A Virtual Memory Translation Mechanism to Support Checkpoint and Rollback Recovery

Nicholas S. Bowen*
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, N.Y. 10598

Dhiraj K. Pradhan
University of Massachusetts at Amherst
Department of Electrical & Computer Engineering
Amherst, MA. 01003

Abstract

Checkpoint and rollback recovery is a technique that allows a system to tolerate a failure by periodically saving the entire state and if an error occurs, rolling back to the prior checkpoint. This technique is particularly suited to applications with long execution times such as those typically found in supercomputer environments. This paper presents a technique that embeds the support for checkpoint and rollback recovery directly into the virtual memory translation hardware. The scheme is general enough to be implemented on various scopes of data such as a portion of an address space, a single address space or multiple address spaces. A basic model is developed which measures the amount of work required by the scheme as a function of the checkpoint interval size. Using this model the degree to which the overhead decreases as the interval size increases is shown.

1 Introduction

Checkpoint and rollback recovery is a technique that allows a system to tolerate a failure by periodically saving the entire state and if an error occurs, rolling back to the prior checkpoint. This has long been used in database systems. The basic idea is that there are two copies of each piece of data, or at least a way to create two copies of the data. One, called the active set, contains the copy that can be updated. The other, called the checkpoint set, contains a complete and consistent copy of the active set at some prior time. The active set is where the state advances past the prior checkpoint. A checkpoint is performed by saving the active state. A rollback is accomplished by purging the active state and then replacing the active state with the checkpoint state. As mentioned, both copies need not always exist but at least must be recreatable. For example, to tolerate to program failure one could save only the active set, plus a record of all changes made to the original data. Thus, from the active data, one could recreate the checkpoint data by undoing all changes made to the active data.

*Work performed while on leave at the University of Massachusetts on Ph.D. studies. E-mail: bowenn@watson.ibm.com

One cost of a restart is the re-execution of the work in progress at the time of a failure. Supercomputer applications generally have the characteristic of significantly long execution times (e.g., measurable in minutes, hours). Although processor speeds continue to increase, it is unlikely the long execution times will ever abate. This is due to two aspects of their complexity; namely the scope of the problem and the approximation techniques in the solution. We illustrate this with an aerodynamic calculation from[12]. In regards to the problem scope, there can be an increase of 10^2 in computational requirements between flows about the air foil and the complete aircraft. For the technique used in the solution, there is a change of 10^5 in computational requirements based on the technique (e.g., simulation, approximation). The net result is that as processor speeds increase, there will be an expansion in the problem scope and refinements in the solution technique to absorb the new speed. Another issue is the class of architectures supporting supercomputer applications. For example, two of IBM's general purpose systems, the RS/6000 workstation and the 3090 mainframe, have been shown as a feasible platform for supercomputer applications[8, 14]. We believe the ideas presented in this paper have a wide variety of applicability in general purpose computing and are particularly suited to long running supercomputer applications.

Shadow paging is a technique used to support checkpoint and rollback recovery[11]. This technique has two complete mapping vectors which are only used on disk references (i.e., misses to the main storage database). Read-only requests share pointers to the original data while modify-requests copy the data from the original location but allocate new disk slots for subsequent page outs. The data is committed by forcing all changed data to disk and writing a status record to a special file which indicates the active and checkpoint versions of the mapping vectors. While the shadow paging uses two complete mapping vectors, there are several schemes that use a single fixed mapping to disk slots where each disk slot contains two copies of the page. These schemes are only invoked on the disk accesses and each disk slot has a header field which contains additional information to determine the correct version of the page. Reuter first proposed this "twin page" storage system to support checkpoint and rollback recovery[13]. Each page has a time stamp and transaction identifier which are used to identify the most recent version of the page. Furthermore, pages modified by a single transaction are linked together to allow selected backout of a single transaction. A full backout requires a scan of all disk pages to find the active pages. Thatte first extended the twin page scheme to be applied in a persistent virtual memory environment[18]. He removed the requirement of making the disk pages adjacent and provided an algorithm to convert the dual pages to single versions for read-only data. Wu and Fuchs optimized the rollback process of the twin page based system for use in both a database system[20] and a recoverable distributed virtual memory system[21] by allowing deferred backouts.

Another option is to periodically take a snapshot of the entire virtual address space. Li, et. al. proposed a real time concurrent snapshot scheme which uses the commonly found page write-protection bits[10]. This allows the checkpoint to proceed concurrently with the snapshot by protecting those pages not yet saved with the page-protect bits. Logging is a database technique of saving the changed data on a sequential file (i.e., a log) to enable atomic updates and rollbacks of changes. The IBM 801 storage architecture uses lock bits (to prevent a page-out to the protected disk copy) and a log (to save enough information to rollback) to support atomic updates[6].

There are several checkpoint and rollback schemes that have been proposed as transparent cache based systems to handle processor transient faults [1, 4, 7, 22]. The active and checkpoint locations are spread between different levels in the physical memory hierarchy (e.g., cache and main memory). The use of pseudo-protection bits are used to keep the active and checkpoint data separated.

Several hardware-based solutions support rollback recovery by providing dual copies of data. A recovery cache has been proposed for the PDP-11[9] which is a special device that manages rollback data by monitoring the address and data bus. Proposals have been made for memory modules to understand the transaction concept from databases by having each logical module implemented with a dual set of memory banks[2, 3]. All writes go to the first module until which time the end of transaction is signaled and the memory controller internally copies the data to the second bank. Staknis proposed a write-many, read-one memory organization in which writes can be directed to multiple physical locations under the control of a mask[17]. This allows checkpoints to be captured by dropping a page from the target of future writes.

A new technique is presented here to use the virtual memory translation mechanism to support checkpoint and rollback recovery that is active on every memory reference. Normal virtual memory supports a two level store (i.e., main memory and disk). This proposal suggests a dual mapping where each individual mapping still supports the normal two levels. For every virtual page there now exists a pair of mappings, one for the active state and one for the checkpointed state. Thus, in addition to the dual disk mapping like the shadow paging there are also dual real memory mappings. The scheme allows the checkpoint processing to be deferred to the next use of the data. This is similar to the timestamp concept of the twin page systems[13, 18, 20, 21], page protection bits in[10] as well as the unwritable-unchangeable concepts in the transparent cache based schemes[1, 7, 22]. Only one "time stamp" (v) is required per logical page mapping along with a single bit (l) which orders the two pages. The checkpoint is performed by simply incrementing the global checkpoint counter V. On every memory reference, V must be checked against the local copy v, and if appropriate, additional processing is performed. The checkpoint processing can be very fast and typically requires only a manipulation of the page tables or a memory to memory copy of a page. This is possible because both the active and checkpoint versions are embedded into the virtual memory domain (i.e., real memory and disks) and multiple copies of the data are often available (i.e., 2 disk slots and potentially 2 main memory pages). It allows the operating system to perform normal virtual memory management without interfering with the checkpoint scheme. The scheme could also be implemented directly in the translation hardware with very little performance penalty.

The paper is organized as follows. In Section 2, a detailed description of the architecture is presented. Section 3 describes the operation in terms of all state transitions. Section 4 discusses several important issues in the implementation and proposes several optimizations. Section 5 presents a performance analysis of the scheme. Finally, Section 6 discusses future work.

2 Virtual Checkpoint Architecture

This section describes the details of the virtual translation mechanism that supports checkpoint and rollback recovery. Three primary topics in this section are a detailed description of the translation fields (Section 2.1), the fault assumptions (Section 2.2), and an overview of the operation (Section 2.3).

2.1 Detailed Architecture

There are several additional fields which must be added to the translation mechanism. A normal virtual memory translation mechanism is assumed to take the virtual page number as the input and output a real frame number (if it exists) and a disk slot number. There is also a change bit that indicates if the data in the real frame has been modified. The new page translation mechanism has information to distinguish among checkpoint and active versions of the page. To avoid cumbersome notations, we shall assume that our focus is the translation of a single virtual page. This means the notation shall exclude any reference to the virtual page number. For each page, the mappings are replicated and are referred to as m_l and $m_{\bar{l}}$. The mapping m_l contains mappings for the real frame (r_l) and the disk copy (d_l). The alternate mapping, $m_{\bar{l}}$, has similar fields. Each page translation mechanism has a one bit field, l, which can be thought of as a switch which points to the most recently used mapping. Thus, m_l is always the mapping that was used last. In addition, each page translation mechanism has a k-bit field (v) which contains a copy of the checkpoint number (V) during the previous reference. The checkpoint number, V, is a global value which is incremented on every checkpoint. The scope of the checkpoint number can take on various ranges. For example, it could be the entire system, several address spaces, a single address space, or a portion of an address space. Each real frame, designated by r_l, contains a change bit (c_l) which indicates if the data at the real frame has been changed and is different than the disk copy at d_l.

Figure 1 shows the fields required for the translation of a single page. Generally, the page map table (for virtual to real) and external page map table (for virtual to disk) are physically separated. Although they are shown together, this is not a requirement of the scheme. Furthermore, even though each page translation contains a pair of mappings, m_0 and m_1, they can be independently operated upon by traditional page managers in regards to real storage management by the operating system (e.g., page stealing).

The scheme not only has the dual disk mappings like[11] but extends this to the real memory mapping.

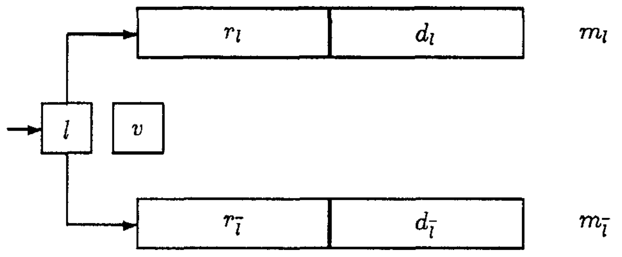

Figure 1: Overview of single page mapping

However, instead of having disjoint mappings for the active and checkpoint, they are now scattered between the two mappings based on the data references since the prior checkpoint. The correct mapping is identified when referenced using a technique conceptually similar to the twin page schemes [13, 18, 20, 21]. However, instead of only checking the status on disk fetches, our scheme determines the correct version on every memory reference. Furthermore, these other schemes require two timestamps, one per slot (the location of the time stamps are the disk header, though Thatte suggests the disk map table as an alternative[18]). The scheme proposed in this paper uses a single time stamp plus a one bit switch which are both located in the page map table used for virtual to real translation. Note that the deferred back outs of [20, 21] would not work well unless an additional associative memory were added to contain the identifiers of the backed-out transactions.

2.2 Fault Assumptions

A very important issue with a design for a highly reliable system is the fault model. The ultimate environment for this proposal is that of a system with non-volatile memory. The recovery in this environment would be to protect against faults such as processor transient faults[4, 7, 22] or complete system failures provided the main memory has back-up power[4]. Under these assumptions, the checkpoint is taken by simply incrementing the global checkpoint counter V.

However, this architecture is not dependent on non-volatile memory. If the memory is considered volatile, then in addition to incrementing V, modified portions of the page table plus modified pages must be forced to disk. Although this creates additional overhead, the scheme would work identically for non-modified pages.

An intermediate scheme would be a system with both volatile and non-volatile memory. The page table and the modified pages could be stored in the non-volatile memory. Examples of this are the Sequoia system where writable memory is duplexed with battery backups[4] or the addition of fault tolerant memory modules into the processors address space[2, 3].

2.3 Operational Overview

This section provides a high level overview of the operation of the system. A detailed description is found in Section 3. As noted, on every reference the values V and v must be compared. When a page is referenced there are two cases which must be distinguished. Figure 2 shows the first case where the page has been referenced at least once since the prior checkpoint and m_l has an active $< r_l, d_l >$ mapping. This is detected by the global checkpoint number V matching the local number v for the individual page. In this case the translation is allowed to proceed without interruption. The mapping m_l provides access to the correct data.

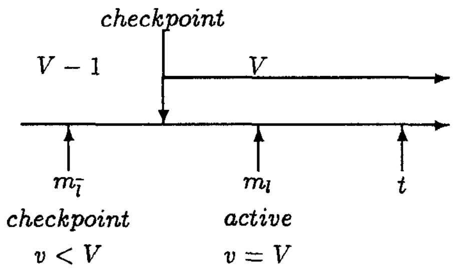

Figure 2: Case 1- page previously referenced

Figure 3 shows the second case where the page has not yet been referenced since the prior checkpoint. This situation occurs because the processing of the checkpoint is deferred until the next time of use. The mapping m_l refers to the previously active mapping which must become the checkpoint version. The mapping $m_{\bar{l}}$ previously contained a checkpoint version and now contains data of no value. This situation is detected by unequal values of the global checkpoint value, V, and the local number v. At the point of translation m_l shall be called the "old-active" (or new-checkpoint) and $m_{\bar{l}}$ shall be called the "old-checkpoint" (or new-active). It does not matter if multiple checkpoints occurred since the last reference to the page. The basic idea is that the data from m_l is used while the meaning of m_l and $m_{\bar{l}}$ are switched. This allows $m_{\bar{l}}$ to be the active while m_l becomes the checkpoint. However, the data from m_l must be copied to $m_{\bar{l}}$. The checkpoint number V is copied to the local value v and the l bit is inverted. This puts the page map into a state such that subsequent references proceed as in the Case 1 example.

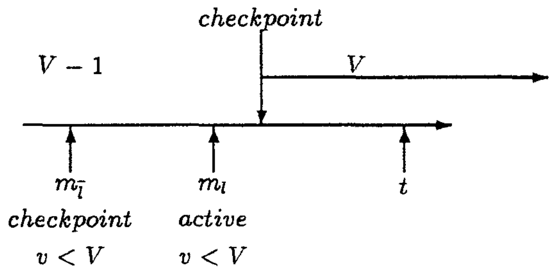

Figure 3: Case 2- first reference after checkpoint

A rollback is functionally performed by discarding any data that has been modified since the prior checkpoint. If the page has not yet been referenced since the prior checkpoint then the page is essentially in a rolled back state and nothing needs to be done (e.g., Case 2 in Figure 3). If the page has been referenced since the prior checkpoint then there is an active page that must be discarded. So for all pages with $V = v$, the v value is decremented and the l bit is inverted. This forces the status to be like Figure 3 where m_l ($m_{\bar{l}}$ before l was inverted) contains the checkpoint and $m_{\bar{l}}$ (m_l before l was inverted) contains useless information.

3 Virtual Checkpoint Operation

This section provides a detailed description of the operation of the scheme. There are six states defined for each page mapping which are based on the primary and secondary real frame mappings (i.e., r_l and $r_{\bar{l}}$) and the change bit for the primary frame (i.e., c_l). These are summarized in Table 1. The notation f_i indicates that some physical page is mapped while $\emptyset$ indicates that no physical frame is mapped.

State	r_l	$r_{\bar{l}}$	c_l
IA	f_i	f_j	0
IB	f_i	f_j	1
IIA	f_i	$\emptyset$	0
IIB	f_i	$\emptyset$	1
III	$\emptyset$	f_j	X
IV	$\emptyset$	$\emptyset$	X

Table 1: Summary of page states

Figure 4 shows all possible state transitions that can occur for each page mapping. There are five classes of events that can cause these transitions; namely initial page references, first references after a checkpoint, program stores into non-modified pages, normal program page faults, and operating system page management functions.

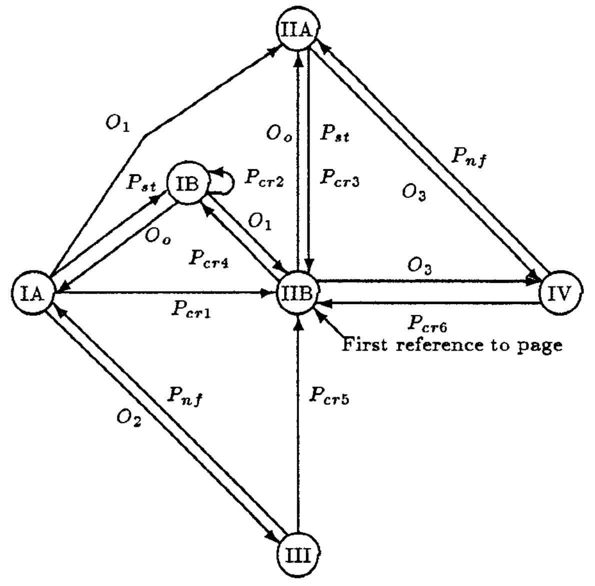

Figure 4: State diagram for page mappings

3.1 First Reference After Checkpoint

This section describes all state transitions shown in Figure 4 that are caused by a program making the first reference to a page after a checkpoint has occurred. By definition $V \neq v$, which causes the checkpoint processing to be performed. At the completion of the checkpoint process, the field v is set to that of V and the l bit is inverted. A table accompanies each transition. Each table shows the before and after image of the mapping. In the "before" image, m_l is the active mapping. In the "after" image, $m_{\bar{l}}$ is the active mapping. The notation for the real map consists of a triplet where the first element is the frame number, the second element represents the data in the frame, and the third element is the change bit. The notation for the disk map consists of a slot number and a symbol representing the contents of the page. The symbol $\emptyset$ indicates a null pointer and ψ is a "don't care" situation. Table 8 summarizes the actions performed at each state transition caused by the first reference after a checkpoint.

State IA

A first reference after a checkpoint when in state IA, designated by P_{cr1} in Figure 4, causes a transition to state IIB. Table 2 illustrates this transition. The active copy has a real frame f_1 which has not been changed since being read from disk. The old-checkpoint mapping points to a different real frame (f_2) whose contents are not needed. The old-checkpoint frame (f_2) is freed and the old-active frame is moved to the new active. The checkpoint page now exists only on disk. The change bit for f_1 is set to invalidate the contents of d_2.

	Mapping	Memory	Disk
Before	m_l	$f_1, A, 0$	d_1, A
	$m_{\bar{l}}$	$f_2, B, 0$	d_2, ψ
After	m_l	$\emptyset$	d_1, A
	$m_{\bar{l}}$	$f_1, A, 1$	d_2, ψ

Table 2: State IA to IIB via a reference

State IB

A first reference after a checkpoint when in state IB, designated by P_{cr2} in Figure 4, causes the page to remain in state IB. Table 3 illustrates this transition. In state IB the old-active is modified ($c_l = 1$) which means that the frame cannot be simply moved to the new-active (because d_1 is not a valid checkpoint page). In this case, the contents of frame f_1 are copied to frame f_2 which sets the change bit.

	Mapping	Memory	Disk
Before	m_l	$f_1, C, 1$	d_1, A
	$m_{\bar{l}}$	f_2, B, ψ	d_2, ψ
After	m_l	$f_1, C, 1$	d_1, A
	$m_{\bar{l}}$	$f_2, C, 1$	d_2, ψ

Table 3: State IB to IB via a reference

State IIA

A first reference after a checkpoint when in state IIA, designated by P_{cr3} in Figure 4, causes a transition to state IIB. Table 4 illustrates this transition. In this case the old-checkpoint (new-active) is not in real storage and the old-active has not been modified since being brought in from the disk. This means that the old-active disk copy (d_1) can be used as the checkpoint copy. Thus, the checkpoint is performed by simply having the new-active mapping point to the main storage frame (f_1). The change bit must be set for the new active to effectively invalidate the disk copy (d_2).

	Mapping	Memory	Disk
Before	m_l	$f_1, A, 0$	d_1, A
	$m_{\bar{l}}$	$\emptyset$	d_2, ψ
After	m_l	$\emptyset$	d_1, A
	$m_{\bar{l}}$	$f_1, A, 1$	d_2, ψ

Table 4: State IIA to IIB via a reference

State IIB

A first reference after a checkpoint when in state IIB, designated by P_{cr4} in Figure 4, causes a transition to state IB. Table 5 illustrates this transition. In state IIB, the old-active has been modified which means that the old-active disk version (d_1) cannot be used as the checkpoint copy. A new frame is allocated (f_2) and the data is copied to this frame. The change bit is automatically set during the copy to invalidate the new-active disk copy (d_2).

	Mapping	Memory	Disk
Before	m_l	$f_1, B, 1$	d_1, A
	$m_{\bar{l}}$	$\emptyset$	d_2, ψ
After	m_l	$f_1, B, 1$	d_1, A
	$m_{\bar{l}}$	$f_2, B, 1$	d_2, ψ

Table 5: State IIB to IB via a reference

State III

A first reference after a checkpoint when in state III, designated by P_{cr5} in Figure 4, causes a transition to state IIB. Table 6 illustrates this transition. State III occurs when the old-active has been paged-out but the old-checkpoint remains in storage. This only occurs if the operating system steals the active frame but leaves the checkpoint frame. This does not appear to be a smart decision but is covered for completeness. The data is paged in from d_1 to f_2. The change bit is set to invalidate the disk copy (d_2).

State IV

A first reference after a checkpoint when in state IV, designated by P_{cr6} in Figure 4, causes a transition to state IIB. Table 7 illustrates this transition. State IV considers the situation when both the active and checkpoint frames have been removed from storage. The valid version of the data is on d_1. A new frame is allocated (f_2) and filled with the contents of d_1. The

	Mapping	Memory	Disk
Before	m_l	$\emptyset$	d_1, A
	$m_{\bar{l}}$	$f_2, B, 0$	d_2, ψ
After	m_l	$\emptyset$	d_1, A
	$m_{\bar{l}}$	$f_2, A, 1$	d_2, ψ

Table 6: State III to IIB via a reference

change bit is set in order to invalidate the disk copy (d_2).

	Mapping	Memory	Disk
Before	m_l	$\emptyset$	d_1, A
	$m_{\bar{l}}$	$\emptyset$	d_2, ψ
After	m_l	$\emptyset$	d_1, A
	$m_{\bar{l}}$	$f_2, A, 1$	d_2, ψ

Table 7: State IV to IIB via a reference

Transition	States	Actions
P_{cr1}	IA$\rightarrow$IIB	$r_l \leftrightarrow r_{\bar{l}}$ free r_l frame $c_{\bar{l}} = 1$
P_{cr2}	IB$\rightarrow$IB	$(f_2) \leftarrow (f_1)$ $c_{\bar{l}} = 1$
P_{cr3}	IIA$\rightarrow$IIB	$r_l \leftrightarrow r_{\bar{l}}$ $c_{\bar{l}} = 1$
P_{cr4}	IIB$\rightarrow$IB	allocate frame f_2 $(f_2) \leftarrow (f_1)$ $c_{\bar{l}} = 1$
P_{cr5}	III$\rightarrow$IIB	page in from d_1 to f_2 $c_{\bar{l}} = 1$
P_{cr6}	IV$\rightarrow$IIB	allocate frame f_2 page in from d_1 to f_2 $c_{\bar{l}} = 1$

Table 8: Summary for first reference after checkpoint

3.2 Initial Page Reference

When the page is referenced for the first time $l = 0$, $r_l = d_l = \emptyset$ and $v = 0$ which causes a transition to state IIB. The change bit is set for the frame. This is an optimization allowing the allocated disk slot to contain whatever was there during the prior use. This means that I/O is not required to zero out the disk slot when initially allocated (the main storage frame is cleared). The reasoning is that the page is most likely modified soon after the initial reference.

3.3 Program Store Transitions

A store into a previously unmodified page ($c_l = 0$) causes a state transition as shown by P_{st} in Figure 4. In state IA ($r_l \neq \emptyset, r_{\bar{l}} \neq \emptyset$), a store causes a transition to state IB. In state IIA ($r_l \neq \emptyset, r_{\bar{l}} = \emptyset$) a store causes a transition to state IIB.

3.4 Normal Page Fault Transitions

In states III and IV, the active mapping does not have a real frame and a normal reference ($V = v$) causes a page fault as shown by P_{nf} in Figure 4. In state III, a page fault causes a transition to state IA. Note that this assumes the page is not modified. Therefore, a *store* causing a page fault would first load the page unmodified (to state IA) and then the store would cause an immediate transition to state IB. Similarly, a page fault in state IV causes a transition to state I[illegible]. However, if the fault was due to a store, then an immediate transition to IIB occurs.

3.5 Operating System Transitions

There are several transitions that can be caused by the operating system during real storage management operations. The transition O_1 occurs when the checkpoint frame is removed while the active frame is retained. The transition O_2 occurs when the active frame (not modified) is stolen and the checkpoint frame is retained. This should not occur as a normal event as a preferred event would be to take the checkpoint frame. The transition O_o occurs when a modified frame is written out to the disk slot while the frame is retained and the change bit reset. Finally, O_3 occurs when the active frame has been stolen after the checkpoint frame has already been stolen.

4 Implementation and Optimizations

This section describes some important issues which are relevant to the implementation of the virtual checkpoint scheme. The first issue regards the test for $V = v$ which must be done on every translation of the virtual address. Two techniques are suggested to make the performance of the test for the $V = v$ condition very fast:

1. Add a v field in the TLB entry and check in parallel with the TLB access.

2. Purge the TLB at the checkpoint time and only check v on TLB misses.

When the check for $V = v$ results in an unequal condition, there are several other conditions which must be tested (see Table 1) to determine the current state and actions required. The current state of the page could be implemented as a 6-state finite state machine. This would require an additional 3 bits per page map entry. Furthermore, the actions described in Table 8 could be implemented in hardware to make the checkpoint process transparent to the operating system.

The space overhead for additional mapping fields ($r_{\bar{l}}, d_{\bar{l}}, v, l$) can be greatly reduced by using a multilevel page table (e.g., segment table and page tables). Furthermore, the segment table could support an additional bit for the type of page table supported. This could allow the use of the proposed type of page table or a standard type.

In the basic scheme presented in Figure 4, state IB can be a predominate state and the overhead to process the checkpoint is a full page memory to memory copy. This may be excessive when the page was not really modified during prior checkpoint intervals. Note that $c_{\bar{l}}$ is set on the entry to state IB because of the uncertainty of $d_{\bar{l}}$. If we could force our way into state IIA with the data at r_l, d_l and $d_{\bar{l}}$ being equal, then the checkpoint could be handled in state IIA by simply moving the frame address from r_l to $r_{\bar{l}}$. This would be a significant performance enhancement for systems with large amounts of read only data. To implement this optimization, a "data-equal" bit is proposed for the page translation table. This is set when $r_l = d_l = d_{\bar{l}}$ and reset whenever d_l or $d_{\bar{l}}$ is written. This would be supported in state IIA such that checkpoints in this state would remain in state IIA and could be pipelined to produce no significant delay. Since a large percentage of the pages referenced in any interval are not modified, this optimization could provide significant gains at very little cost. This is referred to as the DE optimization. Although this appears as a fundamental improvement, there is a set up cost to get the equal copies. Therefore, this is studied as an optimization and not incorporated into the basic scheme.

5 Performance Analysis

This section presents a preliminary study of the performance of the virtual checkpoint scheme. A program address trace from a commercial batch program on a large system is used. This is very similar to the commercial trace used in[15]. Trace "batch5" consists of 4.67 million references which reference 1.86 MB of 4K pages and trace "batch17" consists of 2.97 million references which reference 1.75 MB of 4K pages.

The checkpoint interval depends on aspects such as the total execution time, the failure rate, and the time to perform the checkpoint. We have selected to study the checkpoint overhead as a function of deterministic intervals. Given a checkpoint interval, the overhead depends on the number of unique pages referenced in each interval. We have measured this as a function of the checkpoint interval for the two virtual memory address traces in addition to the average number of unique modified pages. The results for trace batch5 are shown in Figure 5. For short intervals of 1000 references, the number of unique pages is on the order of 20. The overhead can be greatly reduced by going to larger intervals (e.g., batch5 references an average of 131 unique pages in a 100,000 reference interval).

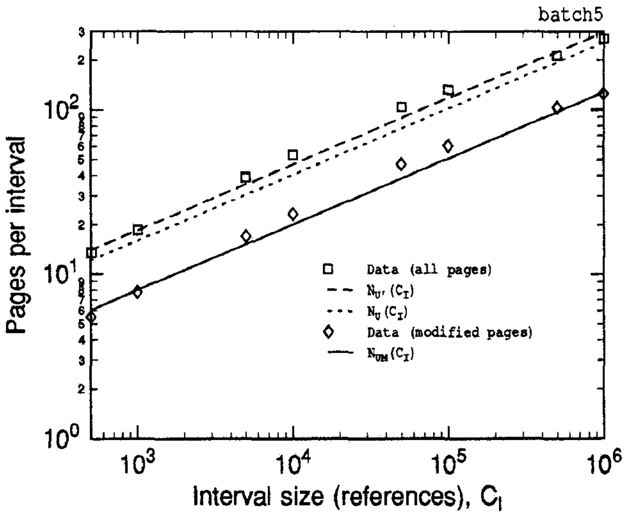

Figure 5: Pages per interval vs. interval size

An analytic model is now developed to measure the number of unique pages referenced per interval. This model is similar to one reported by Thiébaut where he looked at the number of unique cache lines referenced during the entire address trace[19]. Thiébaut's model looks at the trace over two regions (before and after the cache fills). The line before the knee is fitted with a linear regression and the portion after the knee is fitted with straight line from the knee to the last point. The model presented in this paper does not require such accuracy and its unique property is the ability to select the parameters based only on the total references and total blocks. Let N_T be the total number of references in the trace, N_P the total number of 4K pages referenced and C_I the checkpoint interval. Our base model is founded upon the simple observation that an interval of $C_I = N_T$ would lead to N_P unique pages being referenced. Furthermore, experiments show that a log-log graph of unique pages is almost linear[19]. The data points also essentially pass through the origin (i.e., an interval of 1 reference must reference exactly 1 unique page). Thus, our base model for the number of unique pages is:

$$N_U(C_I) = C_I^{\frac{\log N_P}{\log N_T}} \qquad (1)$$

Equation 1 is plotted in Figure 5 and results in an absolute error, averaged over each data point, of 16.4% and 22.5% for batch5 and batch17 respectively (the model underestimates the data in both cases).

The model is improved by forcing the curve to pass through the data point $C_I = 1,000$ with the slope taken from Equation 1. This is an arbitrary choice; it is reasoned that this point could be easily determined by trace analysis or actual measurements. The term N_{1000} is the number of unique pages referenced over the first 1000 references. The refined model for the number of unique pages per interval C_I is:

$$N_{U'}(C_I) = N_{1000}\left[\frac{C_I}{1000}\right]^{\frac{\log N_P}{\log N_T}} \qquad (2)$$

Equation 2 is plotted in Figure 5 and results in an absolute error, averaged over each data point, of 8.1% and 5.0% for batch5 and batch17, respectively.

The final model is that of the number of unique pages that are modified. Through observation of the data, it was found that for all interval sizes, the ratio of the number of unique pages to the number of those modified remained constant. Using N_M as the total number of pages ever modified in the trace, the following model is used for the average number of unique modified pages per interval:

$$N_{UM}(C_I) = \frac{N_M}{N_P} N_{U'}(C_I) \qquad (3)$$

Equation 3 is plotted in Figure 5 and results in an absolute error, averaged over each data point, of 9.9% and 10.0% for batch5 and batch17, respectively.

The model is now used to formally quantify the effect of increasing the checkpoint interval. We have already stated that the overhead can be reduced by simply increasing the checkpoint interval. Assume that the overhead can be approximated by

$$O_a(C_I) = \frac{N_{U'}(C_I)}{C_I} \qquad (4)$$

The result of increasing the checkpoint interval by an order of magnitude is now determined. Specifically, we want to solve for β in $O_a(C_I)/\beta = O_a(10C_I)$. Solving this gives

$$\beta = 10^{\left[1-\frac{\log N_P}{\log N_T}\right]} \qquad (5)$$

The values of $log N_P / log N_T$ for batch5 and batch17 are 0.4015 and 0.4096, respectively. Thus, an increase of the checkpoint interval by a factor of 10 reduces the overhead by a factor of approximately 3.9.

The data presented thus far is from a commercial batch workload. We want to show that the results are applicable to workloads that would be found in supercomputer applications. Martin shows the memory size (in words) and computational requirements for aerodynamic calculations of airflow for various problem scopes and solution techniques[12]. N_T was roughly estimated as the total number of floating point operations and N_P as the total number of pages (assuming 1024 words to a page). The values for $log N_P / log N_T$ are shown in Table 9. An interesting comparison of

	Non Linear, Inviscid	Reynolds-Avg, Navier-Stokes	Large-Eddy Simulation
Air Foil	-	0.26	0.41
Wing	0.34	0.42	0.47
Aircraft	0.40	0.44	0.51

Table 9: $\frac{log N_P}{log N_T}$ values

scalar to vector code is made in[16] where a dramatic reduction in the memory references is found. For the same problem, the instructions decrease because several instructions are replaced by single vector instructions. The data references are reduced because of the heavy use of vector registers. In Table 10 the values for $log N_P / log N_T$ are calculated. The vector numbers are larger because the number of references are reduced while the amount of data is held constant. Both these studies have values near the original traces and produce β values in the range of 5.5 to 3.1. Therefore, these programs should also exhibit the property of decreasing the overhead by increasing the checkpoint interval.

	Scalar	Vector
LIN EQ	0.314	0.356
FFT1K	0.365	0.421
SIMPLE	0.411	0.417
ARC3D	0.377	0.395

Table 10: $\frac{log N_P}{log N_T}$ values

We now briefly compare our schemes to other proposals. As stated, our ultimate goal is for the memory system to be considered reliable. From this point of view, the scheme could not be compared to the twin-page, the snapshot or shadow paging as these are all disk based. However, our scheme does work in a situation where the disk is considered the stable storage. In this case our scheme would require a disk write of all modified pages and modified portions of the page tables. Our scheme would do better than the snapshot scheme because ours only requires saving of referenced data while the snapshot requires saving of all data. Ours would be roughly comparable to the twin-page techniques as it would require more processing to force out the page tables, but this could be compensated for with minor differences such as only having to fetch a single page.

6 Future Work

The next step in the performance analysis is to further understand the dynamic behavior of the virtual checkpoint scheme. This involves assigning costs to the transitions shown in Figure 4 and simulating the address trace under some virtual memory management algorithm. The DE optimization suggested in Section 4 should also provide significant improvements. A preliminary analysis indicates the instruction overhead to be on the order of 5% for intervals of 10^6 references[5]. These results also show the overhead obeys the rule derived for β in Equation 5.

7 Conclusions

This paper has presented a technique that embeds the support for checkpoint and rollback recovery directly into the virtual memory translation hardware. The scheme is general enough so that it could be implemented on various scopes of data such as a portion of an address space, a single address space or multiple address spaces. We have presented a basic model which measures the amount of work required by the scheme as a function of the checkpoint interval size. Using this model the degree to which the overhead decreases as the interval size increases is shown. Characteristics of several supercomputer applications are shown to agree with the model presented. Further work is required to study the details of the performance as well as advanced topics such as dynamic interval sizes and algorithms to more aggressively determine the read-only pages.

References

[1] Ahmed, R. E., Frazier, R. C., and Marinos, P. N., "Cache-aided rollback recovery (carer) algorithms for shared-memory multi-processors," in *20th Symp. on Fault-Tolerant Computing*, pp. 82–88, June 1990.

[2] Banâtre, M. and Joubert, P., "Cache management in tightly coupled fault-tolerant multiprocessor," in *20th Symp. on Fault-Tolerant Computing*, pp. 89–96, June 1990.

[3] Banâtre, M. and Muller, G., "Ensuring data security and integrity with a fast stable storage," in *4th Int. Conference on Data Engineering*, pp. 285–293, Feb. 1988.

[4] Bernstein, P. A., "Sequoia: A fault-tolerant tighly coupled multiprocessor for transaction processing," *COMPUTER*, vol. 21, pp. 37–45, Feb. 1988.

[5] Bowen, N. S. and Pradhan, D. K., "Virtual checkpoints: Architecture and performance," Tech. Rep. TR-91-CSE-15, University of Massachusetts, Apr. 1991.

[6] Chang, A. and Mergen, M. F., "801 storage: Architecture and programming," *ACM Trans. Computer Systems*, vol. 6, pp. 28–50, Feb. 1988.

[7] Hunt, D. B. and Marinos, P. N., "A General Purpose Cache-Aided Rollback Recovery (CARER) Technique," in *17th Symp. on Fault-Tolerant Computing*, pp. 170–175, IEEE Computer Society, June 1987.

[8] Kamel, A., Sguazzero, P., and Zecca, V., "Large scale computing on clustered vector multiprocessors," in *Supercomputing '90*, pp. 418–427, Nov. 1990.

[9] Lee, P. A., Ghani, N., and Heron, K., "A recovery cache for the pdp-11," *IEEE Transactions on Computers*, vol. C-29, pp. 546–549, June 1980.

[10] Li, K., Naughton, J. F., and Plank, J. S., "Real-time, concurrent checkpoint for parallel programs," in *Second ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP)*, pp. 79–88, SIGPLAN Notices, Vol. 25, N.3, Mar. 1990.

[11] Lorie, R. A., "Physical integrity in a large segmented database," *ACM Transactions on Database Systems*, vol. 2, pp. 91–104, Mar. 1977.

[12] Martin, J. L., "Supercomputer performance evaluation: The comparative analysis of high-speed architectures against their applications," in *Performance Evaluation of Supercomputers* (Martin, J., ed.), pp. 3–20, North-Holland, 1988.

[13] Reuter, A., "A fast transaction-oriented logging scheme for undo recovery," *IEEE Transactions on Software Engineering*, vol. SE-6, pp. 348–356, July 1980.

[14] Simmons, M. L. and Wasserman, H. J., "Performance evaluation of the ibm risc system/6000: Comparison of an optimized scaler processor with two vector processors," in *Supercomputing '90*, pp. 132–141, Nov. 1990.

[15] So, K. and Rechtschaffen, R. N., "Cache operations by mru change," *IEEE Transactions on Computers*, vol. 37, pp. 700–709, June 1988.

[16] So, K. and Zecca, V., "Cache performance of vector processors," in *15th Annual Symp. on Computer Architecture*, pp. 261–268, May 1988.

[17] Staknis, M. E., "Sheaved memory: Architectural support for state saving and restoration in paged systems," in *3rd Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 96–102, ACM, Apr. 1989.

[18] Thatte, S. M., "Persistent memory: A storage architecture for object-oriented database systems," in *Proc. 1986 International Workshop on Object-Oriented Database Systems*, pp. 148–159, Sept. 1986.

[19] Thiébaut, D., "On the fractal dimension of computer programs and its applications to the prediction of the cache miss ratio," *IEEE Transactions on Computers*, vol. 38, pp. 1012–1026, July 1989.

[20] Wu, K.-L. and Fuchs, W. K., "Rapid transaction-undo recovery using twin-page storage management," Tech. Rep. RC-15912, IBM Research Division, July 1990.

[21] Wu, K.-L. and Fuchs, W. K., "Recoverable distributed shared virtual memory," *IEEE Transactions on Computers*, vol. 39, pp. 460–469, Apr. 1990.

[22] Wu, K.-L., Fuchs, W. K., and Patel, J. H., "Error recovery in shared memory multiprocessors using private caches," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, pp. 231–240, Apr. 1990.