South African
Computer
Journal
Number 20
December 1997

Suid-Afrikaanse
Rekenaar-
tydskrif
Nommer 20
Desember 1997

Computer Science and Information Systems

Rekenaarwetenskap en Inligtingstelsels

The South African Computer Journal
An official publication of the Computer Society of South Africa and the South African Institute of Computer Scientists

Die Suid-Afrikaanse Rekenaartydskrif
'n Amptelike publikasie van die Rekenaarverenging van Suid-Afrika en die Suid-Afrikaanse Instituut vir Rekenaarwetenskaplikes

Editor
Professor Derrick G Kourie
Department of Computer Science
University of Pretoria
Hatfield 0083
dkourie@dos-lan.cs.up.ac.za

Subeditor: Information Systems
Prof Lucas Introna
Department of Informatics
University of Pretoria
Hatfield 0083
lintrona@econ.up.ac.za

Production Editor
Dr Riël Smit
Mosaic Software (Pty) Ltd
P.O.Box 23906
Claremont 7735
gds@mosaic.co.za

World-Wide Web: http://www.mosaic.co.za/sacj/

Subscriptions

	Annual	Single copy
Southern Africa:	R50,00	R25,00
Elsewhere:	$30,00	$15,00

An additional $15 per year is charged for airmail outside Southern Africa

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

Phone: +27 (11) 315-1319 Fax: +27 (11) 315-2276

Editorial Notes

For two reasons, this edition of SACJ is far later than it ought to have been. The first reason is that there have been some personnel changes in the editorial team. Lucas Introna, having continued for some time as IS editor after transferring to London, asked to be relieved of his duties. Niek du Plooy has kindly agreed to fulfill this role in a temporary capacity until a suitable replacement for Lucas can be found. Due to work pressure, Riël Smit has also withdrawn as production editor, and has been replaced by John Botha. SACJ owes the two retired members a huge debt of gratitude. During his period of tenure, Lucas did sterling work in setting and maintaining a solid standard for IS contributions. Riël put SACJ on a LaTeX path, and has laboured diligently to produce an aesthetically pleasing product. Thanks are also due to Niek and John for their willingness to take over in their respective roles. Until further notice, IS contributors may forward their submissions directly to Niek at his address given on the front inside cover. I shall put successful authors in touch with John for further instructions regarding final preparation of their manuscripts.

The second reason for a delay in this edition has to do with authors who have not scrupulously followed guidelines for producing their final submissions. There have been a variety of problems ranging from missing citations and inappropriate production of figures to incompatible electronic file submissions. All of this, coupled with our new production editor (who—despite an extremely busy schedule—has valiantly climbed a steep LaTeX learning curve) has resulted in an edition that should have been out to press several weeks earlier.

The editorial team will be giving attention to the general matter of format and submission procedures in future. SACJ's citation and reference methods are somewhat archaic and will probably be revised. All the necessary information will be provided on the new SACJ web site at `www.cs.up.ac.za/sacj/`. The site will also contain abstracts of articles in this and future editions.

These are times of conflicting stresses on both the academic and industrial IT communities. They are being felt somewhat more acutely in Southern Africa (and presumably in other developing countries) than in the developed world. Internationally there is tendency to cut back on state financing of universities and a seemingly insatiable demand for IT graduates. Many companies snap up new graduates at attractive salaries, positively discouraging full-time postgraduate studies. International recruitment agencies scour the South African scene for qualified candidates, luring some of our most promising young professionals out of the country. Job-hopping, a drift from academia to industry and from local industry to USA or European industry seems to be the order of the day. Despite the availability of private colleges and institutes, virtual or otherwise, there is a rush of students to university and technikon IT departments, all hoping to get at the IT honey-pot. University administrations are struggling to correct the structural deficiencies of the past and to provide IT departments with sufficient resources to cope with demand. As editor of SACJ, I have no particular competence authoritatively to sum up or analyze these tendencies, but it does seem to me desirable that someone ought to do so. Bodies such as SAICSIT, the CSSA, university authorities, IT industry and state representatives ought actively to pursue joint strategies to ensure that our IT departments are properly resourced and that (non-Zuma) measures are taken to retain graduates in the country. It seems almost redundant to attempt to spell out the consequences of inactivity.

Derrick Kourie
EDITOR

The Recovery Problem in Multidatabase Systems - Characteristics and Solutions

Karen Renaud and Paula Kotzé *
*Department of Computer Science and Information Systems, University of South Africa, P O Box 392, Pretoria, 0001.
E-mail: {renaukv,kotzep}@risc1.unisa.ac.za

Abstract

In many applications today multiple pre-existing database systems are integrated into a single multiple database system called a multidatabase system. One of the biggest problems for transaction management in a multidatabase system is the question of how to recover after failure and leave the multidatabase in a consistent state after the recovery process. In this paper we firstly outline the recovery problem and how the multidatabase situation makes the recovery process difficult. We then discuss various approaches to the recovery problem in multidatabase systems.

Keywords: *Multidatabase systems, transaction management, global commitment, database consistency, crash recovery.*
Computing Review Categories: *H.2.4,H.2.1*

1 Introduction

A multidatabase system (MDBS) is a system composed of autonomous or semi-autonomous pre-existing databases, together with a software layer which controls access to all data within the component databases from the point of view of the multidatabase system, while the component databases still function independently [18].

The software layer which controls all access to the MDBS is called the multidatabase management system (MDMS).

In an MDBS, transactions originating at the local database systems (LDBSs) are called local transactions. These co-exist with the agents of global transactions which originate at the multidatabase (MDB) software layer and which often span many local database systems. Global subtransactions are derivatives or subparts of the global transactions. The global transaction will be split up into a subtransaction for each LDBS which has to be accessed by the operations of the global transaction. The subtransactions will be sent to the LDBSs and when the results are obtained from the local systems, they will be processed in order to supply the global user with the required output.

MDB recovery seeks to maintain atomicity and durability of global transactions in the presence of failures.

Various failures can occur in an MDBS: transaction failures, site failures, media failures, network failures, database management system (DBMS) failures and system failures. We will only consider subtransaction and site failures since the other types of failures do not require unique handling in a MDBS [10]. In the case of subtransaction failures, the autonomy and heterogeneity of component database systems complicate matters.

The reason for this is that the MDMS will relinquish all control over a subtransaction once it is routed to a local system and will only know the result of the transaction once it has completed execution or been aborted. This becomes a real issue if any of the components of the MDBS fail since to ensure the atomicity of the global transaction, all subtransactions must complete. This paper focuses on the recovery issues involved in MDBSs.

2 Recovery issues for subtransaction failures

Any database transaction in a centralized database system is executed in two phases — first the operations (reads and writes) are done, and then the transaction either commits, in which case all changes to the database are made permanent, or it aborts (rolls back) in which case all database updates are lost. If a transaction commits, all changes become visible to other transactions in the local database system. If a transaction rolls back, it will appear as if the operations carried out by the transaction never happened.

In an MDBS, the MDMS has to find a way of ensuring that the entire global transaction commits or rolls back in the same way. Since global transactions span multiple database systems, they are broken up into subtransactions which are dispatched to the local database systems. In the interests of MDB consistency, it is vital that these subtransactions either *all* commit or *all* abort.

The types of problems caused by the local database systems being as autonomous as possible in an MDBS are:

1. The subtransactions which are sent to component database systems commit at all LDBSs but one, and that subtransaction aborts. There are various reasons why this could happen. One example is when a subtransaction becomes involved in a deadlock situation and is chosen by the local DBMS to be aborted in order to break the deadlock. This type of situation violates the atomicity of the global transaction.
2. Subtransactions are sent to the component database systems but the MDMS decides to abort the global

transaction. The MDMS succeeds in doing this in all but one local system, where the subtransaction has already committed and cannot be rolled back. This violates atomicity and also compromises the consistency of the MDBS since the local transactions which execute after the globally aborted committed subtransaction will see incorrect data values.

3. A local site fails and when it is restarted, the local DBMS runs a recovery procedure which rolls back all unfinished transactions — including global subtransactions of committed global transactions.
   Once again the consistency of the MDBS is compromised.
4. The MDMS site fails and when it recovers, has no idea which subtransactions have completed in the interim.

Subtransactions which have aborted and should have committed are dealt with by either retrying or redoing the subtransaction. Retrying involves submitting the entire transaction to the local database system again. The subtransaction would have to be redone by the MDMS which would entail the resending of only the write operations of the aborted transaction to the local database system. Subtransactions which have committed and should have aborted can be dealt with by compensation. To rectify the situation, the MDMS will send a compensating transaction to the local database system, which attempts to semantically reverse the effects of the transaction.

Site failures need a specific special crash recovery procedure which will be activated as soon as the site comes up again after a failure.

It may seem as if the problems could be solved if the MDMS were able to approve the commit action of subtransactions. In the first place one has to consider whether this is a realistic expectation. We contend that it is not. If we were to expect this from component database systems, it would violate their autonomy and be unrealistic since many commercial database products do not provide a prepare-to-commit state.

In the second case, one must determine whether a prepare-to-commit state would solve all our problems. Let us briefly consider the situation if all component database systems were to provide a prepare-to-commit state. The MDMS would seemingly have no trouble if a particular subtransaction were to fail. It could simply abort the other subtransactions when they report that they are ready to commit. However, Mullen *et al.* [13] have shown that even if all component database systems which provide a prepare-to-commit state use strict two-phase locking as their concurrency control method, atomic commitment is still impossible if even a single system failure occurs. They contend that in order to implement a successful two-phase commit, local autonomy must be violated.

We therefore have to find a mechanism for ensuring transaction atomicity with minumum autonomy violation while making provision for the possibility of failure. We will discuss various approaches to this problem in section 4 but will first address the related problem of crash recovery.

3 Crash recovery issues

If a local site crashes, the local DBMS will have a crash recovery procedure that will automatically start executing as soon as the site comes up again — before the database access by users is permitted again. The problems presented by an MDBS are that the local recovery procedure cannot tell the difference between locally uncommitted subtransactions and uncommitted local transactions. It is oblivious to the fact that some of the transactions belong to globally committed multidatabase transactions and will roll them back too — which could violate database consistency if other subtransactions of the global transaction of which the subtransaction forms a part, have already committed.

If these compensating transactions are run after local transactions have used the data items updated by the subtransaction which should not have committed, database consistency is compromised. We therefore have to find a mechanism to deal with this problem too.

When the MDMS site comes up again after failure, it will have to try to ascertain the status of all outstanding subtransactions of active global transactions. Due to the local systems' autonomy, we cannot assume that the local systems will allow a request of transaction status. If the MDMS has missed the notification that a subtransaction has committed or aborted, some way has to be found for the MDMS to be notified after coming up following a site failure.

4 Recovery approaches

4.1 The multidatabase system model

As stated before, a transaction can be split up into two distinct phases — the operations (reads and writes) and the commit or abort. A global transaction G is split up by the MDMS into a set of global subtransactions $GST_1, GST_2, ..., GST_n$. The decomposition process has been thoroughly researched and will not be addressed here. We will assume, for the sake of this discussion, that it is done correctly. There are, however, some provisos:

1. Only one global subtransaction per site.
2. Subtransactions have the same structure as the global transaction. They have read and write instructions as well as send and receive instructions where they communicate with the MDMS.
3. They also end with commit or rollback/abort.

The subtransactions are submitted to the various local database systems and run to completion.

4.2 The recovery model

The recovery process in a multidatabase system has four (possibly five) basic components:

1. The *multidatabase management system*: this has to coordinate the recovery of global transactions and subtransactions and maintain database consistency.
2. An optional *server process*: many multidatabase models use a server-type process which acts as an interface between the MDMS and the local system's DBMS.

3. The *global transactions* themselves: the reason for the entire process. How these transactions are constructed influences the recovery process.
4. The local system's *DBMS*: this component will handle the local recovery in case of a site failure.
5. The *local database* itself: where actual data is kept.

Various researchers have addressed the recovery problem. Some of the most promising work is that of Barker & Özsu, Pu, Breitbart *et al.*, Chen *et al.*, Kang & Keefe, Garcia-Molina *et al.*, Yoo & Kim, Georgakopoulos, Hwang *et al.*, Ye & Keane, Rajapakse & Orlowska and Pal & Lanka.

It is interesting to note that each of the recovery proposals will attempt to address the recovery problems in MDBSs by imposing restrictions upon, or violating the autonomy of, the latter three of the above components. We will discuss the research into these fields by looking at how each of these components is affected by the different proposed recovery protocols.

4.3 The global transactions

As we have stated before, transactions can be logically split up into two distinct phases, the phase where operations are carried out on the database and the phase where the transaction either commits or aborts. One approach, proposed by Garcia-Molina and Salem [8] involves the use of *sagas*. They have designed transactions called sagas which are ideal for long-lived transactions. Sagas are split up into subtransactions which can be interleaved with other transactions and each other in any order without compromising database consistency. The subtransactions of a saga should be executed as an atomic unit. To deal with a case of incomplete saga execution where a subtransaction has aborted at a particular site, Garcia-Molina and Salem require each transaction to have a compensating transaction associated with it which undoes any of the actions performed by the incomplete saga execution in order to return the database to a consistent state.

Another group of researchers, Chen *et al.* [6], require certain phases to be built into a multidatabase transaction. They require a transaction to be split up into three distinct phases — an execution phase which encompasses all operations on the database, an optional confirmation step and an optional undo step. These steps are determined from the semantics of a subtransaction. The transaction is then executed as two separate transactions. The undo step is simply a set of compensating operations to return the database to its original state. This allows relatively simple recovery in the case of a transaction that fails and does not violate DBMS autonomy to any great extent. The one problem with their approach is that the programmers are expected to set up the global transaction in three separate steps. This is also required by Garcia-Molina and Salem, who expect the programmer to set up a compensating transaction for each global transaction which can operate on the MDBS.

4.4 The local system's DBMS

The DBMS will, if left alone, deal with the global subtransaction the same way as it would deal with any of the local transactions submitted to it. That this causes problems has already been established. There are various ways a MDMS can restrict the execution of a local system's DBMS in order to maintain MDB consistency and to facilitate recovery. They can be broadly summarized as follows:

- Require a visible prepare-to-commit state.
- Modify the restart procedure after a local failure.
- Assume that the global subtransaction will not be aborted after operations have been completed and while the transaction is waiting to commit.
- Add a software layer above the local DBMS to handle submission of all transactions, both local and global, to the local DBMS.
- Expect certain local concurrency control mechanisms or restrict schedules (interleavings of transaction operations) produced by the local DBMS.

4.4.1 Visibility of the prepare-to-commit state

Pu [16] simply violates the autonomy of the DBMS by insisting that the DBMS get permission from the MDMS before committing a transaction. As Mullen *et al.* have shown, this does not make provision for the possibility of a site failure.

Ye and Keane [20] also expect the local DBMS to get permission from the MDMS before committing a transaction – a violation of control autonomy.

4.4.2 Modified restart

Barker and Özsu [1] expect the local DBMS to allow the MDMS exclusive access to the database after a site crash so that it can redo or undo the global subtransactions before the local transactions make use of the incorrect data items. Georgakopoulos [9, 10] also requires exclusive access after site failure.

4.4.3 No abortion at prepared state

Georgakopoulos [9, 10] expects the local DBMS not to abort a global transaction after it has completed all its operations. This is also a requirement of Ye and Keane. Georgakopoulos maintains that since the operations have all been carried out, there can be no reason for the local DBMS to abort the transaction. He argues that most DBMSs (eg. SYBASE and ORACLE) only have timeouts on outstanding operations and that a transaction will therefore not be aborted when it has reached the ready-to-commit state. A local site failure could cause a subtransaction to abort in this stage but that would be dealt with by the exclusive access after recovery assumption.

There are problems with this assumption. Pal and Lanka [15] have shown that we can only make this assumption when the local DBMS makes use of 2PL or strict timestamp ordering. It has been shown that object-oriented DBMSs can and do abort transactions at the prepared stage.

4.4.4 Adding a software layer

Yoo and Kim [21] put a stub process on top of the DBMS. This process will receive *all* transactions and route them to the DBMS. In the case of normal failure-free execution, the

local transactions and subtransactions will not be altered in any way and simply routed to the DBMS but in the case of failure, the stub can ensure that the local transactions are delayed until the global subtransactions are resubmitted and can therefore make sure that database consistency is not violated. This is a violation of local autonomy but will not affect any local applications. For this reason it might be more acceptable than some of the other schemes.

Pal and Lanka [15] synchronize local and global transactions outside the local DBMS using locks. A serial order is imposed on global transactions so that no global concurrency is allowed. A locking system is used to prevent local and global subtransactions from conflicting with one another.

Rajapakse and Orlowska [17] have extended Pal and Lanka's mechanism and allow greater concurrency by allowing global subtransactions to execute concurrently when they do not conflict. They maintain a control table of locks which will maintain database consistency and facilitate recovery by preventing local transactions from using data values while the lock is still active in the case of DBMS failure.

Soparkar *et al.* [19] violate autonomy to an even greater extent by expecting local transactions to be submitted to the GTM instead of to the local DBMS. This is also done by Muth and Rakow [14]. This is a severe autonomy violation and would probably be unacceptable to most application developers.

4.4.5 Restrictions on schedules

Some researchers impose restrictions on the schedules[1] produced by the local DBMS. Mullen *et al.* state that the only thing we can expect from the local DBMS is 2PL but often researchers impose more severe restrictions. Breitbart *et al.* expect rigorous schedules; Hwang *et al.* expect cascadeless 2PL while Georgakopoulos expects strictness and serializability. Elmagarmid *et al.* [7] expect the component database systems to have no value dependencies between them.

4.5 The local database

Some researchers have addressed the recovery problem by applying restrictions to how the data is used or by adding special data to the database. Breitbart *et al.* [3–5], Kang and Keefe [12] and Ye and Keane [20] all apply partitioning to the local database. The data is split up into globally updatable and locally updatable transactions. The local transactions are prevented from reading the globally updatable data items in Breitbart's scheme. Hwang, Srivastava and Li [11] have also proposed a recovery method which makes use of partitioning but is less restrictive than that imposed by Breitbart *et al.*. Hwang *et al.* allow local data to be read by global transactions but not to be written by them. This deals admirably with crash recovery since after a crash it is not necessary to restrict local transactions to prevent them from accessing data items, possibly with incorrect values due to the rollback of a global subtransaction which should have committed.

To deal with conflicting global subtransactions, Breitbart *et al.* and Ye & Keane have added a data item called a *ticket* to the database which each global subtransaction has to access before it is allowed to carry out any operations on the database. This forces an ordering on the global subtransactions which prevents them from conflicting with one another.

[1] Interleavings of active transaction operations

5 Summary

This paper illustrates the difficulties inherent in recovery in multidatabase systems. The types of failures which can occur in a multidatabase system have been discussed and the failures which need unique handling were identified. The various approaches to recovery proposed by researchers in the field were discussed. In our opinion, the option which seems to impact to the least extent on the autonomy of the participating database systems would be the addition of a software layer. It is unrealistic to expect certain standards to be adhered to in a member local database system — whether they apply to the global transactions, built-in procedures which form part of the local DBMS, or how the local DBMS schedules the execution of transactions submitted to it. No matter how attractive it may be to do this, or how much simpler it makes the handling of the MDBS, it is advisable not to interfere even to the smallest extent with the existing local database structures and mechanisms.

Crash recovery would seem to be a weak point in many multidatabase systems and research into better ways to effect recovery without violating local autonomy still remains to be done.

References

1. Barker K & Özsu M T. Reliable Transaction Execution in Multidatabase Systems. In: *Proceedings of the First International Workshop on Interoperability in Multidatabase Systems,* edited by Y Kambayashi, M Rusinkiewicz & A Sheth. Los Alamitos: IEEE Computer Society Press, 1991.
2. Barker K. Quantification of autonomy on Multidatabase systems. *Journal of Systems Integration,* **4**(2), 151 - 169, (1994).
3. Breitbart Y & Silberschatz A. Multidatabase Update Issues. *SIGMOD Record,* **17**(3), 135 - 142, (1988).
4. Breitbart Y, Silberschatz A & THOMPSON G R. Reliable Transaction Management in a Multidatabase System. *SIGMOD Record,* **19**(2), 215 - 224, (1990).
5. Breitbart Y. Multidatabase Interoperability. *ACM SIGMOD Record,* **19**(3), 53 - 60, (1990).
6. Chen J, Bukhres O A & Sharif-Askary J. A Customized Multidatabase Transaction Management Strategy. In: *4th International Conference, DEXA '93. Database and Expert Systems Applications,* edited by: Vladimír Mařik, Jiři Lažanský, Roland R Wagner. Prague, Czech Republic, 1993.
7. Du W & Elmagarmid A K. Quasi serializability: a correctness criterion for global concurrency control in InterBase.

In: *Proceedings of the Fifteenth International Conference on Very Large Databases,* edited by P M G Apers & G Wiederhold. Palo Alto: Morgan Kaufmann, 1989.

8. Garcia-Molina H & Salem K. Sagas. In: *Proceedings of ACM-SIGMOD. International Conference on Management of Data,* San Francisco, 1987.
9. Georgakopoulos D. 1990. *Transaction Management in Multidatabase Systems.* Dissertation, University of Houston.
10. Georgakopoulos D. Multidatabase Recovery and Recoverability. In: *Proceedings of the First International Workshop on Interoperability in Multidatabase Systems,* edited by: Y. Kambayashi, M. Rusinkiewicz & A. Sheth, 1991.
11. Hwang S, Srivastava J & LI J. Transaction Recovery in Federated Autonomous Databases. *Distributed and Parallel Databases,* **2**(2), (1994).
12. Kang I E & Keefe T F. Supporting reliable and atomic transaction management in multidatabase systems. In: *Proceedings of the 13th International Conference on Distributed Computing Systems.* Los Alamitos: IEEE Computer Society Press, 1993.
13. Mullen J G, Elmagarmid A K, KIM W & SHARIF-ASKARY J. On the Impossibility of Atomic Commitment in Multidatabase Systems. In: *Proceedings of The Second International Conference on Systems Integration.* edited by: P A Ng, C V Ramamoorthy, L C Seifert & R T Yeh. Los Alamitos: IEEE Computer Society Press, 1992.
14. Muth P & Rakow T C. Atomic Commitment for Integrated Database Systems. In: *Proceedings of the 7th International Conference on Data Engineering.* Los Alamitos: IEEE Computer Society Press, 1991.
15. Pal S & Lanka S. Transaction Processing in Multidatabase Systems Without Atomic Commitment Protocol. In: *ADC 94 Proceedings of the 5th Australian Database Conference,* 1994.
16. Pu C. Superdatabases for composition of heterogeneous databases. In: *Multidatabase Systems: An Advanced Solution for Global Information Sharing,* edited by A R Hurson, M W Bright & A Pakzad. Los Alamitos: IEEE Computer Society Press, 1988.
17. Rajapakse J & Orlowska M E. An extended transaction to maintain consistency and recovery in multidatabase systems. Information Sciences, **90**(1-4),19 - 38, (1996).
18. Renaud K V. *A Comparative Study of Transaction Management Services in Multidatabase Heterogeneous Systems.* Masters Thesis. University of South Africa, 1996.
19. Soparkar N R, Korth H F & Silberschatz A. Trading control autonomy for reliability in Multidatabase Transactions. *Technical Report TR-91-05.* The University of Texas at Austin, Computer Sciences Department, 1991.
20. Ye X & Keane J A. A Distributed Transaction Management Scheme for Multidatabase Systems. In: *Proceedings of the 1994 IEEE Region 10's Ninth Annual International Conference. Theme: Frontiers of Computer Technology,* Singapore, 1994.
21. Yoo H, Kim M H. A reliable Global Atomic Commitment Protocol for Distributed Multidatabase Systems. *Information Sciences,* **83**(1-2), 49 - 76, (1995).

Notes for Contributors

The prime purpose of the journal is to publish original research papers in the fields of Computer Science and Information Systems, as well as shorter technical research notes. However, non-refereed review and exploratory articles of interest to the journal's readers will be considered for publication under sections marked as Communications or Viewpoints. While English is the preferred language of the journal, papers in Afrikaans will also be accepted. Typed manuscripts for review should be submitted in triplicate to the editor.

Form of Manuscript

Manuscripts for *review* should be prepared according to the following guidelines.

- Use wide margins and $1\frac{1}{2}$ or double spacing.
- The first page should include:
  - title (as brief as possible);
  - author's initials and surname;
  - author's affiliation and address;
  - an abstract of less than 200 words;
  - an appropriate keyword list;
  - a list of relevant Computing Review Categories.
- Tables and figures should be numbered and titled.
- References should be listed at the end of the text in alphabetic order of the (first) author's surname, and should be cited in the text in square brackets [1–3]. References should take the form shown at the end of these notes.

Manuscripts accepted for publication should comply with the above guidelines (except for the spacing requirements), and may be provided in one of the following formats (listed in order of preference):

1. As (a) LaTeX file(s), either on a diskette, or via e-mail/ftp – a LaTeX style file is available from the production editor;
2. As an ASCII file accompanied by a hard-copy showing formatting intentions:
   - Tables and figures should be original line drawings/printouts, (not photocopies) on separate sheets of paper, clearly numbered on the back and ready for cutting and pasting. Figure titles should appear in the text where the figures are to be placed.
   - Mathematical and other symbols may be either handwritten or typed. Greek letters and unusual symbols should be identified in the margin, if they are not clear in the text.

   Contact the production editor for markup instructions.
3. In exceptional cases camera-ready format may be accepted – a detailed page specification is available from the production editor;

Authors of accepted papers will be required to sign a copyright transfer form.

Charges

Charges per final page will be levied on papers accepted for publication. They will be scaled to reflect typesetting, reproduction and other costs. Currently, the minimum rate is R30-00 per final page for LaTeX or camera-ready contributions that require no further attention. The maximum is R120-00 per page (charges include VAT).

These charges may be waived upon request of the author and at the discretion of the editor.

Proofs

Proofs of accepted papers in category 2 above may be sent to the author to ensure that typesetting is correct, and not for addition of new material or major amendments to the text. Corrected proofs should be returned to the production editor within three days.

Note that, in the case of camera-ready submissions, it is the author's responsibility to ensure that such submissions are error-free. Camera-ready submissions will only be accepted if they are in strict accordance with the detailed guidelines.

Letters and Communications

Letters to the editor are welcomed. They should be signed, and should be limited to less than about 500 words.

Announcements and communications of interest to the readership will be considered for publication in a separate section of the journal. Communications may also reflect minor research contributions. However, such communications will not be refereed and will not be deemed as fully-fledged publications for state subsidy purposes.

Book reviews

Contributions in this regard will be welcomed. Views and opinions expressed in such reviews should, however, be regarded as those of the reviewer alone.

Advertisement

Placement of advertisements at R1000-00 per full page per issue and R500-00 per half page per issue will be considered. These charges exclude specialized production costs which will be borne by the advertiser. Enquiries should be directed to the editor.

References

1. E Ashcroft and Z Manna. 'The translation of 'goto' programs to 'while' programs'. In *Proceedings of IFIP Congress 71*, pp. 250–255, Amsterdam, (1972). North-Holland.
2. C Bohm and G Jacopini. 'Flow diagrams, turing machines and languages with only two formation rules'. *Communications of the ACM*, 9:366–371, (1966).
3. S Ginsburg. *Mathematical theory of context free languages*. McGraw Hill, New York, 1966.

South African Computer Journal

Number 20, December 1997
ISSN 1015-7999

Suid-Afrikaanse Rekenaar-tydskrif

Nommer 20, Desember 1997
ISSN 1015-7999

Contents