Systems* **8** (3) (1983).
10. H. Garcia-Molina, Performance comparison of two update algorithms for distributed databases. In *Proc. Third Berkeley Workshop on Distributed Data Management and Computer Networks, Berkeley (Calif.), Aug. 1978.*
11. J. N. Gray, Notes on database operating systems. In *Operating Systems: An Advanced Course*. Springer-Verlag, Heidelberg (1979).
12. L. Haas, P. Selinger, E. Bertino, D. Daniels, B. Lindsay, G. Iohman, Y. Masunaga, C. Mohan, P. Ng, P. Wilms and R. Yost, R*: a research project on a distributed relational DBMS. *Database Engineering* **5** (4) (1982).
13. C. H. Lee, Queuing analysis of global locking synchronization schemes for multicopy databases. *IEEE Trans. Software Eng.* **C-29** (5) 371–384 (1980).
14. W. K. Lin and J. Nolte, Performance of two-phase locking. In *Proc. Sixth Berkeley Workshop on Distributed Data Management and Computer Networks, Berkeley (Calif.), Feb. 1982.*
15. G. Lohman, C. Mohan, L. Haas, B. Lindsay, P. Selinger, P. Wilms and D. Daniels, Query processing in *R**. In *Query Processing in Database Systems.* edited W. Kim, D. Reiner and D. Batory. Springer-Verlag, Heidelberg (1985). Also IBM Research Report RJ 4272, San Jose, Calif. (1984).
16. C. Mohan and B. Lindsay, Efficient commit protocols for the tree of processes model of distributed transactions. *Proc. Second ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Montreal, 1983.*
17. D. Ries, The effects of concurrency control on the performance of a distributed management system. In *Proc. Fourth Berkeley Workshop on Distributed Data Management and Computer Networks, Berkeley (Calif.), Aug. 1979.*
18. P. Selinger and M. Adiba, *Access Path Selection in Distributed Database Management Systems.* IBM Research Report RJ 2883, San Jose, Calif. (1980).
19. K. C. Sevcik, Data base system performance using an analytical model. *Proc. Seventh Conference on Very Large Data Bases, Cannes, 1981*, pp. 182–198.
20. K. C. Sevcik, Comparison of concurrency control methods using analytical models. *Proc. IFIP Conference, Paris, Sept. 19–23, 1983.*
21. D. Skeen, A. quorum-based Commit protocol. *Sixth Int. Workshop on Distributed Data Management and Networks* (1982).
22. M. Stonebraker and E. Neuhold, A distributed version of INGRES. *Proc. Second Berkeley Workshop on Distributed Data Management and Computer Networks, May 1977.*
23. M. Stonebraker, J. Woodfill, J. Ranstrom, J. Kalash, K. Arnold and E. Andersen, Performance analysis of distributed data base systems. In *Proc. Third Symposium on Reliability in Distributed Software and Database Systems, Clearwater Beach, Florida, October 17–19, 1983.*
24. M. Stonebraker, J. Woodfill, J. Ranstrom, M. Murphy, M. Meyer and E. Allman, Performance enhancements to a relational database system. *ACM Trans. Database Systems*, **8** (2), 167–185 (1983).
25. Y. Tay and R. Su, Choice and performance in locking for databases. In *Proc. Tenth International Conference on Very Large Data Bases, Singapore, August 1984.*
26. R. Thomas, A majority consensus approach to concurrency control for multiple copy databases. *ACM Trans. Database Systems* **4** (2) (1979).
27. A. Thomasian, Performance evaluation of centralized databases with static locking *IEEE Trans. Software Engineering* **SE-11** (4), pp. 346–355 (1985).