Stability in a Network of MONADS-PC Computers

Frans A. Henskens, John Rosenberg, and Michael R. Hannaford

Department of Electrical Engineering and Computer Science
University of Newcastle
N.S.W. 2308
Australia

ABSTRACT

The MONADS-PC computer system implements an architecture supporting a very large persistent store based on a uniform virtual memory. We have previously shown how this virtual memory scheme can be extended to encompass a local area network of MONADS-PC computers. In this paper we examine the question of the integrity of the store in such a network. A modification to the MONADS architecture to implement stability is reviewed and extended to guarantee stability of a network-wide persistent store. The stability scheme allows for temporary interruption to the physical network without affecting the validity of exported pages owned by a node.

1. INTRODUCTION

The MONADS project at the University of Newcastle, Australia and the University of Bremen, West Germany, has designed and implemented an unconventional and novel computer architecture [8, 9, 10, 16] known as the MONADS-PC [14].

A feature of the architecture is its very large persistent store, supported by a paged virtual address space. The store is single-level in that it has no conventional file system; rather disk storage is seen as an extension of physical memory. The system state at any time is described by a combination of the contents of physical memory and the disk store. The system would be left in an inconsistent state if the physical memory contents were lost because of, for instance, a power failure or system crash.

File-based systems use utilities such as fsck [19] to restore file system integrity after an unexpected system shut down. The possible resultant loss of files, whilst potentially annoying to the user, is usually not critical to the system because such files are independent entities. The MONADS persistent store, on the other hand, contains all data including system management information and references between objects. Inconsistencies in such data are critical to system integrity and security, and can lead to problems such as incorrect volume page tables and dangling references.

Published in: *Proceedings of the International Workshop on Computer Architectures to support Security and Persistence of Information*, ed J. Rosenberg and J. L. Keedy, Springer-Verlag and British Computer Society, pp. 246-256, 1990.

A network of MONADS-PC computers has been implemented [6]. This network distributes the large persistent store across the networked machines. The system is based on a network paging protocol which guarantees full coherency of shared objects as well as both naming and location transparency. The distribution of the store across a network increases the potential area of impact of an unexpected shutdown of a node.

We have previously described a scheme based on shadow paging which will guarantee the integrity of the store in a single node following a failure [15]. The presence of multiple nodes with cross references between them adds complexity to the problem because of the possibility of partial failure of the network.

In this paper we examine the question of the integrity of the store in such a network. The modification to the MONADS architecture to implement stability [15] is reviewed and extended to guarantee stability of a network-wide persistent store. The stability scheme allows for temporary interruption to the physical network without affecting the validity of exported pages owned by a node.

2. DESCRIPTION OF THE MONADS NETWORK

In this section we describe the basic operation of a network of MONADS-PC computers. Although the physical implementation of the network is not of importance to this discussion, we do make assumptions. First, it is assumed that we are dealing with a network of homogeneous machines. Second, it is assumed that any two nodes that need to communicate can do so, and that the network communication mechanism is either reliable or fails entirely. Issues related to achieving this service, such as the handling of duplicate packets, will not be considered in this paper.

All data in the MONADS network exists in a single virtual address space which is partitioned in such a way that each node is allocated an individual range of addresses. These addresses are guaranteed to be unique across the network by including the owning node number in the high order bits of each address. The range of addresses owned by a node is further divided into volumes. Each volume corresponds to a physical disk, or partition of a disk, located at that node. Finally volumes are divided into regions called address spaces. The full structure of a virtual address is illustrated in figure 1.

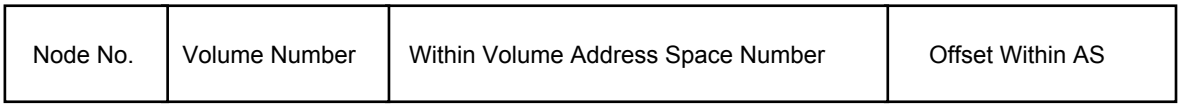

Figure 1: Full Structure of a Virt ual Address

Address spaces are used to hold a related set of data such as a program, an information hiding module, the equivalent of a file or a process stack. Address spaces are further divided into variable length segments, which are accessed by non-manufactureable and non-forgeable segment capabilities. For memory management purposes, the kernel sees each address space as a sequence of pages of fixed length, whilst the programmer or compiler sees each address space as a collection of its segments. Segments are mapped onto address spaces in such a way

that page and segment boundaries are decoupled [7], thus avoiding the internal fragmentation problems associated with most combined paging and segmentation schemes [12, 13].

Segments may themselves contain segment capabilities, so that it is possible to build arbitrarily complex data structures. However the MONADS architecture enforces certain rules to limit the propagation of segment capabilities. These rules ensure that, with the exception of those held in process stacks, segment capabilities always point to a segment within the same address space. Process stacks may contain segment capabilities pointing into any address space. The structure of the higher level architecture makes it possible to determine and control which address spaces may be referenced from any given process stack. The mechanisms used to implement these controls are described in [10].

A practical implementation of this addressing scheme requires that the node number, volume number and within-volume address space number fields are large enough to ensure that a supply of unique values will never be exhausted. Virtual addresses in the prototype MONADS-PC network are 60 bits long. The maximum size of an address space in the prototype is 256 megabytes made up of pages of size 4 kilobytes. A new implementation of the MONADS architecture, the MONADS-MM, which is currently being designed, will allow for 128 bit virtual addresses, with 32 bits for each field [17].

Each MONADS node has an Address Translation Unit (ATU) [1] which maps virtual addresses into physical memory addresses. The ATU is an inverted page table implemented in hardware, and its size is proportional to the size of the physical memory of the node. The design of the ATU is such that the size of the virtual address space does not greatly affect the speed of translation.

Each individual address space is wholly stored on a single volume, and a volume will typically contain many address spaces. A disk page table is maintained for each address space, and this is held in a protected region of the address space which it describes. In addition a volume directory is maintained for each volume indicating the location of the page table for each address space on that volume. A root page for each volume effectively contains a page table for the volume directory. Thus a volume may be viewed as a tree of pages, with the volume root page pointing to the volume directory, which points to the page table for each address space. In this paper we will refer to these data structures as the volume and address space red-tape.

When a virtual address is presented to the ATU for translation, it either returns the appropriate physical memory address, or generates a page-fault. In addition the ATU can support read-only pages and a write fault is generated if an attempt is made to modify such a page.

If a page-fault is generated, the node number field of the faulting address is examined to determine if the volume containing the required page is stored on the local node. If it is, the page fault is resolved by copying the faulting page into physical memory, and the waiting process is reactivated. If the page is stored on another node, the kernel transmits a message to the remote node requesting a copy of the page. The system supports mechanisms that allow for movement of objects between volumes, and of volumes between nodes. These are described in [3]. In this paper, for simplicity, we assume that each object resides at the node on which it was first created, that is the node whose number appears in the addresses of data

within the object. Moved objects and moved volumes may be stabilised using the techniques described in this paper in conjunction with the mechanisms described in [3].

Each node in the network maintains an Exported Pages Table (XPT) and an Imported Pages Table (IPT). These tables have the same structure as illustrated in figure 2. The XPT and the IPT maintain information on exported and imported pages respectively, and are used to implement the memory coherence algorithm described in [2, 6]. This algorithm implements a single writer/multiple reader protocol. Before an owner node brings a page off disk to resolve a page-fault, it must ensure that no read-write copy of the page exists at another node. This check is done by reference to the XPT. If a read-write copy does exist at another node, it must mark the page as read-only in its ATU and send a copy of the page back to the owner node. The page-fault at the owner node is, in this case, resolved using the copy of the page returned by the importing node rather than by disk access.

Pages are always exported with read-only access rights, and an importing node must request promotion to read-write access rights, if necessary, as a separate operation. It is allowable for multiple read-only copies of a page to exist in the physical memories of nodes in the network. If a read-write copy exists in the physical memory of a node, then it must be the only instance of the page in physical memory network wide. When an owner node receives a request for promotion of a page to read-write access, it must use the XPT to ensure that any other copies of the page are removed from the ATUs of all but the requesting node before granting the promotion request.

Node Number	Volume Number and Address Space Number	Within AS Page Number	Access Rights

One entry per exported page

Figure 2: Exported Pages Table

3. SINGLE NODE STABILITY

When a node is closed down in a controlled manner, all modified pages in physical memory are copied to secondary (disk) storage prior to completion of the shutdown. The disk locations of the contents of the pages of physical memory may be obtained from a data structure called the Main Memory Table (MMT) which is shown in figure 3. The ATU indicates whether the page has been modified. This controlled shutdown ensures that, on restart, all system management data and inter-object references are consistent.

In the case of unexpected shutdown caused by, for example, system software or hardware failure, copying of in-memory modified pages to disk cannot occur. So that the system has

consistent stored data for restart, it is necessary that, when running, it systematically moves the stored data between consistent states. Achieving this goal is not easy, because the usual paging mechanisms associated with virtual memory management necessarily result in the essentially random flushing of modified pages to disk to free up physical memory. The act of permanently storing a stable state to which it is possible to revert is called checkpointing, and a system that can always be restarted from a consistent state is said to be stable.

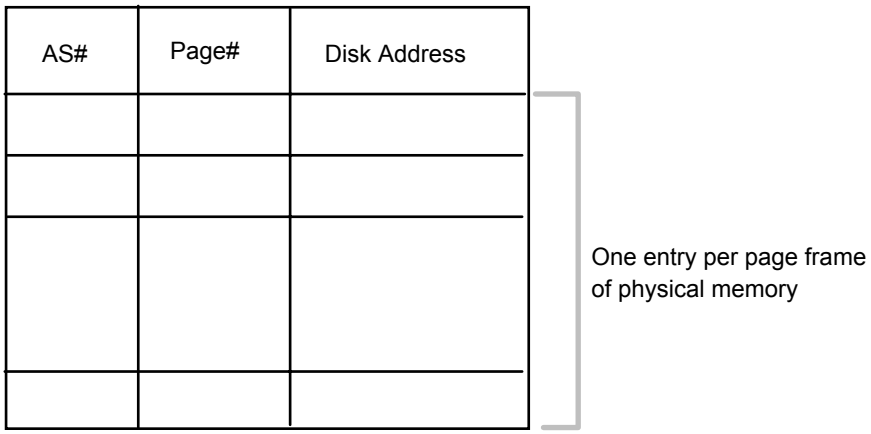

Figure 3: Main Memory Table (irrel evant fields omitted)

The stability scheme described in [15] is based on shadow paging as initially proposed by Lorie [11], and extended by others [4, 18, 20]. In this paper we initially consider this scheme for a single volume only. It can be extended to stabilise multiple volumes, which is required if there are cross references between volumes, by using a two phase commit as shown in [15]. This extension will be discussed later.

A Shadowed Pages Table (SPT) is maintained for each volume mounted on a node (see figure 4). The SPT is used to detect pages that have been shadowed since the last checkpoint to avoid multiple shadows being created, and to release disk space following a checkpoint operation. Only modified pages are shadowed, and at most two copies of a page exist on disk at any time, the latest version and the version at the last checkpoint. Disk space is dynamically allocated for pages, and at each checkpoint the disk space used for the previous checkpoint version of the page is returned to the free disk space pool. Following a checkpoint, the new stable version of pages forms the basis for further system operation.

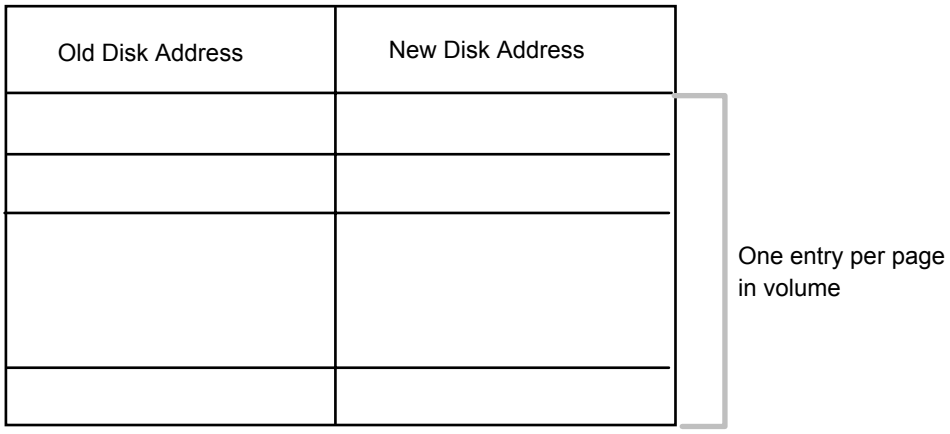

Figure 4: Shadowed Pages Table

Checkpoint operations are regularly performed as part of normal running of the system, but can be instigated by user software if required. The latter is essential to allow synchronisation with higher level transaction mechanisms.

The SPT can be efficiently implemented as two bit lists of disk pages on the corresponding volume, one each for old and new disk pages. When a write fault occurs on a page the current (or old) disk page is marked in the first bit list and becomes the shadow page. In addition, a new disk page is obtained from the free list and is marked in the second bit list. When the modified page is written back to disk, it is written to the new disk page. A read-write page, of course, may be written back to disk at an arbitrary time as part of virtual memory management, and may subsequently need to be paged back in. This scenario presents the possibility of multiple write-faults on the same page between checkpoints, so the new disk page bit list is checked every time a page is brought into memory, and if the disk page is found, indicating that it has previously been shadowed, the virtual memory page is mapped into the ATU as read-write, thus ensuring that at most one shadow page exists for any virtual memory page.

A checkpoint operation returns all pages in the old disk page list (ie the previous checkpoint versions of pages that have since been modified) to the volume free page list, and copies to disk, marking as read-only in the ATU, all modified pages in main memory, thus creating a new checkpoint version of the volume. Changing the access rights of read-write pages to read-only ensures that the next attempt to write to any page will cause a write-fault on that page, meaning that it will be entered in the SPT ready for the next checkpoint operation. Since the data structures describing the allocation of disk pages are themselves stored in virtual memory, the final stage of the checkpoint operation involves the writing to disk of the root page of the data structures. To enable recovery from a system failure during this last stage the root page is written as an atomic operation using Challis' algorithm [5]. There are two root pages for each volume, with the most recent correct page pointing to the last stable state. Correctness is ensured by insertion of timestamps at the beginning and end of the root pages as they are written.

4. NETWORK-WIDE STABILITY

Exported read-only pages pose no threat to the stability of the volume on which they are stored. Stability for exported read-write pages is not ensured by the single node stability scheme described above because such pages will not reside in the physical memory of the owner node at the time of initiation of a checkpoint operation.

The first stage in ensuring network-wide stability is the sending by the owner node of 'please return up-to-date copy' messages to remote nodes with read-write access to pages from the volume about to be checkpointed. As each page is returned it is marked as read-only in the remote node's ATU. When the page arrives at the owner node, it is stored in one of a pool of physical memory pages allocated by the kernel for receipt of incoming messages. Once the transfer of the virtual page into physical memory is complete, an entry must be made for the page in the MMT. Part of the MMT entry is the disk page to which the virtual page will be written on checkpoint, so a new disk page must be available for completion of the entry.

As described in [15], the lack of free disk space on a volume is one of the reasons for initiation of a checkpoint. It is, then, clearly unsatisfactory to assume that free disk space will be available for storage of pages returned by remote nodes. The solution is to allocate the new disk page to an exported read-write page prior to the granting of read-write access. At the same time an entry is added to the SPT. Finally, the new disk page address is entered in the address space page table as the storage location for the virtual page, guaranteeing that space exists on the disk to save the virtual page during the next checkpoint.

The single node stability scheme [15] resolves page-faults by bringing pages off disk into memory with read-write access if an entry for the page appears in the SPT. This is not appropriate if the node is networked and a copy of the page exists in the physical memory of another node, because the page coherence algorithm does not allow a read-write page to appear in the physical memory of multiple nodes simultaneously. As described in section 2, the XPT is checked when page-faults occur as part of the page coherence algorithm. If an exported read-write copy of the page exists, the page will be returned by the importer, thus making disk access unnecessary. If an exported read-only copy exists, the page is brought into memory off disk. In both of these cases the page is mapped into the ATU of the owner node as read-only. If no entry for the page exists in the XPT, the page is not in the memory of any node in the network, and so it can be safely brought in off disk and mapped into the ATU of the owner node with read-only or read-write access depending on the SPT.

The page coherence algorithm [2, 6] does not allow a read-write page to appear in the physical memory of multiple nodes simultaneously, and thus no MMT entry can exist at the owner node for an exported read-write page. When an exported page is returned to its owner node, the page must be entered into the MMT, which involves knowledge of the corresponding disk page number. The disk page number could be obtained from the address space page table. However such access could cause page faults [15]. We have adopted an alternative approach, which avoids this possibility. This involves storing the new disk page information in the XPT, which is held in locked-down memory in the kernel of the system.

5. RECOVERY FROM NODE FAILURE

The above system will allow correct operation of checkpointing across the network with little change to the single node scheme, provided that modified versions of pages are always available at the time of a checkpoint operation. Unfortunately this may not be the case because of either the failure of a node or of the interconnecting media. There are two separate cases of a failing node. These are the failure of the importer of a page and the failure of the exporter of a page. We examine each of these cases separately.

5.1. Failure of an Importing Node

As described in section 4, the volume checkpoint operation results in the owner node requesting the return of all exported read-write pages. If an importing node fails, it obviously cannot respond to such a request. It is clear that any modifications to pages stored in the physical memory of a node are lost if the node fails, so examination of importing node failure reduces to analysis of the behaviour of the exporting node. In our analysis we presume, for simplicity, that a single page has been imported by the failed node. If multiple pages are affected, the described single-page scheme may be applied to each page individually.

After several attempts to retrieve the page the exporting node concludes that the importing node has failed, and that modifications to the exported page are lost. It is tempting to simply remove the SPT entry for the page, appropriately modify the volume and address-space red-tape information, and then continue the checkpoint operation as if the lost page had never been exported. This proposition is unsatisfactory if the following sequence of events relating to virtual pages 'X' and 'Y' occurs after the most recent checkpoint operation.

1. Pages X and Y are transferred to a remote node with read-write access.

2. A segment is created in page Y, and pointed to by a segment capability in page X.

3. As part of the normal page discard process on the importing node, page X is returned to the owner node.

4 The importing node fails, and subsequent to that the owner node attempts to stabilise the volume containing pages X and Y.

If the above sequence of events were to occur, and the stabilise proceeded without the inclusion of the modified version of page Y, then page X would continue to contain a reference to the now non-existent segment that had been created in page Y. The problem is potentially extremely serious from a security point of view because a new segment may subsequently be created in the same location in page Y as that pointed to by the segment capability in page X. This would allow access to a segment in violation of the capability protection scheme that is designed to protect it.

In order to maintain full consistency and security it is essential that only a consistent set of pages be saved at the time of a checkpoint. If this is not possible because a node containing a page from the volume does not respond, then the volume cannot be stabilised at that time. The four available options are to:

1 stabilise without including the unavailable page

2 not stabilise at this time

3 delay the stabilise until the non responding node returns the necessary page, or

4 revert to the previous checkpoint state.

Option 1 can only be allowed if it can be determined that excluding the missing page would not result in a dangling reference, i.e. the missing page contains no reachable segments. Such a determination can only be made by computing the transitive closure for the volume being stabilised. In general, this would be prohibitively expensive, and may not even be possible if the page containing the root of addressing is itself missing.

Option 2 may not always be possible because the stabilise may be essential before system operation can proceed. This may be necessary if the stabilise operation was instigated because of a lack of disk space, or that the stabilise was required as part of the implementation of a higher level transaction mechanism.

Option 3 must be subject to time-out since operation of the owner node on the volume cannot proceed until the stabilise operation has completed, and the remote node may be unavailable for an extended period.

We propose that option 3 be implemented with a parameterised time-out period. If the time-out is reached, then the system concludes that the stabilise cannot take place and the volume reverts to its last checkpoint state. Reverting to the last checkpoint state involves instructing all contactable nodes with pages from the volume to invalidate these pages. The importing nodes can be easily identified by reference to the XPT. In addition, pages from the volume in the memory of the owner node must also be invalidated, these pages being identified via the MMT.

The reversion to the last checkpoint state may not be as serious as it at first seems. It is expected that there will be a higher level transaction mechanism for controlling concurrency and serialisability. Such a mechanism could well provide a transaction log to allow a roll forward from the reverted state, thus recovering at least some of the lost modifications to the volume.

5.2. Failure of an Exporting Node

If an exporting node fails, it will restart at its last stable state, with any local modifications performed since that checkpoint being lost. It is interesting to consider the fate of a read-write page, 'X', exported by the failed node since the last checkpoint. Attempts by the importing node to page out X are unsuccessful, and X is effectively trapped in the physical memory of the importing node. Such a situation would be detrimental to performance at the importing node because its available physical memory would be reduced by the page size for each trapped page, this situation being potentially critical for a diskless node.

A solution is to use up/down protocol messages to periodically monitor the status of the owner of imported pages. If the owner of page X is found to be non-responding, page X is

immediately marked as read-only, thus preventing further modifications, and an attempt is made to return a copy of the page to its owner. If the attempted return fails, the page is invalidated, thus losing any modifications to the page in exactly the same way that post-checkpoint modifications at the owner node are lost following a crash. In a similar fashion, on the detected failure of an exporting node, all read-only pages from that node are invalidated.

There is a further problem with this scheme. If an exporting node suffers a system failure, it will subsequently be restarted at its last checkpoint state. It is possible that other nodes have pages from volumes on the restarted node in their memory. Regardless of whether these pages are read-only or read-write, they may not be consistent with the checkpoint state to which the owner node has reverted. It is therefore imperative that any pages previously exported by the restarted node be invalidated at remote nodes. As described in the previous paragraph this will happen automatically if an importing node detects the exporting node's failure. However a short-term crash may not be detected.

When a node restarts, it broadcasts a message informing all other nodes that it is back on-line and that all pages previously imported from it are to be marked as invalid. Once any such pages have been invalidated, nodes return a message informing their existence, and the node names are stored by the restarted node. Any request for provision of a page to a remote node is refused if that node does not appear in this table and the requesting node is again instructed to invalidate all pages imported from the node. A successful acknowledgement to this instruction results in the requesting node being inserted into the table. We can thus guarantee that all inconsistent versions of pages will eventually be invalidated. As part of the up/down protocol, any newly on-line node is added to the table.

5.3. Failure of the Interconnecting Media

Failure of the interconnecting media appears the same as the failure of an exporting or importing node in that pages cannot be transferred between the nodes. The timeouts described in sections 5.1 and 5.2 will handle the situation of a media interruption of duration less than the time-out period. A longer term media problem is equivalent to a node failure, and is handled using the techniques described above.

There must be flexibility in the setting of the time-out periods to allow for different physical network implementations and for fluctuations in network traffic. It would also be sensible to provide a breakout facility to force a restart before the end of a time-out.

6. MULTIPLE VOLUME STABILISE ACROSS NODES

As was described in section 2, it is possible to have cross references between volumes. These are always held in the form of segment capabilities stored in stack address spaces. In order to ensure consistency it is essential that volumes containing cross references are stabilised together. In a network, these volumes can exist on different nodes. As part of the control of between-object references, a dependency graph must be maintained at each node to describe volume inter-relationships. Various protocols for constructing and maintaining the dependency graph are being investigated. The problem is considerably simplified in the MONADS architecture because of the clear distinction between within-address-space references and references to other address spaces, potentially on other volumes.

When a volume is stabilised, the dependency graph is consulted, and if cross references exist, what is essentially a two phase commit is used to cause all the related volumes to be stabilised together. The stabilise may involve volumes stored on more than one node. Appropriate protocols for implementing the two phase commit across the network are still under development.

7. CONCLUSION

Stability can be achieved for a single node persistent store by the use of shadow paging. The large virtual memory store can be extended to encompass a network of nodes, with the physical location of objects within the network being totally transparent to the user.

By extending the mechanisms used to achieve single node stability, we can ensure network-wide stability. Such stability is achieved with a minimum of network traffic overhead, and uses existing memory coherency protocol messages. In achieving stability, the service offered to the remote user is equivalent to that offered to local users, in that, at worst, modifications made since the last checkpoint for a volume on a failed node are lost.

The proposed scheme guarantees the security of data within the system. Via the use of time-out periods, temporary interruptions to the physical network media can occur without loss of data. Where cross references exist between volumes, the volumes may be stabilised together utilising a two phase commit protocol over the network. When combined with an appropriate higher level transaction mechanism, it should be possible to roll forward to recover most modifications made to a volume between the last checkpoint on the volume and a system failure.

ACKNOWLEDGEMENTS

The MONADS project is supported by grants from the Australian Research Council and the University of Newcastle Senate Research Committee. Thanks are also due to Mr. P. Brössler, Dr. A. Brown, Prof. R. Morrison and Mr. D. Munro for their contributions and suggestions.

REFERENCES

1. Abramson, D. A. "Hardware Management of a Large Virtual Memory", *Proc. 4th Australian Computer Science Conference*, Brisbane, pp. 1-13, 1981.

2. Abramson, D. A. and Keedy, J. L. "Implementing a Large Virtual Memory in a Distributed Computing System", *Proc. 18th Hawaii Conference on System Sciences*, pp. 515-522, 1985.

3. Brössler, P., Henskens, F. A., Keedy, J. L. and Rosenberg, J. "Addressing Objects in a Very Large Distributed System", *Proc. IFIP Conference on Distributed Systems*, Amsterdam, pp. 105-116, 1987.

4. Brown, A. L. "Persistent Object Stores", Universities of St. Andrews and Glasgow, Report 71, 1989.

5. Challis, M. F. "Database Consistency and Integrity in a Multi-user Environment", in *Databases: Improving Usability and Responsiveness*, ed B. Scheiderman, Academic Press, pp. 245-270, 1978.

6. Henskens, F. A., Rosenberg, J. and Keedy, J. L. "A Capability-based Fully Transparent Network", University of Newcastle, N.S.W. 2308, Australia, Report 89/7, 1989.

7. Keedy, J. L. "Paging and Small Segments: A Memory Management Model", *Proc. IFIP-80, 8th World Computer Congress*, Melbourne, Australia, pp. 337-342, 1980.

8. Keedy, J. L. "A Memory Architecture for Object-Oriented Systems", in *Objekt-orientierte Software und Hardwarearchitekturen*, ed H. Stoyan and H. Wedekind, Teubner-Verlag, Stuttgard, pp. 238-250, 1983.

9. Keedy, J. L. "An Implementation of Capabilities without a Central Mapping Table", *Proc. 17th Hawaii International Conference on System Sciences*, pp. 180-185, 1984.

10. Keedy, J. L. and Rosenberg, J. "Support for Objects in the MONADS Architecture", *Proceedings of the International Workshop on Persistent Object Systems*, Newcastle, Australia, pp. 202-213, 1989.

11. Lorie, R. A. "Physical Integrity in a Large Segmented Database", in *ACM Transactions on Database Systems*, 2,1, pp. 91-104, 1977.

12. Organick, E. I. "The Multics System: An Examination of its Structure", MIT Press, Cambridge, Mass., 1972.

13. Randell, B. "A Note on Storage Fragmentation and Program Segmentation", in *Communications of the ACM*, 12, 7, pp. 365-369, 1969.

14. Rosenberg, J. and Abramson, D. A. "MONADS-PC: A Capability Based Workstation to Support Software Engineering", *Proc, 18th Hawaii International Conference on System Sciences*, pp. 515-522, 1985.

15. Rosenberg, J., Henskens, F. A., Brown, F., Morrison, R. and Munro, D. "Stability in a Persistent Store Based on a Large Virtual Memory", *Proceedings of the International Workshop on Architectural Support for Security and Persistence of Information*, Bremen, West Germany, 1990.

16. Rosenberg, J. and Keedy, J. L. "Object Management and Addressing in the MONADS Architecture", *Proceedings of the International Workshop on Persistent Object Systems*, Appin, Scotland, 1987.

17. Rosenberg, J., Koch, D. M. and Keedy, J. L. "A Massive Memory Supercomputer", *Proc. 22nd Hawaii International Conference on System Sciences*, vol 1, pp. 338-345, 1989.

18. Ross, D. M. "Virtual Files: A Framework for Experimental Design", University of Edinburgh, Report CST-26-83, 1983.

19. SUN Microsystems Inc. "Systems and Networks Administration", Report 800-1733-10, Revision A, 1988.

20. Thatte, S. M. "Persistent Memory", *Proc. IEEE Workshop on Object-Oriented DBMS*, pp. 148-159, 1986.

Author List:

Mr. Frans A. Henskens
Department of Electrical Engineering & Computer Science
University of Newcastle
N.S.W. 2308
Australia

Email: henskens@nucs.nu.oz

Assoc. Prof. John Rosenberg
Department of Electrical Engineering & Computer Science
University of Newcastle
N.S.W. 2308
Australia

Email: johnr@nucs.nu.oz

Dr. Michael R. Hannaford
Department of Electrical Engineering & Computer Science
University of Newcastle
N.S.W. 2308
Australia

Email: mrh@nucs.nu.oz