Consistent State Restoration in Shared Memory Systems

Roberto Baldoni Jean-Michel Hélary
Achour Mostefaoui Michel Raynal
IRISA, Campus de Beaulieu
35042 Rennes cedex, FRANCE
<name>@irisa.fr

Abstract

In many systems, backward recovery constitutes a classical technique to ensure fault-tolerance. It consists in restoring a computation in a consistent global state, saved in a global checkpoint, from which this computation can be resumed. A global checkpoint includes a set of local checkpoints, one from each process which correspond to local states dumped onto stable storage. In this paper, we are interested in defining formally the domino effect *for shared memory systems be the shared memory a physical one (as in multiprocessor systems) or a virtual one (as in distributed shared memory systems) and in designing a domino–free adaptive algorithm. These results lie on a necessary and sufficient condition which shows when a set of local checkpoints can belong to some consistent global checkpoint.*

1. Introduction

The shared variables programming paradigm is the classical one for programs executed on shared memory multiprocessors. Since several years, protocols implementing this paradigm on distributed memory parallel machines [6] or on top of distributed systems kernels [1], made it independent of a particular type of machine. Consequently, the shared memory model has become still more attractive as, in a distributed context, it simplifies load balancing, data sharing and scalability.

Here we address the problem of a consistent state restoration of a failed computation by using checkpointing which is a typical backward recovery technique. A local checkpoint is a local state of a process and a global checkpoint includes a set of local checkpoints, one from each process, plus the values of the shared variables. So, if a failure occurs, the computation is restored by loading a previously dumped consistent global checkpoint and then restarted. Informally, a global checkpoint is consistent if, when it includes the reading of a value from some variable, it includes also the corresponding writing. The determination of consistent global checkpoints is a fundamental point to avoid, after a failure occurrence, an unbounded rollback of processes due to the impossibility to restore a consistent global state. This disastrous avalanche, which actually undoes the computation execution, is called the domino effect [11].

In this paper, a formal definition of a computation including local and global checkpoints in the shared memory model is given in Section 2. Then, a consistency theorem is stated in Section 3. This theorem states a necessary and sufficient condition for local checkpoints of a set of processes to be members of a same consistent global checkpoint. Actually, it extends to shared memory systems the Netzer-Xu result [9] obtained for reliable point-to-point message passing systems, and its proof, given in the appendix, is adapted from the one given in [9] (the interested reader can refer also to [3]). Based on the previous theorem, Section 4 proposes a formal definition for the domino effect (this effect is usually characterized only by operational arguments). Finally, Section 5 shows a domino-free checkpointing algorithm. This algorithm is fully distributed, adaptive and purely local (*i.e.* processes execute it in an uncoordinated manner). If the shared memory were implemented in a distributed system, this checkpointing algorithm would require neither additional control messages nor the piggybacking of control information.

2. Shared Memory Model

A shared memory system (P, X) is composed of a set P of n sequential processes $P_1, \ldots, P_n$ that communicate and exchange information via a finite set X of shared variables (objects). Variables that can be accessed only by one

0-8186-7876-3/96 $10.00 © 1997 IEEE

process (i.e., local variables) are not considered through this paper. A shared variable can be accessed by a read or a write operation. Each execution of an operation by a process produces an event. More precisely, a *write* event on a variable x issued by process P_i is denoted $w_i(x)v$ where v is the new value associated with x; similarly, $r_i(x)v$ denotes a *read* event on a variable x issued by P_i and returning the value v from the variable x. Moreover, the set of events produced by P_i is denoted h_i. For simplicity, as in [7], we assume that all values written into a variable x are distinct and that each variable x has an initial value in_x (written by an initial fictitious write operation).

Each process P_i runs asynchronously a local program, made of read and write operations, whose execution is modeled by a sequence of events. Such a sequence $\widehat{h_i}$ is called a local history of P_i and e_i^s denotes the s-th event ($s \geq 0$) produced by P_i: $\widehat{h_i} = e_i^0 e_i^1 \dots e_i^s e_i^{s+1} \dots (s \geq 0)$.

The set of all the events $H = \bigcup_i h_i$ is structured by *a local precedence* relation, denoted $\rightarrow_l$, defined in the following way:

$$e_i^s \rightarrow_l e_j^t \Leftrightarrow i = j \text{ and } s < t$$

Let us consider n local histories $\{\widehat{h_1}, \dots, \widehat{h_n}\}$, $\widehat{h_i}$ being produced by P_i. Interactions, through shared variables, between processes define on H a binary relation called *read-from* and denoted $\rightarrow_{rf}$. More formally, $\rightarrow_{rf}$ is a relation, defined on distinct pairs of write and read events, that satisfies the following property:

$$e_i^s \rightarrow_{rf} e_j^t \Leftrightarrow e_i^s = w_i(x)v \text{ and } e_j^t = r_j(x)v.$$

It is clear that distinct read events on the same variable x can return the same value. Moreover, we assume that each value returned by a read event has always been defined by a write event (possibly the initial ficticious write on x). This assumption is formally expressed in the second point of the following definition.

Definition 2.1 *A history (or a* computation*)* $\widehat{H}$ *of a shared memory system* (P, X) *is a partial order* $\widehat{H} = (H, \rightarrow_H)$ *such that:*

- $H = \bigcup_i h_i$
- $(\forall e_j^t \in H)\ (e_j^t = r_j(x)v \Rightarrow (\exists e_i^s\ :\ e_i^s = w_i(x)v))$
- $e_i^s \rightarrow_H e_j^t \Leftrightarrow \begin{array}{l} e_i^s \rightarrow_l e_j^t \text{ or} \\ e_i^s \rightarrow_{rf} e_j^t \text{ or} \\ \exists e_k^v : e_i^s \rightarrow_H e_k^v \text{ and } e_k^v \rightarrow_H e_j^t \end{array}$

Two events e_i^s and e_j^t are *concurrent* in $\widehat{H}$ if we have neither $e_i^s \rightarrow_H e_j^t$ nor $e_j^t \rightarrow_H e_i^s$.

The shared memory model can be refined by imposing additional constraints on computations. A set of such additional constraints actually define a consistency criterion. The fact that an overwritten value cannot be read again is expressed by the concept of *legality* [2, 7, 10] for read events. A read event $r(x)v$ is legal in $\widehat{H}$ if $\nexists\ e_i^s\ :\ w(x)v \rightarrow_H e_i^s \rightarrow_H r(x)v$ where $e_i^s = r(x)u$ or $e_i^s = w(x)u$ and $u \neq v$. A computation $\widehat{H}$ is legal if all its read events are legal.

Two consistency criteria for shared memory are usually considered:

- *Sequential consistency.* A computation $\widehat{H}$ is sequentially consistent [5, 8] if it has a linear extension[1] in which all read events are legal. Intuitively, this consistency criterion defines $\widehat{H}$ as correct if it could have been produced by multiplexing processes on a monoprocessor system. Actually, a sequentially consistent computation totally orders write events.
- *Causal consistency.* A computation $\widehat{H}$ is causally consistent [2, 10] if all read operations are legal. In that case, a process never gets an overwritten value, but distinct processes can disagree on the value of a variable.

In the following, we assume (i) the owner of a variable is defined as the last process that wrote into it (as in [4, 6]) and (ii) computations are sequentially consistent; this implies: at any time there is only one owner for each shared variable.

2.1. Consistent Global Checkpoints

The local state of a process is defined as the value of its local variables plus the values of the variables of which it is the current owner. The initial state of P_i is σ_i^0. The event e_i^s moves P_i from the local state σ_i^s to the local state σ_i^{s+1}. So the local state σ_i^s corresponds to the partial local history $\widehat{h_i^{s-1}} = e_i^0 e_i^1 \dots e_i^{s-1}$. By definition e_i^x *belongs to* σ_j^s ($e_i^x \in \sigma_j^s$) if $i = j$ and $x < s$.

A *local checkpoint* c_i of a process P_i is a local state (the vice versa is not true) of a process that has been dumped onto stable storage (it is said that P_i has taken a local checkpoint). Such a checkpoint is taken "atomically" with respect to read and write events produced by P_i.

We consider in the following a computation $\widehat{H}$ and a set of local checkpoints $C_{\widehat{H}}$ defined on $\widehat{H}$. c_i^s represents the s-th checkpoint taken by P_i and corresponds to some local state σ_i^t with $t \geq s$. By convention, we assume that each

[1] A linear extension $\widehat{S} = (S, \rightarrow_S)$ of a partial order $\widehat{H} = (H, \rightarrow_H)$ is a topological sort of $\widehat{H}$. i.e., (i) $H = S$, (ii) $\forall e_i^s, e_j^t \in S :: e_i^s \rightarrow_S e_j^t$ or $e_j^t \rightarrow_S e_i^s$ and (iii) $e_i^s \rightarrow_H e_j^t \Rightarrow e_i^s \rightarrow_S e_j^t$.

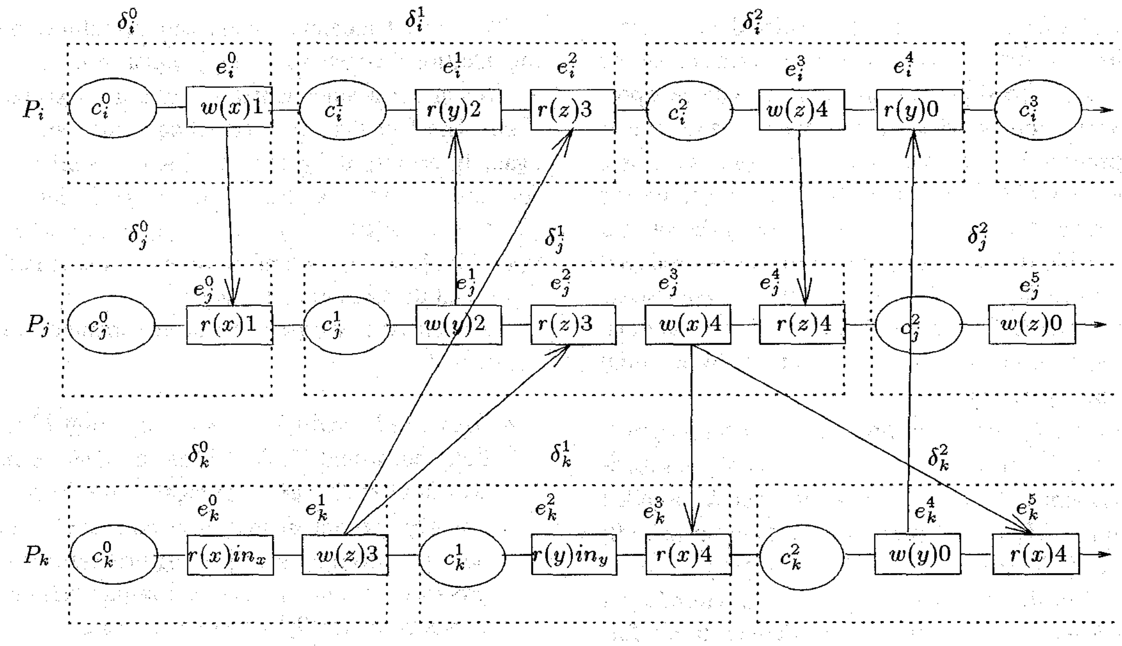

Figure 1. Example of a computation with a set of local checkpoints

process takes an initial checkpoint c_i^0, corresponding to σ_i^0, when it starts. We also assume that every event belongs to a local checkpoint; this assumption means that every process has a "final" local checkpoint. Figure 1 (without considering rectangular boxes with dotted lines) depicts a set $C_{\widehat{H}}$ of local checkpoints associated with a sequentially consistent computation where pairs of the *read-from* relation are indicated by thick arrows.

Now, we are in the position to give the definition of *orphan* read event:

Definition 2.2 *Let us consider a read event $r_i(x)v$ and let P_j be the process that produced the corresponding write event $w_j(x)v$. The read event $r_i(x)v$ is called* orphan *with respect to the ordered pair of checkpoints $(c_j^t, c_i^s,)$ if $r_i(x)v$ belongs to c_i^s while $w_j(x)v$ does not belong to c_j^t.*

Informally, with respect to an ordered pair of local checkpoints, an orphan read event returns a non-written value. Examples of orphan read events are depicted in Figure 1: the event e_i^1 is orphan with respect to the ordered pair (c_j^1, c_i^2), the event e_k^3 is orphan with respect to (c_j^1, c_k^2).

A global checkpoint is a set of local checkpoints, one from each process: $\{c_1^{x_1}, c_2^{x_2}, \ldots c_n^{x_n}\}$ where $c_i^{x_i}$ is a local checkpoint from P_i.

Definition 2.3 *Let $\mathcal{I} = \{1, \ldots, n\}$ and $\mathcal{C} = \{c_i^{x_i}\}_{i \in \mathcal{I}}$ be a global checkpoint. $\mathcal{C}$ is consistent if and only if, it has no orphan read events. i.e., $\forall i \in \mathcal{I}, \forall x \in X$:*
$(r_i(x)u \in c_i^{x_i} \Rightarrow \exists P_j \in P : w_j(x)u \in c_j^{x_j})$

As an example, $\{c_i^1, c_j^1, c_k^1\}$, $\{c_i^1, c_j^2, c_k^2\}$ and $\{c_i^2, c_j^2, c_k^2\}$ are global checkpoints of the computation shown in Figure 1. While $\{c_i^1, c_j^1, c_k^1\}$ is consistent, $\{c_i^1, c_j^2, c_k^2\}$ and $\{c_i^2, c_j^2, c_k^2\}$ are not consistent due to the orphan read event e_j^4.

3. Mutual Consistency of Local Checkpoints

Using an abstraction level defined by checkpoints, any local checkpoint c_i^s defines an interval. We call *checkpoint interval* δ_i^s the sequence of events occurring at P_i between c_i^s and c_i^{s+1}, including the event having produced c_i^s and excluding the event producing the next local checkpoint c_i^{s+1}, if it exists. In the computation shown in Figure 1, checkpoint intervals are depicted as rectangular boxes with dotted lines. On checkpoint intervals we define the following relation of precedence $\prec$:

Definition 3.1 *δ_i^x precedes δ_j^y (denoted $\delta_i^x \prec \delta_j^y$) if and only if:*

1. *$j = i$ and $y = x + 1$, or*
2. *$r_j(x)v$ belongs to δ_j^y and the corresponding $w_i(x)v$ belongs to δ_i^x, or*
3. *$\exists z \, \exists k : \delta_i^x \prec \delta_k^z \wedge \delta_k^z \prec \delta_j^y$.*

As an example, Figure 2 shows the relation $\prec$ of the computation depicted in Figure 1 where, for clarity's sake,

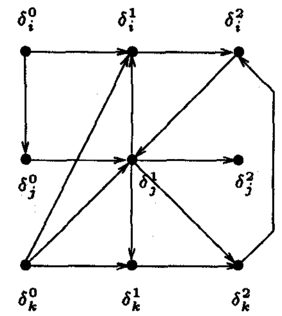

Figure 2. Relation of precedence $\prec$

we do not consider precedences due to transitivity (point 3. of Definition 3.1).

It results from Definition 2.1 that, for every checkpoint interval δ_i^x, the set of checkpoint intervals δ_j^y such that $\delta_j^y \prec \delta_i^x$ is finite.

On this relation relies a necessary and sufficient condition to decide if an arbitrary set of local checkpoints can belong to a global checkpoint (Due to limited space the proof appear in the appendix, the interested reader can refer also to [3]).

Theorem 3.2 *Let $\mathcal{I} \subseteq \{1, \ldots, n\}$ and $C = \{c_i^{x_i}\}_{i \in \mathcal{I}} \subseteq C_{\widehat{\mathcal{H}}}$ be a set of local checkpoints associated with the computation $\widehat{H}$. Then C is a subset of a consistent global checkpoint if and only if:*

$$(\forall i \in \mathcal{I},\ \forall j \in \mathcal{I}) \ :: \ (x_j = 0) \vee (\neg(\delta_i^{x_i} \prec \delta_j^{x_j - 1}))$$

Hence any two local checkpoints c_i^s and c_j^t cannot belong to the same global checkpoint if the checkpoint interval δ_i^s, generated by c_i^s, precedes a checkpoint interval that is in the past of the checkpoint interval δ_j^t generated by c_j^t. This suggests an interesting corollary to test if a local checkpoint can never be a member of some consistent global checkpoint:

Corollary 3.3 *A local checkpoint c_i^s is* useless *if and only if $(s > 0) \wedge (\delta_i^s \prec \delta_i^{s-1})$*

Non-useless checkpoints are called *useful*. For example, the local checkpoints c_i^2 and c_k^2, depicted in Figure 1, are useless. Indeed, we have $\delta_j^1 \prec \delta_i^1$ and $\delta_i^2 \prec \delta_j^1$ for c_i^2 and $\delta_i^2 \prec \delta_j^1 \prec \delta_k^1$ and $\delta_k^2 \prec \delta_i^2$ for c_k^2.

4. Characterizing the Domino Effect

The domino effect [11] is the well-known phenomenon describing, during recovery, the rollback of processes while determining a consistent global checkpoint from which a computation can consistently be resumed. A shared memory system suffers the domino effect if processes rollback unboundedly. Such a "recovery backward progress" of a process, due to the impossibility to find a consistent global checkpoint, is captured by Corollary 3.3, where for non-useful checkpoints, we have $\delta_i^s \prec \delta_i^{s-1}$. This suggests to use precedence on checkpoint intervals to formally define a bounded domino effect.

Definition 4.1 *Let $\widehat{H}$ be a computation, $C_{\widehat{\mathcal{H}}}$ be a set of checkpoints defined on $\widehat{H}$ and $\alpha \geq 0$ be an integer. The domino effect is α-bounded in $C_{\widehat{\mathcal{H}}}$ if and only if:*

$$(\forall P_i \in P) \ :: \ (\delta_i^s \prec \delta_i^t \wedge s > t \Rightarrow s - t \leq \alpha)$$

If $C_{\widehat{\mathcal{H}}}$ is such that the domino effect is α-bounded, we say that $C_{\widehat{\mathcal{H}}}$ is α-bounded. Let us consider the computation $\widehat{H}$ and the set of checkpoints $C_{\widehat{\mathcal{H}}}$ shown in Figure 3 (for simplicity's sake we do not explicit objects involved in the *read-from* relations); the corresponding precedence relation on checkpoint intervals is shown in Figure 4. In this computation orphan read events due to the *read-from* relation plus the useless checkpoint c_2^2 create, by transitivity, the precedence $\delta_4^3 \prec \delta_4^0$ that fixes an upper bound to the domino effect which is 3-bounded.

$C_{\widehat{\mathcal{H}}}$ is defined on the fly by a checkpointing algorithm as $\widehat{H}$ progresses. So neither $\widehat{H}$ nor $C_{\widehat{\mathcal{H}}}$ are known in advance. Suppose that, for a given checkpointing algorithm, we are able to produce a non-negative integer α such that, for any $\widehat{H}$, it will produce a set $C_{\widehat{\mathcal{H}}}$ which is α-bounded; then, we know that, in case of failure, a recovery will not require processes to rollback unboundedly. This consideration provides the following definition for domino boundedness.

Definition 4.2 *A checkpointing algorithm ensures* domino-bounded rollback recovery *if it is possible to define an integer $\alpha \geq 0$ such that, for any $\widehat{H}$, the set $C_{\widehat{\mathcal{H}}}$ produced by this algorithm is α-bounded.*

Definition 4.3 *A checkpointing algorithm ensures* domino-free rollback recovery *if for any $\widehat{H}$, the set $C_{\widehat{\mathcal{H}}}$ produced by this algorithm is 0-bounded (such a checkpointing algorithm is called domino-free).*

For domino-free checkpointing algorithms, let us introduce the following simple lemma whose proof follows directly from Corollary 3.3 and from Definitions 4.1 and 4.3:

Lemma 4.1 *A checkpointing algorithm is* domino-free *if and only if it does not produce useless checkpoints.*

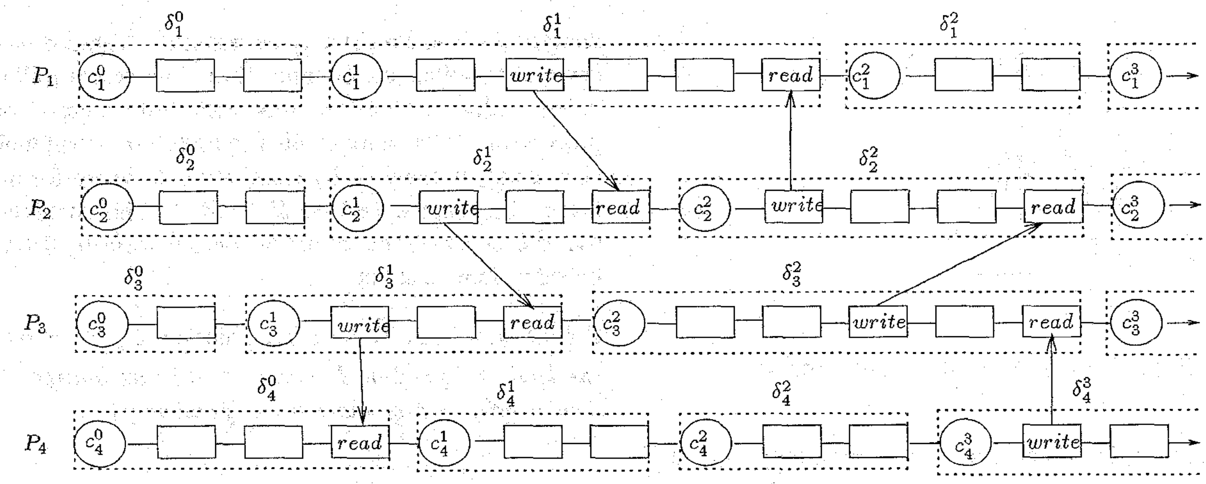

Figure 3. Example of a domino effect 3-bounded

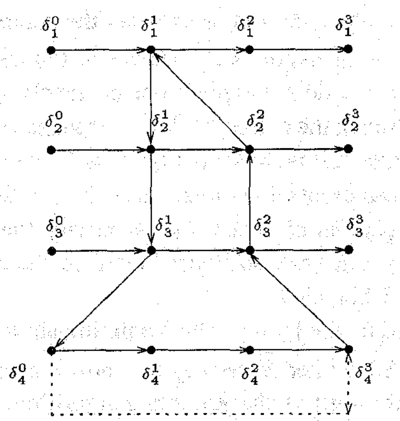

Figure 4. Precedence relation of Figure 3

5. An Adaptive Domino-Free Checkpointing Algorithm

In this section an uncoordinated domino-free checkpointing algorithm that follows an *adaptive approach* is designed. In such an approach processes take local checkpoints in an independent and arbitrary way. In consequence, according to dependencies on these uncoordinated local checkpoints, some of them can be useless. The adaptive checkpointing algorithm forces some processes to take additional local checkpoints in order that all the local checkpoints be useful.

The algorithm presented here adopts a purely local strategy: each process does not hold any information about the state of the other processes (*i.e.*, in a distributed context no control information is piggybacked on existing messages and no additional control messages are exchanged among processes) and local checkpoints are taken according to a rule (actually a regular expression defined on read and write events) which monitors local checkpointing. The rule followed by processes guarantees that no useless checkpoint are taken and, according to Lemma 4.1, ensures the domino-freedom property.

Let us now introduce the following property that shows which is the sequence of events that occurs in a checkpoint interval whenever there exists a useless checkpoint. It is clear that all local checkpoints are useful if no one of these sequences ever appear in any checkpoint interval.

Property 5.1 *Let $\widehat{H}$ be a computation and c_i^s be a local checkpoint belonging to $\mathcal{C}_{\widehat{H}}$. If c_i^s is useless then there exists a checkpoint interval δ_j^l in which the following sequence of events occurs: $\langle write\ read\rangle$*

Proof if c_i^s is a useless checkpoint (i.e., $\delta_i^s \prec \delta_i^{s-1}$), then $\exists j\ :\ \delta_i^s \prec \delta_j^l$ and $\delta_j^l \prec \delta_i^{s-1}$.

- As $\delta_j^l \prec \delta_i^{s-1}$ then, from Definition 3.1, there exists a sequence of read-from pairs of the relation *read-from*: $\rightarrow_{rf}^1, \rightarrow_{rf}^2, \ldots, \rightarrow_{rf}^q$ with $q \geq 1$ and denoted $(\rightarrow_{rf})_q$ such that $\delta_j^l \prec \delta_1^{x_1} \prec \ldots \delta_{q-1}^{x_{q-1}} \prec \delta_i^{s-1}$, a $write$ event denoted $w1$ belongs to δ_j^l, a $read$ event denoted $r1$ belongs to δ_i^{s-1} and $w1 \rightarrow_H r1$.

- As $\delta_i^s \prec \delta_j^l$ then, from Definition 3.1, there exists a sequence $(\rightarrow_{rf})_q$ with $q \geq 1$ such that $\delta_i^s \prec \delta_1^{x_1} \prec \ldots \delta_{q-1}^{x_{q-1}} \prec \delta_j^l$, a $read$ event denoted $r2$ belongs to δ_j^l, a $write$ event denoted $w2$ belongs to δ_i^s and $r2 \rightarrow_H w2$.

So, if there exists a useless checkpoint c_i^s then $r1 \rightarrow_l w2$ in process P_i and one of the following two sequences of events occurs in δ_j^l: $\langle write\ read\rangle$ (i.e., $\langle w1\ r2\rangle$) or $\langle read\ write\rangle$ (i.e., $\langle r2\ w1\rangle$).

If $\langle r2\ w1\rangle$ occurs in process P_l it follows that $r2 \rightarrow_l w1$. So we have $r2 \rightarrow_H w1 \rightarrow_H r1 \rightarrow_l w2 \rightarrow_H r2$ that violates the partial order property of $\widehat{H}$ (Definition 2.1). Then, the claim follows, namely $\langle w1\ r2\rangle$ in δ_j^l. $\square$

Hence, to guarantee that all local checkpoints be useful, each process must prevent the occurrence of this sequence of events in any checkpoint interval by properly selecting local states as local checkpoints. This behavior is described by the following regular expression employed by each process independently and defined over the alphabet formed by the set of events $\{checkpoint, write, read\}$ where "*checkpoint*" is an additional control event whose operational meaning is "dump the current local state onto stable storage":

$$[checkpoint\ read^*\ write^*]^*$$

In other words, if in a process, after a *write* event e_i^s, a *read* event e_i^{s+1} is being processed, the local state σ_i^{s+1} (corresponding to the local partial history $\widehat{h_i^s} = e_i^0 e_i^1 e_i^2 \dots e_i^s$) is dumped onto stable storage by generating a *checkpoint* control event that precedes e_i^{s+1}. So, no sequence of the type shown in Property 5.1 can ever belong to any checkpoint interval. Then, no useless checkpoint can ever be produced by the algorithm and this ensures domino-freeness.

A Remark on Rollback Recovery. If a failure occurs, the determination of a consistent global checkpoint from which the computation is resumed should be negotiated among the processes. It could be convenient to associate, on–the–fly, each local checkpoint with a global checkpoint to which it belongs. In this way, the failed process P_i reloads its last local checkpoint, say c_i^s, and forces each process to reload the corresponding local checkpoint belonging to the global checkpoint associated with c_i^s.

To this end, we add a vector $CKPT$ of size n on each process. $CKPT_i[j]$ stores 1+"the number of the greatest checkpoint interval number in which process P_j wrote a variable read by process P_i". Each time, P_i takes a local checkpoint, say c_i^s, the value of $CKPT_i$ is dumped onto stable storage in the vector $GLOBAL_i^s$ (with $GLOBAL_i[i]^s = s$). The global checkpoint $\mathcal{C} = \{c_j^{GLOBAL_i[j]^s}\}_{j\in\{1,\dots n\}}$ is consistent. Indeed, if $\mathcal{C}$ is non-consistent, there is either (i) an orphan messages between any pair of local checkpoint in $\mathcal{C}$ or (ii) $\mathcal{C}$ contains a useless checkpoint. Point (ii) is contradicted by the algorithm while point (i) by the definition of vectors $GLOBAL$ and $CKPT$.

6. Conclusion

In many shared memory systems built on parallel or distributed systems, checkpointing is one of the techniques to pursue backward recovery. It consists in restoring a failed computation in a consistent global state, from which this computation can be resumed. In this paper a formal definition of a computation in the shared memory model (including local and global checkpoint) apart from the underlying system has been given. Then, based on a necessary and sufficient condition, stating when an arbitrary set of local checkpoints can participate in global checkpoint, we have first derived a formal definition for the domino effect and, second, we have designed an adaptive and purely local (in a distributed context, no information is exchanged among processes) checkpointing algorithm that prevents the domino effect.

In the context of message passing systems, Russell [12] and Netzer and Xu [9] addressed similar problems. In particular, Russell proposed a domino-free checkpointing algorithm while Netzer and Xu derived a necessary and sufficient condition for mutual checkpoint consistency. Their results cannot be applied on shared memory systems as message passing imposes a one-to-one correspondence between send and receive events, while the shared memory model allows a one-to-any correspondence between write and read operations (including one-to-zero) [3].

References

[1] V. Abrossimov, M. Rozier. Generic Virtual Memory Management for Operating Systems Kernels. *Proc. 12th Symp. on Operating System Principles*, ACM Operating Systems Rev., 23(5):123–136, 1989.

[2] M. Ahamad, G. Neiger, J.E. Burns, P. Kohli, P.W. Hutto. Causal Memory: Definitions, Implementation and Programming. *Dist. Comp.*, 9(1):37–49, 1995.

[3] R. Baldoni, J.M. Helary, M. Raynal. About State Recording in Asynchronous Computations. *Proc. of 15th ACM Symposium on Principles of Distributed Computing*, Philadelphia, pp. 55, 1996.

[4] M. Choy, H.V. Leong, M.H. Wong. On Distributed Object Checkpointing and Recovery. *Proc. of 14th ACM Symp. on Princ. of Dist. Comp.*, Ottawa, 1995.

[5] L. Lamport. How to Make a Multiprocessor Computer that Correctly Executes Multiprocess Programs. *IEEE Trans. on Comp.*, C28(9):690-691, 1979.

[6] K. Li, P. Hudak. Memory Coherence in Shared Virtual Memory Systems Multiprocess Programs. *ACM Trans. on Comp. Syst.*, 7(4):321–359, 1989.

[7] J. Misra. Axioms for memory access in asynchronous hardware systems. *ACM Trans. on Programming Languages and Systems*, 8(1):142–153, 1986.

[8] M. Mizuno, M. Raynal, and J.Z. Zhou. Sequential Consistency in Distributed Systems. *Proc. Int. Workshop " Theory and Practice in Dist. Systems"*, Dagstuhl, Germany, LNCS 938:224–241, 1994.

[9] R.H.B. Netzer, J. Xu. Necessary and sufficient conditions for consistent global snapshots, *IEEE Trans. on Par. and Dist. Systems*, 6(2):165-169, 1995.

[10] M. Raynal, A. Schiper, From Causal Consistency to Sequential Consistency in Shared Memory Systems, *Proc. 15th Int. Conf. on Foundations of Software Technology & Theoretical Computer Science*, Bangalore, India, LNCS 1026:180–194, 1995.

[11] B. Randell. System structure for software fault-tolerance. *IEEE Transactions on Software Engineering*, SE1(2):220–232, 1975.

[12] D.L. Russell. State restoration in systems of communicating processes. *IEEE Transactions on Software Engineering*, SE-6(2):183–194, 1980.

Appendix

For simplicity's sake, in what follows we do not explicit objects involved in the *read-from* relations as it is not required from proofs. Let us first prove the following Lemma:

Lemma 6.8 *If $\delta_i^x \prec \delta_j^y$ and $i \neq j$ then there exists a sequence of distinct pairs of events $(write_r, read_r)_{r=1,\ldots,p} \in \rightarrow_g$, such that:*
1) $write_1 \in \delta_i^s, s \geq x; read_p \in \delta_j^t, t \leq y$; and
2) $\forall r : 1 \leq r \leq p-1 :: read_r \rightarrow_l write_{r+1}$ or the two events $read_r$, $write_{r+1}$ belong to the same interval.

Proof If $\delta_i^x \prec \delta_j^y$ holds, there is a sequence $\delta_i^x = \delta_{i_0}^{x_0} \prec \delta_{i_1}^{x_1} \prec \ldots \prec \delta_{i_q}^{x_q} = \delta_j^y$ (as $i \neq j$, then $q \geq 1$) where each pair $\delta_{i_k}^{x_k} \prec \delta_{i_{k+1}}^{x_{k+1}}$ is due to point 1. or 2. of Definition 3.1. Due to the possibility of cycles, the elements of this sequence are not necessarily distinct. However, with $q \geq 1$, a subsequence including only distinct intervals, starting at δ_i^x and ending at δ_j^y, can be extracted; so, we can always consider, without loss of generality, the case where the elements of this sequence are distinct. For each k $(0 \leq k \leq q-1)$ we have $\delta_{i_k}^{x_k} \prec \delta_{i_{k+1}}^{x_{k+1}}$, where $\prec$ is due to point 1. or 2. of the definition 3.1, i.e.:

- either $i_{k+1} = i_k$ and $x_{k+1} = x_k + 1$
- or $i_k \neq i_{k+1}$ and there is a pair of events $(e_{i_k}^s, e_{i_{k+1}}^t) \in \rightarrow_g$ where $e_{i_k}^s$ is a $write$ event belonging to $\delta_{i_k}^{x_k}$ and $e_{i_{k+1}}^t$ is a $read$ event belonging to $\delta_{i_{k+1}}^{x_{k+1}}$

These form a sequence of distinct pairs of events $(write_r, read_r)_{r=1,\ldots,p} \in \rightarrow_g$ satisfying the required property. □

Theorem 6.9 *Let $\mathcal{I} \subseteq \{1,\ldots,n\}$ and $\mathcal{R} = \{c_i^{x_i}\}_{i\in\mathcal{I}} \subseteq \mathcal{R}_{\widehat{C}}$ be a set of checkpoints of the asynchronous computation $\widehat{C}$. Then $\mathcal{R}$ is a subset of a consistent global checkpoint if and only if:*

$$(\mathcal{CT})\ \forall i, \forall j : i \in \mathcal{I}, j \in \mathcal{I} :: (x_j = 0) \vee (\neg(\delta_i^{x_i} \prec \delta_j^{x_j - 1}))$$

Proof (adapted from [9])
Sufficiency. We prove that if $\mathcal{CT}$ is satisfied then $\mathcal{R}$ can be included in a consistent global checkpoint. Let us consider the global checkpoint defined as follows:

- if $i \in \mathcal{I}$, we take $c_i^{x_i}$;
- if $i \notin \mathcal{I}$, for each $j \in \mathcal{I}$ we consider the integer $m_i(j)$ defined in the following way: let $PREC(j) = \{y \mid \delta_i^y \prec \delta_j^{x_j - 1}\}$ (such a set is finite, as a result from the remark following Definition 3.1).
  - if $PREC(j) = \emptyset$ then $m_i(j) = 0$ (note that this is the case in particular when $x_j = 0$).
  - otherwise, $m_i(j) = max(PREC(j)) + 1$

  Then we take $c_i^{x_i}$ with $x_i = \max_{j\in\mathcal{I}}(m_i(j))$. Remark that x_i could be equal to 0.

Thus,the global checkpoint $\{c_1^{x_1}, c_2^{x_2}, \ldots, c_n^{x_n}\}$ has, by definition, the following properties:

$\forall i \notin \mathcal{I}, \forall j \in \mathcal{I}$ ::

$$(x_j = 0) \vee \neg(\delta_i^{x_i} \prec \delta_j^{x_j - 1}) \qquad (1)$$

$\forall i \notin \mathcal{I}$ such that $x_i > 0, \exists k \in \mathcal{I}$ ::

$$(x_k > 0) \wedge (\delta_i^{x_i - 1} \prec \delta_k^{x_k - 1}) \qquad (2)$$

We show that $\{c_1^{x_1}, c_2^{x_2}, \ldots, c_n^{x_n}\}$ is consistent. Assume the contrary; there exists i and j and two events e_i^s, e_j^t such that $e_i^s \notin c_i^{x_i}$, $e_j^t \in c_j^{x_j}$ and $e_i^s \rightarrow_g e_j^t$ (*i.e.* e_j^t is an orphan read event with respect to these checkpoints) and thus, from point 2. of Definition 3.1, we have:

$$(x_j > 0) \wedge (\delta_i^{x_i} \prec \delta_j^{x_j - 1}) \qquad (3)$$

Four cases have to be considered:

1. $i \in \mathcal{I}, j \in \mathcal{I}$. Relation (3) contradicts $\mathcal{CT}$;
2. $i \in \mathcal{I}, j \notin \mathcal{I}$. Since $(x_j > 0)$ we have, from (2): $\exists k : k \in \mathcal{I} :: (x_k > 0) \wedge (\delta_j^{x_j - 1} \prec \delta_k^{x_k - 1})$.
   By transitivity (using Relation (3)) we have $\delta_i^{x_i} \prec \delta_k^{x_k - 1}$ which contradicts the assumption $\mathcal{CT}$;
3. $i \notin \mathcal{I}, j \in \mathcal{I}$. Relation (3) contradicts (1);
4. $i \notin \mathcal{I}, j \notin \mathcal{I}$. Since $(x_j > 0)$ we have, from (2): $\exists k : k \in \mathcal{I} :: (x_k > 0) \wedge (\delta_j^{x_j - 1} \prec \delta_k^{x_k - 1})$.
   By transitivity (using Relation (3)) we have $\exists k \in \mathcal{I} :: (x_k > 0) \wedge (\delta_i^{x_i} \prec \delta_k^{x_k - 1})$ which contradicts (1).

Necessity. We prove that if there is a consistent global checkpoint $\{c_1^{x_1}, c_2^{x_2}, \ldots, c_n^{x_n}\}$ including $\mathcal{R}$ then property $\mathcal{CT}$ holds for any $\mathcal{I} \subseteq \{1, \ldots, n\}$. Assume the contrary: there exist $i \in \mathcal{I}$ and $j \in \mathcal{I}$ such that $(x_j > 0) \wedge (\delta_i^{x_i} \prec \delta_j^{x_j - 1})$. From Lemma 6.8, there exists a sequence of distinct pairs $(write_r, read_r)_{r=1,\ldots,p} \in \rightarrow_g$ such that :

$$\begin{array}{lll} & write_1 \in \delta_i^s, s \geq x_i & \\ read_1 \in \delta_{i_1}^{y_1}, & write_2 \in \delta_{i_1}^{z_1} & \text{with } y_1 \leq z_1 \\ & \ldots & \\ read_{p-1} \in \delta_{i_{p-1}}^{y_{p-1}}, & write_p \in \delta_{i_{p-1}}^{z_{p-1}} & \text{with } y_{p-1} \leq z_{p-1} \\ read_p \in \delta_j^t, \ t \leq x_j - 1 & & \end{array}$$

We show by induction on p that $\forall t \geq x_j$, $c_i^{x_i}$ and c_j^t cannot belong to the same global checkpoint.

Base step. $p = 1$. In this case, $write_1 \in \delta_i^{x_i}$ and $read_1 \in \delta_j^{x_j - 1}$ thus the event $read_1$ is orphan in all ordered pairs $(r_i^{x_i}, r_j^t)$ with $t \geq x_j$.

Induction step. We suppose the result true for some $p \geq 1$ and show that it holds for $p + 1$. We have:

$$\begin{array}{lll} & write_1 \in \delta_i^s, s \geq x_i & \\ & \ldots & \\ read_p \in \delta_{i_p}^{y_p}, & write_{p+1} \in \delta_{i_p}^{z_p} & \text{with } y_p \leq z_p \\ read_{p+1} \in \delta_j^t, t \leq x_j - 1 & & \end{array}$$

From the assumption induction applied to the sequence of distinct pairs $(write_r, read_r)_{r=1,\ldots,p} \in \rightarrow_g$, we have : for any $t \geq y_p + 1$, $c_i^{x_i}$ and $c_{i_p}^t$ cannot belong to the same consistent global checkpoints. Moreover, $write_{p+1} \in \delta_{i_p}^{z_p}$ and $read_{p+1} \in \delta_j^t$ imply that, for any $u \leq z_p$ and for any $v \geq t + 1$, $c_{i_p}^u$ and c_j^v cannot belong to the same consistent global checkpoint. Since $y_p \leq z_p$, it follows that for any $u \leq y_p$, $c_{y_p}^u$ and c_j^v cannot belong to some consistent global checkpoint. Thus no local checkpoint in process P_{i_p} can be included with $c_i^{x_i}$ and $c_j^{x_j}$ to form a consistent global checkpoint. □