ISSN 0254–2757

QI QUÆSTIONES INFORMATICÆ

Volume 6 • Number 2 September 1988

The official journal of the Computer Society of South Africa and of the South African Institute of Computer Scientists

Die amptelike vaktydskrif van die Rekenaarvereniging van Suid-Afrika en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

QUÆSTIONES INFORMATICÆ

The official journal of the Computer Society of South Africa and of the South African Institute of Computer Scientists

Die amptelike vaktydskrif van die Rekenaarvereniging van Suid-Afrika en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

Subscriptions

The annual subscription is

	SA	US	UK
Individuals	R20	$ 7	£ 5
Institutions	R30	$14	£10

to be sent to:
Computer Society of South Africa
Box 1714 Halfway House 1685

Quæstiones Informaticæ is prepared by the Computer Science Department of the University of the Witwatersrand and printed by Printed Matter, for the Computer Society of South Africa and the South African Institute of Computer Scientists.

Modelling Distributed Database Concurrency Control Overheads

MH Rennhackkamp and SH Von Solms
Department of Computer Science, University of Stellenbosch, Stellenbosch, 7600
Department of Computer Science, Rand Afrikaans University, PO Box 524, Johannesburg, 2000

Abstract

Numerous concurrency control methods have been proposed for distributed databases. Various criteria are used to compare these methods. The comparisons range from qualitative overviews through quantitative analyses to theoretical studies. A quantitative study based on an abstract model of concurrency control methods is presented, where the overheads of the methods are analytically compared using a set of evaluation parameters.
After an overview of the model's development, it is presented in detail. As an example it is applied to two-phase locking as can be used in a fully-redundant distributed environment. It is concluded that although the model has shortcomings, it does provide a framework according to which distributed database concurrency controls can be compared.

Received August 1987, Accepted July 1988

1. Introduction

Many distributed database concurrency control methods have been proposed to coordinate simultaneous database accesses at dispersed sites by multiple users; in an attempt to exploit parallelism maximally to provide efficient data accesses. It is not clear which of these methods is superior; due to the diversity of the methods and the mechanisms of locking, (timestamp) ordering and optimism they are based on.

Numerous evaluations and comparitive studies have been made. The criteria for these comparisons cover a variety of aspects and the actual evaluations range from qualitative overviews, through simulations and analytical models to detailed theoretical studies.

A quantitative evaluation of concurrency control overheads is presented, based on a uniform representation of the methods, using an abstract relational database model. The analytical evaluation uses parameters to depict the CPU, storage and message overheads.

After an overview of the model's development, it is presented in detail. A simple two-phase locking (2PL) method with the two-phase commit (2PC) protocol incorporated, as can be used in a fully-redundant distributed database, is used as an example. After noting some of the virtues and shortcomings of the model, it is concluded that the model does facilitate conclusive comparisons.

2. Background

Carey [2] presented an abstract model according to which centralised database concurrency control algorithms can be represented and analytically evaluated. The algorithms are depicted uniformly in implementation independent terms to evaluate the overheads incurred by each method. He concluded his presentation with suggestions for extensions to the model, to consider the methods used in multiple copy and distributed databases.

These extensions have been applied, as well as other changes to the model to depict the methods more clearly [4]. Carey's evaluations did not consider conflicting transactions, but this was done in the subsequent study by extending the set of parameters. Conflicts are only considered once, meaning the conflicts of re-executed transactions are not considered. Overheads due to abnormal conditions, such as transmission failures, are neither taken account of.

3. The model

The model functions on the premise that concurrency control methods can be described terms of the information they need, the conditions when conflicts occur and the way in which transactions are processed. The overhead costs of each method can then be determined in implementation independent units of cost by analysing the uniform representations.

The model does not take recovery into consi-

deration; e.g. when a site indicates its readiness to commit, it is assumed that the site will commit when commanded to do so. However, it does take conflicts into consideration, where two transactions wish to access the same data item simultaneously.

3.1 Transaction structure

A transaction is represented by the following primitives:

```
BEGIN(t-id)
READ-REQ(t-id, obj-id) or
...
WRITE-REQ(t-id, obj-id)
...
COMMIT(t-id)
```

There may be many read and write operations in a transaction. The primitive components of transaction requests are sent to the remote sites where the local operations are executed. The global part (at the transaction's site of origin) controls the synchronisation with the other sites, while the local parts maintain the local concurrency control.

3.2 Concurrency control database

The information about transactions, requests and data objects is stored in a collection of relations, called the concurrency control database (CC-DB). It represents all the information any concurrency control algorithm may need and forms the cornerstone of the model, with which all the methods can be described uniformly; the differences between methods are lessened by representing all the information in this structure. All the algorithms do not necessarily use all the fields and relations of the CC-DB. A description of the relations follows.

TSI(t-id, state, timestamp)
The transaction state information relation contains fields for the transaction identifier, its state (ready, blocked, committed or aborted) and a possible transaction timestamp. It documents all active transactions and their states; an optional timestamp can be associated with each transaction.

ACC(obj-id, t-id, mode, timestamp)
This relation documents accesses to data objects, using fields for the object identifier. transaction identifier, mode of access and a possible object timestamp. The object and transaction identifiers document accesses; the mode and optional timestamp fields are used for exclusion purposes.

BLKD(t-id, cause-id)
The blocked relation documents blocked transactions. It contains the identifiers of the blocked transactions and the transactions they wait for. It is used in the locking based methods to depict the deadlock resolution information.

HIST(obj-id, t-id, mode, timestamp)
The history relation stores information about conditionally granted accesses, mostly in the optimistic methods. It contains similar fields to the ACC relation.

SITE(obj-id, site-id)
This relation documents the sites of data objects, which are normally kept in the data dictionary. It can contain an entry for each site of each data object.

3.3 Macros

Sequences of operations on the CC-DB are called macros; these are used to depict the primitive operations executed during the processing of a transaction.

The CC-DB operations are presented as query language statements; using decision (if) and iteration (while) structures. Comments are delimited by curly brackets. Pseudocode is also used liberally.

Some general macros, namely to block, restart and entirely remove (expunge) transactions are used; a boolean macro CYCLE(t-id) is also used to detect whether a transaction is involved in a deadlock cycle depicted in the blocked relation.

Blocking

When a transaction is blocked, its state is changed and the cause of its blocking is documented in the BLKD relation.

```
BLOCK(t-id, cause-id):
replace TSI(state := "blocked")
    where TSI.t-id = t-id
append BLKD(t-id, cause-id)
```

Expunging

When a transaction is expunged, it is removed from the CC-DB.

```
EXPUNGE(t-id):
delete TSI
    where TSI.t-id = t-id
delete ACC
    where ACC.t-id = t-id
delete BLKD
    where BLKD.t-id = t-id
    or BLKD.cause = t-id
delete HIST
    where HIST.t-id = t-id
```

Restarting
When a transaction is restarted, its state is changed and it is removed to nullify its effects. This macro is also used for aborting a transaction.

```
RESTART(t-id):
replace TSI(state := "aborted")
    where TSI.t-id = t-id
Undo all the DB changes
EXPUNGE(t-id)
```

Communication
Messages must be sent to the indicated sites. A command or macro is not explicitly received; it is merely executed at the indicated sites.

```
SEND (message, sites):
    {Send the message or command to
        the indicated site(s)}
```

A macro must be used where a returned message must be explicitly received.

```
RECEIVE(message, sites):
    {Receive message(s) from the
        indicated site(s)}
```

3.4 Concurrency control modelling
The concurrency control methods are described by the following.

Concurrency control database
Each concurrency control method makes use of different structures to represent concurrency control information. The CC-DB fields and relations used must therefore be specified for evaluation purposes.

Conflict situations
The conditions under which blocking and restarting take place vary for each method; it must be specified which of these situations occur, as well as under which conditions. Boolean typed macro functions are used for this purpose.

Macros
Different CC-DB operations are executed in each method to represent the execution of the transaction initiation, access request and commitment primitives. A major component of the model is to describe these operations in a uniform manner using macros.

3.5 Parameters
Transaction volumes are quantified by the following parameters:

T_a: The average number of transactions in the system.

F_c: The average fraction of conflicting transactions. The meaning of this parameter depends on the context where it is used. In the 2PL cases, it means the fraction of blocked transactions which wait for locked data items. In the timestamp ordering (TSO) methods it enumerates the fraction of the transactions which must be restarted due to conflicts with transactions with more recent timestamps. In the optimistic methods it depicts the fraction of the transactions restarted because they did not satisfy the validation criteria.

F_{fc}: The average fraction of fatally conflicting transactions. In the 2PL methods, it represents the fraction of transactions which become blocked and cause deadlock cycles; in the TSO methods it depicts transactions restarted during commitment.

F_{rc}: The average fraction of recently committed transactions. It is used to account for overheads incurred after commitment. In the optimistic methods it represents the transactions which have committed since the startup time of the oldest active transaction.

Different access requests are executed in different ways, causing different overhead costs. The following parameters are used to identify fractions of the requests:

R_r: The average number of read requests per transaction.

R_w: The average number of write requests per transaction.

R_a: The average number of general access requests per transaction. Where equal, $R_r + R_w = R_a$ is used to simplify evaluation expressions.

D: The cost of deadlock detection at a single site. Distributed deadlock detection cost is not easily assessed; it depends greatly on the method used. It will be assumed that it is proportional to D at every site.

The following parameters are used to depict the sites.

S: The total number of sites in the system.

F_s: The average fraction of the sites where data objects reside. It augments the previous parameter in methods where data is not necessarily fully-redundantly duplicated; it represents the average duplication factor.

O_a: The average number of data objects in the

system. It represents the size of the data dictionary, where there is an entry for each data item. It is used in some TSO or optimistic methods where timestamps are stored with the data items, independent of transaction executions.

3.6 Overheads

The total amount of storage a concurrency control method utilises during its execution is determined, based on the sizes of the CC-DB relations used. The unit of storage overhead is taken as one field of one tuple of one CC-DB relation.

The analysis of CPU overhead is based on the total number of CC-DB accesses when executing the macros. The unit of CPU overhead is taken as one tuple access, insertion or replacement in one CC-DB relation.

The analysis of message overhead is based on the total number of messages required when executing the macros. The unit of message overhead is taken as one message sent to a single site.

4. Modelling distributed fully redundant two-phase locking

The use of 2PL as a distributed concurrency control method was discussed in detail by Bernstein & Goodman [1], Ceri & Pelagatti [3] and many others.

A simple scenario for 2PL is where locking is applied to all the copies of data items duplicated at all the sites. The locking primitives issued by the site of origin are sent to all the sites, where local locks are applied to the data item copies. Transactions discover their conflicts at all the sites where they request locks.

2PL as such is sufficient to provide serialisability, but it does not provide atomicity. If a transaction releases its exclusive locks during the shrinking phase, any transaction can observe its results prior to its commitment. To guarantee isolation, a transaction must satisfy a stronger form of 2PL where all exclusive locks are held until commitment.

In order to guarantee global consistency, 2PC must also be incorporated in the protocol to ensure that all the updates on the local data item copies are either all committed or aborted.

4.1 Concurrency control database

The following CC-DB fields and relations are used:

```
TSI(t-id, state)
ACC(obj-id, t-id, mode)
BLKD(t-id, cause-id)
```

Timestamps are not used; neither are transactions conditionally granted, thus the HIST relation is not used. Although it is a distributed method, the site relation is not used as the data objects are duplicated and locked at all sites. An ACC relation entry models the lock set on the particular data object.

4.2 Conflict situations

As locking takes place globally, block and restart conflicts will be detected locally at all the sites.

Blocking

A blocking conflict occurs if there is an access entry for a requested data object, with a conflicting access mode. It depicts the conflict where a transaction wishes to write or read a data object already locked by another transaction. This macro is in liberal pseudocode; the identifier of the conflicting transaction is also returned in the cause-id parameter; the calling macro must document the blocking conflict.

```
BLOCKCONFLICT(t-id, obj-id, mode,
    VAR cause-id): BOOLEAN
    cause-id := -
    cause-id := any ACC.t-id
        where (not (t-id = ACC.t-id)
            and obj-id = ACC.obj-id
            and (mode = "write"
            or (mode = "read"
            and ACC.mode = "write")))
    BLOCKCONFLICT :=
        not (cause-id = -)
```

Restarting

A restarting conflict occurs if the requested transaction is involved in a deadlock cycle. This exists if two or more transactions are waiting for each other in a circular fashion in such a manner that neither can proceed; depicted by entries in the blocked relation forming a cyclic path containing the transaction identifier.

```
RESTARTCONFLICT(t-id): BOOLEAN
    RESTARTCONFLICT := CYCLE(t-id)
        and (BLKD.cause-id = t-id
        or BLKD.t-id = t-id)
```

4.3 Transaction initiation

On transaction initiation, the root agent (RA) at the transaction's site of origin must broadcast the begin operation to all the remote sites.

```
On BEGIN(t-id):
    SEND("L-BEGIN(t-id)",
        all-other-sites)
    L-BEGIN(t-id)
```

A local begin records the active transaction at the site.

```
On L-BEGIN(t-id):
    append TSI(t-id, "ready")
```

4.4 Access requests

In this fully-redundant, overly strong 2PL implementation, a global read access request is broadcast to all the sites where it is executed locally. All the sites must apply a local lock to the data object before accessing it.

```
On READ-REQ(t-id, obj-id):
    SEND("L-REQ(t-id, obj-id, "read")",
        all-other-sites)
    L-REQ(t-id, obj-id, "read")
```

The macro for global write requests is similar to the read request macro. The only difference between modelling read and write operations is in the conflict situations; these appear in the blocking conflict macro. A local access operation is executed if the necessary lock can be granted; otherwise the transaction is blocked and even restarted if its blocking causes a deadlock cycle.

```
On L-REQ(t-id, obj-id, mode):
    If not BLOCKCONFLICT(t-id, obj-id,
        mode, -)
        and (TSI.t-id = t-id
        and TSI.state = "ready") Do
        {Set the lock and access the
            requested data item}
        append ACC(obj-id, t-id, mode)
            where not (ACC.obj-id =
            obj-id and ACC.t-id = t-id)
    Else BLOCKCONFLICT(t-id, obj-id,
        mode, cause-id)
        or (TSI.t-id = t-id and not
        (TSI.state = "ready")) Do
        {Block the transaction}
        BLOCK(t-id, cause-id)
        If RESTARTCONFLICT(t-id) Do
            {Unblock the transactions
            blocked by the restarted
            transaction}
            replace TSI(state := "ready")
            where (TSI.state = "blocked"
            and TSI.t-id = BLKD.t-id
            and BLKD.cause-id = t-id)
            {Restart the transaction
            causing the deadlock}
            RESTART(t-id)
        Endif
    Endif
```

4.5 Transaction commitment

Transaction commitment includes 2PC, with the RA as coordinator. In the preparation phase, all the sites are prompted by a message to determine their willingness to commit. The sites determine if the transaction is not blocked by a locked data item or if it has not been restarted to resolve a deadlock.

In the implementation phase the sites indicate their readiness to commit by returning ready or not ready messages. Only if all the sites answer positively, will the RA decide to commit, which is activated by broadcasting local commits to all the sites; otherwise an abort is broadcast to all the sites. The timeout mechanism is used to detect sites which are not ready to commit due to their inactivity.

The RA's site identifier is included in the preparation, commit and abort messages to indicate where the replies must be returned to.

```
On COMMIT(t-id):
    VAR msg
    {1st preparation phase}
    SEND("L-PREPARE(t-id, site)",
        all-other-sites)
    L-PREPARE(t-id, site)
    Activate timeout
    {2nd implementation phase}
    RECEIVE(msg, all-other-sites)
    If timeout {at least one site not ready}
        or any (msg = "not-ready") Do
        SEND("L-ABORT(t-id, site)",
            all-other-sites)
        L-ABORT(t-id, site)
        {Restart the transaction globally}
        RESTART(t-id)
    Else all (msg = "ready") Do
        SEND("L-COMMIT(t-id, site)",
            all-other-sites)
        L-COMMIT(t-id, site)
    Endif
    RECEIVE(acks)
    Return status
```

During preparation the sites control the state of the transaction and make sure that it is not still blocked by another transaction. The site's readiness or not to commit is returned to the RA.

```
On L-PREPARE(t-id, site-id):
    If  (any (BLKD.t-id = t-id)
        or (not TSI.state = "ready"
        and TSI.t-id = t-id)) Do
        {not willing to commit}
        SEND ("not ready", site-id)
    ELSE (not any (BLKD.t-id = t-id)
        and TSI.state = "ready"
        and TSI.t-id - t-id) Do
        {willing to commit}
        SEND ("ready", site-id)
    Endif
```

Local transaction commitment entails changing the transaction's state and making all the DB changes permanent. All the blocked transactions waiting for locks set by the transaction

must be reactivated when the locks are released. The sites indicate their completion by returning acknowledgements to the RA.

```
On L-Commit(t-id, site-id):
    replace TSI(state := "committed")
        where TSI.t-id = t-id
    {Unblock the transactions blocked by
        the committing transaction}
    replace TSI(state := "ready")
        where (TSI.state = "blocked"
        and TSI.t-id = BLKD.t-id
        and BLKD.cause-id = t-id)
    {Make all the DB changes permanent}
    {Release all the locks}
    EXPUNGE(t-id)
    SEND("ack", site-id)
```

During a local abort, apart from unblocking the transactions blocked by this transaction, the effects of the transaction must be nullified. After expunging the transaction, the RA is informed by an acknowledgement.

```
On L-ABORT(t-id, site-id):
    {Unblock the transactions blocked by
        the restarted transaction}
    replace TSI(state := "ready")
        where (TSI.state = "blocked"
        and TSI.t-id = BLKD.t-id
        and BLKD.cause-id = t-id)
    {Undo all the DB changes}
    RESTART(t-id)
    SEND("ack", site-id)
```

4.6 Overheads

The overheads due to storage space utilised, processing and communications done are quantified analytically.

Storage overhead

All the relations are replicated at every site. The TSI relation has 2 fields. It contains the states of all active and all re-initiated fatally conflicting transactions. It represents a cost of $2T_tS$ plus $2T_tF_fF_cS$.

The ACC relation has 3 fields. It documents the read locks of each transaction and each fatally conflicting transaction; numerous read locks can be set on every data item. It represents a cost of $3T_tR_rS$ plus $3T_tR_rF_fS$.

The ACC relation also documents all the write locks of each active and each restarted transaction; only one lock can be set on each data item. It represents a cost of $3T_tR_wS$ plus $3T_tR_wF_fS$ in the worst case (where the transactions have write requests to different data items) to $3S$ plus $3F_fS$ in the best case (where all the transactions have a write request to one data item).

The BLKD relation has 2 fields. It contains an entry for every blocked access request; a transaction can only be blocked at one access. Fatally conflicting transactions must also be blocked before deadlocks can be detected. It represents a cost of $2T_tF_cS$ plus $2T_tF_fF_cS$.

Thus, the total storage overhead is given by the expression $2T_tS + 2T_tF_cS + 4T_tF_fF_cS + 3T_tR_rS + 3T_tR_rF_fs + 3S + 3F_fS < STO_{D2PL} < 2T_tS + 2T_tF_cS + 4T_tF_fF_cS + 3T_tR_tS - 3T_tR_tF_fF_cS$.

CPU overhead

All the operations are executed at every site. A BEGIN operation involves a single access to document new and restarted transactions. It costs $1T_tS$ plus $1T_tF_fF_cS$.

Any non-blocking access request involves one access to determine that the data item is not locked, one access to determine that the transaction is active and one access to lock the data item. It costs $3T_tR_tS$, plus $3T_tR_tF_fF_cS$ for restarted transactions.

Every blocked access request, not involved in a deadlock cycle, involves one access to determine that the data item is locked, two accesses to change the transaction's state and note the deadlock detection information, then D accesses to determine it is not involved in a deadlock cycle. It costs $(3 + D)T_tR_tF_cS$.

Any fatally conflicting request involves the same number of accesses as in the previous case to detect that it is involved in a deadlock cycle and an additional $(2 + R_t)$ accesses to restart the transaction. It costs $(5 + R_t + D)T_tR_tF_fF_cS$. In addition it costs three accesses to unblock the blocked transactions, thus $3T_tF_fF_cS$ assuming one transaction becomes unblocked or $3T_tF_c(T_tF_f)S$ if all the transactions become unblocked.

A committing COMMIT operation involves two accesses during preparation to determine if the transaction is ready to commit; during commitment it involves one access to change the transaction's state, then an additional $(2 + R_t)$ accesses to remove the transaction. It costs $(5 + R_t)T_tF_cS$; plus the possible maximum of $3T_tF_cS$ for unblocking.

A globally restarting COMMIT operation involves two accesses during preparation to determine that the transaction has either already been aborted, or is still blocked; during the abort phase it involves one access to change the transaction's state and an additional $(2 + R_t)$ accesses to remove the transaction. Thus it costs $(5 + R_t)T_tF_fF_cS$, plus the possible maximum of $3T_tF_cF_fF_cS$ for unblocking.

Thus, the total CPU overhead is given by the expression $6T_t + 9T_tF_fF_cS + 4T_tR_tS + (3 + D)T_tR_tF_cS + (9 + R_t + D)T_tR_tF_fF_cS <$

$CPU_{D2PL} < 6T_aS + 3T_aF_cS + 6T_aF_{fc}S + 3T_aF_cF_{fc}S + 4T_aR_aS + (3+D)T_aR_aF_cS + (9 - R_a + D)T_aR_aF_{fc}S + 2T_a^2F_cF_{fc}S$.

Message overhead

A BEGIN operation involves one message sent to each other site for new and fatally conflicted transactions. It costs $1T_a(S-1)$ plus $1T_aF_{fc}(S-1)$.

Any access requests of transactions and restarted transactions involve one message sent to each other site. It costs $1T_aR_a(S-1)$ plus $1T_aR_aF_{fc}(S-1)$.

Any COMMIT operation involves one message sent to each other site for preparation, one answer back from each, one commit or abort message sent to each and then one acknowledgement back from each. It costs $4T_a(S-1)$, plus $4T_aF_{fc}(S-1)$ for transactions which were globally restarted.

Thus, the total message overhead is given by the expression $MES_{D2PL} = 5T_a(S-1) + 5T_aF_{fc}(S-1) + 1T_aR_a(S-1) + 1T_aR_aF_{fc}(S-1)$.

5. Virtues of the model

The prime advantage of the model is that it provides a framework for representing concurrency control methods. The differences between the methods are standarised by the uniform representations. This leads to meaningful quantitative overhead comparisons.

Another advantage of the model is that it is easy to use. Researchers in database directions use it comfortably.

The SITE relation representing the data dictionary allows a researcher to evaluate the effects of different data dictionary implementations.

An aspect not obvious from this paper, is that the storage overhead of a data object lock (three ACC fields) is represented as being less than a data object timestamp (four HIST fields). The storage of transaction timestamps also results in higher storage costs.

6. Shortcomings of the model

A problem of the model is that it is not always clear how to depict complex concurrency control operations using macros or how to to represent complex data structures using the CC-DB. Although usually representable in some *ad hoc* manner, it can contribute a greater source of overhead than a realistic representation.

During the analytical comparison of methods based on different mechanisms (eg locks, timestamps amd optimism), the conflict parameters are assumed relatable [2],[4]. The validity of this assumption has not been proved.

Apart from the fraction of recently committed transactions parameter, no attention has been given to the duration of storage overheads. In the modelling of conflicts, time duration is intuitively implied when re-executing conflicting transactions [4]. These aspects can be analysed in detail. Similarly, the sizes of messages have not been parameterised. In some optimistic methods the entire effect of a transaction is broadcast as a single message. Such messages incur greater message overheads, but is not accurately depicted by the model.

It is not clear how to rate methods when they differ with respect to different overheads. In the evaluations made, the different overheads have been considered weighing equal [4]. This assumption should be justified by analysing the relative contribution of each factor.

7. Conclusion

Although an actual comparison was not presented here, due to space limitations, it can be concluded that the abstract model of distributed database concurrency control methods can be used for meaningful analytical comparisons, despite its shortcomings.

The model can be used as a framework for future research. It should be modified to eliminate its discrepancies and it can be extended for the evaluation of aspects closely integrated with concurrency control, such as recovery or query decomposition methods.

It should be kept in mind that overheads are not the only criteria for comparisons. Related criteria include the degree of concurrency provided, to determine whether the amount of concurrency achieved relatively merrits the overheads incurred by a method.

References

[1] P.A. Bernstein & N. Goodman, [1981], Concurrency Control in Distributed Database Systems, *Computing Surveys*, **13**(2), 185-221.

[2] M.J. Carey, [1983], An Abstract Model of Database Concurrency Control Algorithms, *ACM SIGMOD Record*, **13**(4), 97-107.

[3] S. Ceri & G. Pelagatti, [1985], Distributed Databases - Principles and Systems, *McGraw-Hill*.

[4] M.H. Rennhackkamp, [1986], Evaluation of Concurrency Control Methods in Distributed Databases, *M.Sc thesis*, Department of Computer Science, University of Stellenbosch.

This paper was received in camera-ready format.

NOTES FOR CONTRIBUTORS

The purpose of the journal will be to publish original papers in any field of computing. Papers submitted may be research articles, review articles and exploratory articles of general interest to readers of the journal. The preferred languages of the journal will be the congress languages of IFIP although papers in other languages will not be precluded.

Manuscripts should be submitted in triplicate to:

Professor J M Bishop
Department of Computer Science
University of the Witwatersrand
Johannesburg
Wits
2050

Form of manuscript

Manuscripts should be in double-space typing on one side only of sheets of A4 size with wide margins.

The first page should include the article title (which should be brief), the author's name and affiliation and address. Each paper must be accompanied by an abstract less than 200 words which will be printed at the beginning of the paper, together with an appropriate key word list and a list of relevant Computing Review categories.

Manuscripts may be provided on disc using any Apple Macintosh package or in ASCII format.

For authors wishing to provide camera-ready copy, a page specification is freely available on request from the Editor.

Tables and figures

Tables and figures should not be included in the text, although tables and figures should be referred to in the printed text. Tables should be typed on separate sheets and should be numbered consecutively and titled.

Figures should also be supplied on separate sheets, and each should be clearly identified on the back in pencil and the author's name and figure number. Original line drawings (not photocopies) should be submitted and should include all the relevant details. Photographs used as illustrations should be avoided if possible. If this cannot be avoided, glossy bromide prints are required.

Symbols

Mathematical and other symbols may be either handwritten or typewritten. Greek letters and unusual symbols should be identified in the margin. Distinction should be made between capital and lower case letters; between the letter O and zero; between the letter I, the number one and prime; between K and kappa.

References

References should be listed at the end of the manuscript in alphabetic order of the author's name, and cited in the text in square brackets. Journal references should be arranged thus:

[1] E. Ashcroft and Z. Manna, [1972], The Translation of 'GOTO' Programs to 'WHILE' programs, *Proceedings of IFIP Congress 71*, North-Holland, Amsterdam, 250-255.

[2] C. Bohm and G. Jacopini, [1966], Flow Diagrams, Turing Machines and Languages with only Two Formation Rules, *Comm. ACM*, **9**, 366-371.

[3] S. Ginsburg, [1966], *Mathematical Theory of Context-free Languages* McGraw Hill, NewYork.

Proofs

Proofs will be sent to the author to ensure that the papers have been correctly typeset and *not* for the addition of new material or major amendment to the texts. Excessive alterations may be disallowed. Corrected proofs must be returned to the production manager within three days to minimise the risk of the author's contribution having to be held over to a later issue.

Only orginal papers will be accepted, and copyright in published papers will be vested in the publisher.

Letters

A section of "Letters to the Editor" (each limited to about 500 words) will provide a forum for discussion of recent problems