An Algorithm for Stabilising Multiple Stores

[1]Ewa Z. Bem and [2]John Rosenberg

[1]School of Computing and Information Technology
University of Western Sydney Nepean
PO Box 10, Kingswood, NSW 2747, Australia
ewa@cit.nepean.uws.edu.au

[2]Faculty of Information Technology
Monash University
johnr@infotech.monash.edu.au

Abstract. The algorithm for stabilising multiple stores, which we present in this paper, was developed in the process of designing the global stability and resilience mechanism for Grasshopper, an operating system explicitly designed for experimentation in persistence. Grasshopper's persistent store is divided into multiple logical partitions (local stores) with arbitrary data interdependencies. A global asynchronous checkpoint mechanism is used to ensure the resilience of the store as a whole. In order to eliminate the known deficiencies of such an approach our algorithm takes advantage of the hardware techniques originally developed for fault tolerant systems, ie. mirrored disks and an uninterruptible power source (UPS). We show that these two techniques complement each other resulting in a simple and efficient algorithm where the main cost is the cost of additional hardware. Although developed in the context of the Grasshopper system, the algorithm can be applied to multiple persistent stores in general.
Keywords: multiple stores, persistence, global checkpoint, stability, fault tolerance

1 Introduction

The commonly applied mechanism to ensure stability and resilience in multiple stores is an asynchronous “fuzzy” global checkpoint with shadow paging or logging used for stability. Variations of this mechanism were proposed for L3/Eumel [6], Napier store [9], the MONADS systems [10], the KeyKOS nanokernel [4], and others. This approach addresses the main problem of the synchronous “stop the world” global checkpoint, as it does not require that user activity stops while the data modified since the last checkpoint is copied to stable storage. With asynchronous checkpoint this data, commonly represented in terms of pages, is copied lazily in the period of time between two checkpoints. Unfortunately this mechanism still suffers from major performance penalties:

- the additional disk I/O operations needed to stabilise the modified pages compete with the I/O operations requested by users,
- typically the logging and shadow paging techniques contribute to the loss of data locality.

The algorithm for stability and resilience for multiple persistent stores, presented in this paper, is an attempt to remedy these problems by combining the techniques originally developed to provide fault tolerance ie. mirrored disks and an uninterruptible power source (UPS). Both of these techniques are used in the IBM AS/400 system [11], but they are applied there in a significantly different manner than the one we propose.

Our algorithm was developed in the context of Grasshopper [1,3,7], an experimental orthogonally persistent operating system operating system, which supports persistence of both data and computations as the only permanent data storage. Since there is no file system, all data in the Grasshopper system has to be stable ie. survive any breakdown of service, and resilient ie. retain its consistency despite system malfunctions. Grasshopper’s persistent store is composed of multiple local stores (partitions), each managed independently by an entity called a manager, responsible for providing stability to data in its partition. Arbitrary dependencies may develop amongst partitions, so in order to ensure coherence and resilience of the whole store, another entity, called a controller, is responsible for synchronising the activities of the managers.

This paper is organised as follows. Section 2 describes the structure of the global multiple store, section 3 presents the algorithm we developed, section 4 demonstrates how this algorithm can be applied in a mixed environment with each local store stabilised in a different manner, and section 5 presents our conclusions.

2 Structure of the Global Store

As stated before the global store consists of multiple independently managed local stores. Each store has a manager responsible for all the aspects of data in the store, including data stability. The resilience of the global store is the responsibility of a controller. The activities of the controller and the managers can be grouped according to the phases in the life cycle of the store shown in fig. 1.

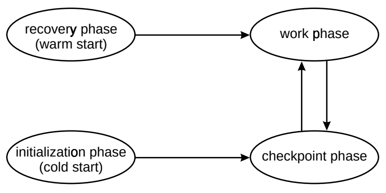

Fig. 1. Life cycle of the persistent store

Both the controller and the managers require some additional storage in memory and on disk to maintain their respective control data. The controller maintains two versions of a global root page of the store in a known area of the disk and an active copy of the current global root page in memory. Each disk version of global root page defines one stable state of the store. The store effectively moves from one stable state to another and each such state is represented by a state interval. Each global root page contains two copies of the corresponding state interval number, links to local root pages of each of the partitions, and information required to restart the managers. The two interval numbers in all root pages (global and local), physically located at the start and at the end of a page, are used to ensure the consistency of the disk block according to Challis' algorithm [2].

The in-memory active global root page contains a number of flags to indicate the current state of each partition. These flags are used by the controller to synchronise the global checkpoint events. The controller is not directly responsible for any user data in the local store; this is the responsibility of managers.

Each manager maintains two sets of control data on disk corresponding to the two most recent checkpoints (stable states) of the local store. Each set contains a partition local root page, a free block bitmap and a disk page table. The manager also maintains active control data in memory derived from disk copies. On disk the disk page table contains elements called disk page table entries (DPTE), each of which contains the virtual address of the corresponding page and its location on disk (disk address), the active DPTE in memory has an additional field for the physical memory address of resident pages, and flags indicating the state of the page. Logically the disk page table is a linear lookup table, which maps page virtual addresses to disk addresses. Since this table is sparsely populated, it is implemented as a hash table to avoid an unnecessary waste of storage.

2.1 Activities of the controller

The initialisation phase performed by the controller spans two state intervals to allow for the orderly creation of global control data, all partitions with their managers, and local control data within partitions. Once the store is initialised the controller commences the work phase by enabling user activity. The time between two checkpoints, determined by the controller, may be fixed, or it may be adjusted dynamically based on various aspects of system behaviour, for example the number of pages modified during the current interval, availability of physical pages in memory etc. During the checkpoint phase the controller directs the activities of the managers to ensure that the global order of events is as follows:

- all managers stabilise their partitions, excluding the root pages
- all managers stabilise their corresponding local root pages
- the controller writes the global root page to stable storage
- the controller increments the state interval number

In the recovery phase the store is rebuilt from a copy on disk. It logically follows the checkpoint phase, although in fact there could have been an intervening work phase which was interrupted by a system crash. The disk potentially contains two complete stable states of the store, one corresponding to the last checkpoint and the other to the preceding checkpoint. Consequently there may be two global root pages available, and the controller has to decide which one of them is most recent, based on the state interval number recorded in each page. If the crash occurred while the more recent root page was being written to disk, it will not have two matching interval numbers. In such case the other root page is selected.

2.2 Activities of the manager

The role of the manager is to respond to requests from the controller, and to collect and maintain enough information to be able to comply with these requests. During the work phase (normal operation) the manager responds to all page-related events in its partition. These events include new page creation, read fault, write fault, page discard etc.

In the checkpoint phase the manager stabilises all the pages modified since the last checkpoint, the new bitmap and the new disk page tables. For each page the appropriate flags in its memory DPTE are set to ensure the correct handling of the page in the next state interval. The in-memory allocate bitmap is overwritten with the new bitmap. On request from the controller the manager stabilises the active local root page, creating a new stable state of its partition.

In the recovery phase the manager selects the local root page which matches the state interval number selected by the controller, and recreates the in-memory control data. It is then ready for normal operation, which will commence as soon as the controller enables user activity.

3 The Proposed Algorithm

Combining the mirrored disks and UPS techniques allowed us to build a simple and efficient algorithm for providing stability in a local store. We applied the fault tolerance technique of mirrored disks as the basis for stability of pages to eliminate the deficiencies of commonly used software-based mechanisms. In itself disk mirroring at the hardware level could not contribute to stability, since it would overwrite both copies of a page at the same time. The way two identical disks are used to provide stability is to delay writing of one disk until the state of the other one is consistent. To make this distinction clear we use the term disk pair. The disks in a pair are designated as the primary and the secondary disk. When the system is first started both disks are initialised identically. The primary disk corresponds to the current state interval, and the secondary disk to the immediately preceding state interval.

With the addition of UPS the main memory can be treated as a stable media and an extension of the hard disk, so pages resident in memory are stable and can be shadowed temporarily by copying to another memory location. Pages are only copied to disk as a result of the page discard, and during the system shutdown. No disk I/O is performed for stabilisation of the pages in the store.

A page in the store can be in one of the following states:

- DO (disk original) - the page resides on both disks, no copy exists in memory (initial state)
- MO (memory original) - the page resides on both disks, and in memory (read only page)
- MD (memory dirty) - the page was modified in the current state interval (read/write page)
- MC (memory clean copy-on-write) - the page was modified in the previous state interval, and will be copied in memory before it is modified again
- ML (memory locked) - the page is a copy of an MC page modified in the current state interval, it is locked for both read and write.

The state transitions of pages are best illustrated by viewing the pages in memory as members of two logical lists: an active list associated with the primary disk, and a passive list associated with the secondary disk. A page can either be a member of both lists (DO, MO and MC pages), the active list only (MD pages), or the passive list only (ML pages). An MD page can further modified since it already has a shadow either on disk or in memory. All pages in the passive list are write protected since any attempt to write has to be captured to ensure an appropriate state transition for the page:

- MC page – a copy in memory is created, the original page changes state to MD page, and the copy to ML,
- MO page – changes state to MD,
- ML page – does not belong to the current address space, and any attempt at writing is illegal.

To perform the store stabilisation during checkpoint, MD pages change state to MC, and ML pages are marked free in the active in-memory bitmap. The ML pages do not belong to the checkpointed state of the store any more. The page state transition diagram which excludes the discard of MD pages is shown in figure 2. The page replacement algorithm is adjusted to the requirements of checkpoint, and the pages are discarded in the following order:

- ML page – written to secondary disk only
- MO page – resident on both disks, no disk I/O involved
- MC page – written to both disks

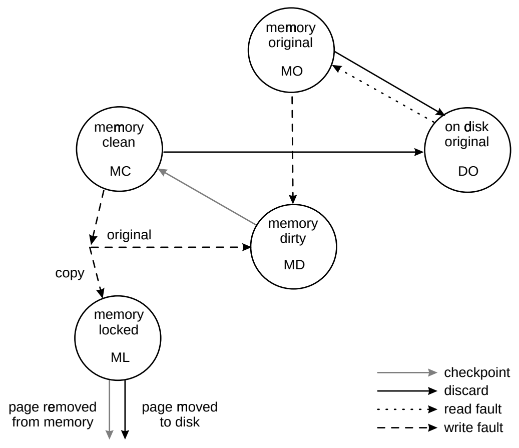

Fig. 2. Page state transitions without MD discard.

The possible discard of MD pages (figure 3) creates a number of problems. As an MD page belongs to the current state interval (and the active list) it should be written to the primary disk only. During the next checkpoint this page will be transferred to the MC state, and as such becomes the member of both lists, so a copy should exist on both disks. A discarded MD page would have to be brought back from disk to memory, so that it can be copied to the secondary disk.

This is in conflict with the original design intention to eliminate checkpoint related disk I/O activity. In addition if MD page discard is allowed, it is necessary to keep track of the movement of pages to disks in the pair, and if necessary make copies to the secondary disk on stabilise.

A preferred approach is to disallow discard of MD pages. Instead, in case of memory shortage, after all ML, MC and MO pages were removed from memory, the controller should trigger a new checkpoint. This action transfers all MD pages to the MC state, and makes them available for discard. Also the ML pages are removed, which makes more memory available for active pages. Frequent checkpoints triggered by memory shortage will affect the performance of the system, just like page thrashing in virtual memory systems. The only real remedy in such case is adding more main memory.

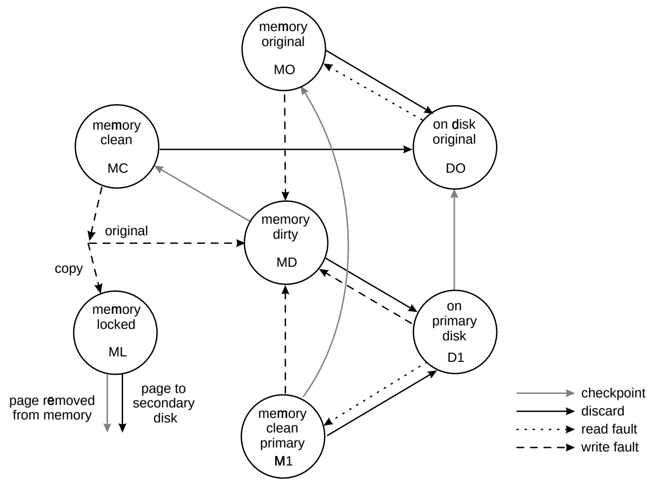

Fig. 3. Page state transitions with full discard

With UPS the recovery phase is inherently complicated because two levels of physical storage are involved in the process. When the system goes down it is not known how long it will remain off line. As a UPS cannot supply power to the system indefinitely, the available time has to be used to transfer the MD, MC and ML pages from the main memory to disk.

In the case of an orderly system shutdown, a disk checkpoint is performed which copies all MC and MD pages to disk. The ML pages may be excluded as we can guarantee the successful completion of the checkpoint. A system crash may not allow for the orderly sequence of actions described above. In such a case a dedicated disk is used to dump the contents of main memory with no attempt at selecting any specific pages. In order to be able to restore the contents of the productive disks when the system is eventually restarted, the physical location in memory of the global root pages has to be known. Also to maintain the principle of persistence by reachability, the physical memory location of partition local root pages has to be available in the global root page. Once this information is available the production disks can be updated using the information stored in the memory dump. With the disks updated and reflecting the current consistent state of the store, the restart can proceed in the usual way.

4 A Mixed Environment

In a multiple store it is reasonable to expect that each local store should be able to chose the stabilisation mechanism which best fits its data characteristics. The proposed algorithm can be used in some of the local stores, while the others are using conventional shadow paging or logging.

The store in this example consists of three partitions each using a different stabilisation mechanism:

- Partition A - the after image shadow paging [8]
- Partition B - the UPS with disk pairs
- Partition C - the KeyKOS style log [5]

The resilience of the whole store is ensured by the asynchronous global checkpoint, as shown in figure 4. In step 1 the controller requests all three managers to prepare for the stabilise mode by protecting all pages modified in their respective partitions in the current state interval. In partition A this means that all the pages modified in the current state interval are write protected to be copied to disk in the next step. In partition B all the MD pages are moved to MC state, and ML pages are dropped. In partition C the modified pages in memory are logically copied to the working area, with the actual copy deferred, using a copy-on-write mechanism. The roles of the two swap areas are reversed; the working area becomes the checkpoint area and vice versa.

When the managers complete the above activities the store enters the proper stabilise stage. In partition A all the pages marked for copying are progressively written to disk. For partition B no special actions are necessary. Any page in MC state, if written, is copied in memory, just as during the normal operation. Partition C undergoes the migration phase, ie. all the pages in the checkpoint area are written to their home locations on disk. This partition is potentially a weak point of the store, as it uses the least efficient stabilisation mechanism.

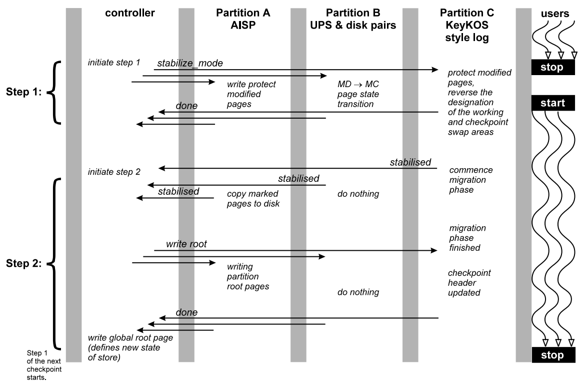
controller
Partition A AISP
Partition B UPS & disk pairs
Partition C KeyKOS style log
users
Step 1:
initiate step 1
stabilize_mode
stop
write protect modified pages
MD → MC page state transition
protect modified pages, reverse the designation of the working and checkpoint swap areas
start
done
stabilised
initiate step 2
stabilised
commence migration phase
stabilised
copy marked pages to disk
do nothing
migration phase finished
write root
Step 2:
writing partition root pages
checkpoint header updated
do nothing
done
write global root page (defines new state of store)
Step 1 of the next checkpoint starts.
stop

Fig. 4. Global checkpoint in the mixed environment

Eventually the controller requests the managers to stabilise their respective local root pages. It is expected that all data pages in the partitions are stable by this time. In partition A the root page is simply written to disk. In partition B the root page is already stable in memory. If required its ML copy may be written to the secondary disk at this stage. For partition C the action corresponding to stabilising the root page is sending a copy of the new checkpoint header to the controller. This completes step 2 of the checkpoint phase, and step 1 of the next checkpoint may now commence.

Obviously the efficiency of the global checkpoint in a mixed environment is limited by the slowest store, in this case the KeyKOS style log-based partition. It is important to select the stabilisation techniques for the local stores in such a way that the largest and most active stores use the most efficient mechanisms. Once this selection is made any change of the stabilisation mechanism for a specific store may require a considerable conversion effort.

5 Conclusions

We have shown that the proposed stabilisation algorithm eliminates all checkpoint related disk I/O activity and maintains the original clustering of pages in the store, which are the two main sources of poor performance for global asynchronous checkpoint systems. The loss of performance is prevented at the cost of increased demand for main memory and disk space, and the addition of UPS hardware. The discard mechanism, if properly integrated with checkpoint, helps to reduce the demand for memory by reducing the number of MO, MC and ML pages.

Despite the demonstrated ability to support a variety of stabilisation algorithms in the proposed model, we believe that the optimal solution for any multiple store is to use the most efficient available algorithm for all the partitions in the store. The proposed checkpoint algorithm achieves the stability and resilience of the store at a very low cost in terms of performance. It does not affect the error-free operation, and guarantees that in the case of an unexpected crash or failure of power supply the loss of productive work is very small. We have shown that this is only possible if the additional hardware is used to supplement and support software mechanisms. With the prices of main memory and magnetic disks falling continuously, the additional cost of storage is affordable. The prices of UPS devices vary depending on their capacity and battery run-time, but on average the cost is of the same magnitude as other computer system components.

The software overhead imposed by the algorithm is much smaller than in other known systems. It is limited to maintenance of root pages, and disk page tables for the partitions, and the actual processing is limited to shadowing the pages in memory, and a single pass over the active DPT once for each checkpointing cycle.

References

1. Bem, E.Z., “Global Stability and Resilience in a Persistent Operating System”, PhD Thesis, University of Sydney, 1999
2. Challis, M.F. “Database Consistency and Integrity in a Multiuser Environment” Databases: Improving Useability and resposiveness, Academic press, pp. 245-270, 1978
3. Dearle, A., et al “Grasshopper: An Orthogonally Persistent Operating System”, Computer Systems, Vol 7(3), Summer 1994
4. Hardy, N. “The KeyKOS Architecture” Operating System Review, 1985
5. Landau, C.R., “The Checkpoint Mechanism in KeyKOS” Proceedings of the 2nd International Workshop on Object Orientation in Operating Systems, IEEE, 1992
6. Liedtke, J., “A Persistent System in Real Use – Experiences of the First 13 Years” Proceedings of the 3rd International Workshop on Object-Orientation in Operating Systems, North Carolina, 1993
7. Lindström, A., A. Dearle, R. di Bona, S. Norris, J. Rosenberg, F. Vaughan “Persistence in the Grasshopper Kernel” Proceedings of the Eighteenth Australasian Computer Science Conference, ACSC-18, ed. Ramamohanarao Kotagiri, Glenelg, South Australia, February 1995
8. Lorie, R.A. “Physical Integrity in a Large Segmented Database” ACM Transactions on Database Systems, vol.2 no 1, pp.91-104, 1976
9. Morrison, R., A.L. Brown, R.C.H. Conor, A. Dearle “The Napier88 Reference Manual” Technical report PPRR-77-89 University of St Andrew, St Andrew,1989
10. Rosenberg, J. “The MONADS Architecture: A Layered View” The Fourth International Workshop on Persistent Object Systems, eds A. Dearle, G.M. Shaw and S.B. Zdonik, Morgan Kaufmann, 1990
11. Soltis, F.G. “Inside AS/400” Duke Press, Loveland, Colorado,1995