Session level rollback recovery

Augusto Ciuffoletti
Dipartimento di Informatica – Università di Pisa
Corso Italia 40
I-56100 Pisa (Italy)
e.mail: augusto@di.unipi.it
URL: http://www.di.unipi.it/∼augusto

Abstract

The problem of rollback recovery is traditionally approached using a model oriented to packet delivery. Instead, we introduce a model centered around complex sessions, and we explain why this model is more appropriate.

Using this model, we extract a basic coordination scheme that is common to the three distributed activities that compose rollback recovery: checkpointing, rollback and disposal. The basic coordination scheme is refined to describe each of the three activities.

Keywords: *distributed control, checkpointing algorithm, recovery algorithm, checkpoint disposal algorithm, quasi-synchronous coordination, consistent cut.*

1 Introduction

One aspect of the reliability of a complex system is its ability to recover from failures of its components. One way to implement this feature is *rollback recovery*: it basically consists in restoring a consistent computation, preserving partial results, after a crash.

The design of rollback recovery is based on some distributed knowledge of the history of the system; one model is universally adopted to describe such knowledge and we claim that it is not adequate for the purpose.

This model is based on the concept of *process*, a sequence of states that represents a local computation. Communication among processes is modeled using *messages*, that update the state of a (receiving) *process* using the state of a (sending) *process*.

The sequential organization of the states of a process, and the precedence between sending and receiving states, introduce partial orderings among states. If we aggregate them by transitive closure, we obtain a partial ordering that describes the history of the system: this structure is universally known as Lamport's *happened before* relation.

The irreflexivity of Lamport's relation is equivalent to the statement of the existence of a *timestamping function* for the states, which is a tautology in our physical world: the effect follows its cause. We say that a model based solely on such ordering is *un-structured*, since there is no assumption about the patterns in the communication, and *a-synchronous*, since the model does not introduce timing restrictions.

Such model is simple, since it uses well known concepts, and strong, since it does not introduce relevant assumptions: these are attractive features of a formal model. In fact, the *unstructured/asynchronous* model is universally adopted as the foundation of papers that address *quasi-synchronous* [7] checkpointing/recovery algorithms: just to mention the most recent issues, [9] and [6], that follow a scheme initiated by the "chase" protocol introduced by Merlin and Randell [8] in 1978.

Instead, we argue that the *asynchronous/unstructured* model is not adequate to support the design of rollback recovery.

Our first point is that strong models are not necessarily good foundations for effective designs: strong models need to be bound with realistic restrictions, that promote simple and effective designs. The final result is affected by the appropriateness of the restrictions more than from the strength of the basic model.

The *asynchronous/unstructured* model focusses on a simple communication mechanism: connectionless delivery of a single packet. However, we point out that behind each delivery there is the overhead of rollback recovery related activities.

Alvisi et al. [1] have experimentally investigated the performance of quasi-synchronous checkpointing algorithms that use the asynchronous/unstructured system model: such algorithms perform an analysis of system

history each time a packet is delivered, and record a new checkpoint when needed. The paper concludes that if the overhead introduced by the algorithm is expensive with respect to the cost of the packet delivery it is connected with, most of the advantages of the quasi-synchronous checkpointing approach are lost. Although quasi-synchronous protocols impose a limited computational overhead, according with [4], the balance is still unequal, and the authors indicate the need of scheduling the recording of the checkpoint concurrently to the checkpointed application, thus incurring further concurrency control problems and overhead.

In addition, the authors of [1] indicate as practically impossible to design a static checkpointing policy that optimizes the size of the checkpoints. Upon receipt of a message, the checkpointing policy may require the registration of a checkpoint: this request is neither predictable nor negotiable, since from its fulfillment depends the possibility of future rollbacks. An undesirable side effect is that the size of a checkpoint cannot be anticipated, since it may be requested when the recording of the internal state is more expensive: pending interrupts, opened files, unflushed buffers contribute in an impredictable way to the size of a checkpoint. The impredictability of the size of the checkpoint is a direct consequence of the absence of structure in the basic model: appropriate restrictions could make viable the task of limiting the size of the checkpoints.

The results in [1] support the impression that the asynchronous/unstructured model is unsuitable to represent the application framework of a rollback recovery algorithm. Our proposal is to focus on complex interactions, like those initiated by the *Session Initiation Protocol* (see RFC2543), whose cost is comparable to the overhead introduced by recovery related activities.

2 A session-based model

Our model extends that previously introduced in [3] and is based on the concept of *session*. This is the atomic computational step whose initial state we want to checkpoint in order to be able to recover from a failure that might occur later.

A *session* consists in the coordinated activity of a number of *participants*, the *processes*. A *session* is opened running an *initiation protocol* that entails a complex communication pattern: the final state of each participant depends on the initial state of all participants. For instance the participants might have to reach some sort of consensus, as in the case of the *Initial Sequence Numbers* in the TCP protocol (RFC793).

Unlike the *unstructured/asynchronous* model, our *session oriented* model introduces both a structure in the computation (the *session*), and a form of synchronization among participants (the *initiation protocol*). Based on this model, we are able to design a complete quasi-synchronous rollback recovery scheme, composed of checkpoint coordination, rollback control, and checkpoint disposal.

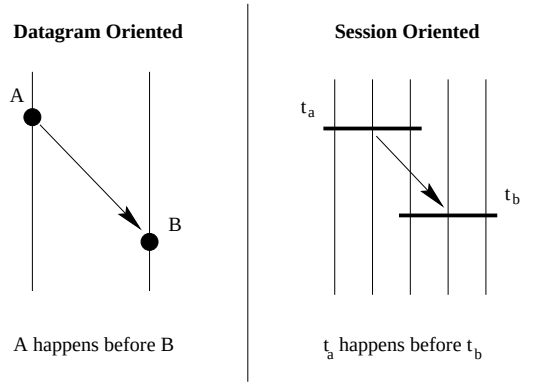

Figure 1. Comparison of ordering relations

Definition 1 *A* **process** $P = (S, <)$ *consists of set of* **states** $S = s_0, \ldots, s_n$ *and of a relation* $<$ *over* S*, whose transitive closure is a irreflexive total order. An* **event** *is a pair of states* $e = (s_a, s_b)$ *such that* $s_a < s_b$*.*

Definition 2 *A* **concurrent computation** $(S, <)$ *consists of a set of states* S *that can be partitioned in a family of sets* $\{S_i\}$ *such that each* $(S_i, <_i)$ *is a process.*

Definition 3 *The* **present** $\perp$ *and the* **origin** $\top$ *of a* concurrent computation $(S, <)$ *contain respectively the states that have no successor, and those that have no predecessor in the concurrent computation.*

Definition 4 *A* **session** t *is a set of events* $e_0, \ldots, e_n$ *in a concurrent computation* $(S, <)$*.*
We define pre$(t) \stackrel{\text{def}}{=} \{s \mid (s, s') \in t\}$ *and post*$(t) \stackrel{\text{def}}{=} \{s \mid (s', s) \in t\}$*.*

We have replaced the connectionless message exchange of the *unstructured/synchronous* model with an arbitrarily complex coordinated activity, the *session*. The graphical representations are compared in figure 1.

Definition 5 *A* **communication scheme** $H = ((S, <), T)$ *consists of a* concurrent computation $(S, <)$*, and of a set* T *of* sessions *that partition*[1] *the* events *in* $(S, <)$*.*

In figure 2 we give a graphical description of a communication scheme: vertical segments represent states,

[1] Note that, for the sake of uniformity, each *event* must be included in one *session*, and therefore local *events* are singleton *sessions*.

and horizontal segments represent sessions. Processes are represented by the alignment of vertical segments.

Definition 6 *Let* $((S,<),T)$ *be* communication scheme.

$$t_a \ll t_b \stackrel{\text{def}}{\Leftrightarrow} \exists s \mid s \in (\mathit{post}(t_a) \cap \mathit{pre}(t_b))$$

The transitive closure $\ll^*$ corresponds to Lamport's "happens before".

Definition 7 *A* **history** *is a* communication scheme $((S,<),T)$ *such that either*

- S *is the empty set or*
- $\exists t_f \in T \mid H' = ((S',<'),T')$ *where*

$$\begin{aligned} S' &\stackrel{\text{def}}{=} S/\mathit{post}(t_f) \\ s_a <' s_b &\stackrel{\text{def}}{\Leftrightarrow} (s_a, s_b \in S') \wedge (s_a < s_b) \\ T' &\stackrel{\text{def}}{=} T/\{t_f\} \end{aligned}$$

is an history.

Simply put, a *concurrent scheme* is an *history* if and only if we can find a session that can be removed, so that the remaining *concurrent scheme* is still an *history*. If we repeat this operation, assigning decreasing timestamps to the removed sessions, we obtain a timestamping for the sessions.

Lemma 1 *A communication scheme* $((S,<),T)$ *is an history if and only if the transitive closure of* $\ll$ *is irreflexive*[2].

which ensures that *histories* model all and only "real" computations.

The well-known concept of *consistent cut* is written as a property cc() that characterizes a set of *sessions* in an *history*:

Definition 8 *Let* T *be a subset of sessions in the history* H.

- $\triangle(H,T) = \{t \mid \exists t_c \in T, t_c \ll^* t\}$
- $cc(H,T) = \top(\triangle(H,T))$

The definition is sound since the $\triangle$ of any set of sessions in an history is an history.

We say that the sessions in T *trigger* the consistent cut. Figure 2 shows an history and three consistent cuts triggered by four sessions, t_a, t_b, t_c, t_d.

We can introduce a relation among consistent cuts, and a set of consistent cuts that are totally ordered by this relation:

[2]irreflexive means $\forall x, \neg(x \ll x)$.

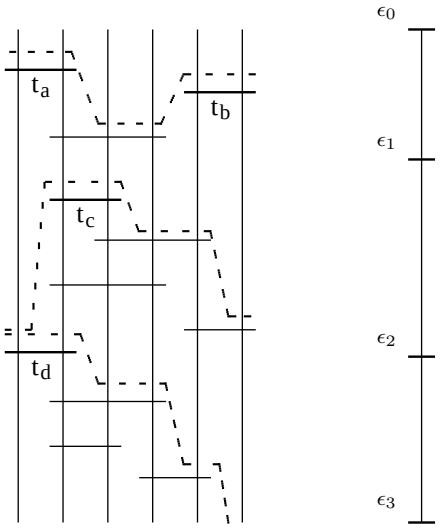

Figure 2. The layout generated by $T = \{t_a, t_b, t_c, t_d\}$ **is** $\mathcal{T} = (T_0 \ldots T_3)$ **with** $T_1 = \{t_a, t_b\}$**,** $T_2 = \{t_c\}$ **and** $T_2 = \{t_d\}$**. On the right is the corresponding global process.**

Definition 9 *Let H be a history.*

$$cc(H,T_x) \prec cc(H,T_y) \stackrel{\text{def}}{\Leftrightarrow} \triangle(H,T_y) \subseteq \triangle(H,T_x)$$

An **outline** $\mathcal{T}$ *of an history H is a series of sets of sessions* $(T_0 \ldots T_n)$ *such that*

$$\forall i \in [0 \ldots (n-1)], cc(H,T_i) \prec cc(H,T_{i+1}).$$

Using the $\prec$ relation, we can describe the concurrent operation of the processes as a sequential process: the *global process*. In that structure the consistent cuts take the place of the states. Figure 2 shows the global process associated with an history.

Definition 10 *A* **global process** *of an history* H *is a process* $\mathcal{P} = (\mathcal{T}, \prec)$*, where* $\mathcal{T} = (T_0 \ldots T_n)$ *is an* outline *of the history* H.
A **global event** *is a couple of consistent cuts* $\epsilon_i = (cc(H,T_i), cc(H,T_{i+1}))$.

2.1 Comparison with the unstructured/asynchronous model

The formal difference between the *unstructured/asynchronous* model, and our *session oriented* model, is that the former associates timestamps to local states, while the latter timestamps distributed activities, the *sessions*.

Our rollback recovery operates at session level, giving the system the ability to restore the checkpoints recorded by processes at the beginning of selected sessions, in order to recover from faults in following sessions. For this, we introduce some further overhead in the *initiation protocol*, which *per se* requires a communication overhead and a rich information exchange. We claim as appropriate to assume that participants reduce their internal state before entering a new session, thus reducing the size of a possible checkpoint.

Using the *session oriented* model the design of a complete rollback recovery scheme is approachable, as we see in the next session. The task is extremely more complex using the *unstructured/asynchronous* model, and the evidence for this is that papers using that model limit their scope to one of the three components, either checkpointing, or rollback, or checkpoint disposal, leaving open the problem of their integration.

3 Rollback recovery algorithm requirements

A quasi-synchronous rollback recovery algorithm is decomposed into three distinct tasks: **checkpointing**, to select the states that must be recorded as checkpoints, **rollback**, to trigger the restoration of the internal state of processes that experienced a failure, as well as of all those that have been possibly reached by the effects of the failure, and **checkpoint disposal**, to reclaim the resources allocated to the checkpoints when the application decides that they will not be used anymore.

These tasks are often considered as three distinct research areas, and for each of them there is an immense bibliography. The research concerning the *checkpoint* algorithms presently regards the quasi-synchronous approach as an interesting option: relevant references in this field are [6, 9, 7]. An experimental investigation about this approach is in [1].

The *rollback* algorithm is often left implicitly centralized (as in [6]), or refers to a different approach, that logs messages instead of recording checkpoints [2].

The research concerning the *disposal* of the resources allocated to useless checkpoints applies to a static, system-wide criteria: a checkpoint is useless when it is followed, on the same process, by another checkpoint belonging to a *global* recovery line (one that contains a checkpoint for each process in the system) [10] . This condition is far from being practical: i) the reference distributed system (i.e. the Internet) has a virtually unlimited number of processes, and ii) as a general rule the criteria to decide the disposal of a checkpoint are local: for instance, they may be based on the fact that an irrevocable event has been performed since the registration of the checkpoint.

Using a *session oriented* model, a rollback is a *global event* that is fed by the results of a diagnostic activity that identifies one or more faulty sessions. We assume that the diagnostics are known to a few units in the system, those that *trigger* the rollback. The *global rollback event* is the result of the coordinated execution of *local rollback events*, on each process that was exposed to the results of faulty sessions. A *local rollback event* consists in restoring a past state, previously recorded in a *checkpoint* recorded during a *local checkpointing event*. Each *local checkpointing event* must be in its turn coordinated with other similar local events in a *global checkpointing event*. The set of restored checkpoints is indicated as the *recovery line*.

A *global rollback event* must satisfy two consistency requirements: the effects of the faulty sessions do not affect the states in the recovery line, and the effects of the faulty sessions do not affect the states after the restoration of the local checkpoint. Using the session-based model:

Definition 11 *Let* $\epsilon_c = (cc(c), \_)$ *be a* global checkpointing event, *and* $\epsilon_r = (cc(r), \_)$ *a* global rollback event *to the* recovery line *represented by* $cc(c)$. *They form a consistent recovery if*

- *each local state in* $cc(r)$ *is identical to that in* $cc(c)$ *for the same process, and*
- $cc(c) \prec cc(r)$.

which requires that processes are able to detect when the current state (i.e., the initial state of the forthcoming session) is part of a *consistent cut*, and to obtain some information about it. In case the *consistent cut* corresponds to a *global checkpoint event*, this information is used to decide the recording of a checkpoint, and to associate it to the appropriate recovery line. In case the *consistent cut* corresponds to a global state restored by a *global rollback event*, the information about the *consistent cut* is used to decide whether a restore operation is needed, and to select the checkpoint. Finally, a given checkpoint can be flushed by the event following the state in the *consistent cut* associated to the *global disposal event*.

Therefore the coordination of checkpointing, rollback, and checkpoint disposal is based on the same protocol, that is executed during the *initiation protocol*.

3.1 The basic coordination scheme

This protocol detects that the current state of the process is part of a *consistent cut*, and, in this case, obtains a description of the associated *global event*. From this

description the process derives the description of a local action.

As a first step, we define two integer values that play a key role in the *basic coordination scheme*: the *level* ($\mathcal{L}$) and the *baselevel* ($\mathcal{B}$). These values are associated to a session, and can be computed by each process participating to that session during its *initiation*:

Definition 12 *Let T be a subset of the* sessions *in a* history H. *The functions $\mathcal{B}_T(t)$ and $\mathcal{L}_T(t)$ associate an integer value to a* session t *in the* history*:*

$$B_T(t) = \begin{cases} 0 & \textit{if } pre(t) \subseteq \top(H) \\ max\{\mathcal{L}_T(t') \mid t' \ll t\} & \textit{otherwise} \end{cases}$$

$$L_T(t) = \begin{cases} \mathcal{B}_T(t)+1 & \textit{if } t \in T, \\ \mathcal{B}_T(t) & \textit{otherwise} \end{cases}$$

The computation of the *level* should be added to the native *session initiation protocol*.

The first step consists in computing the maximum of the *levels* of the preceding *sessions*: it is a sort of consensus algorithm, and should be piggybacked to the *session initiation protocol* adding some piece of information needed to support the computation of the *baselevel*.

Next the participants decide whether to trigger a *consistent cut*. This decision depends on the specific application. For instance, in the case of *checkpointing*, participants might check that a checkpoint has been recorded from less than 60 seconds. In case the participants decide to record their state in a *consistent cut*, the *baselevel* computed in the previous step is incremented by one. This operation either triggers a new *consistent cut*, or includes the states in an already existing one.

The following theorem claims that, when a participant computes the *level* of the forthcoming session, it also identifies the *consistent cuts* that contain the current state. The theorem gives an effective algorithm to compute such identifiers, and concludes the design of the basic coordination scheme, illustrated in table 1.

Theorem 1 *Let T be a subset of the* sessions *in a* history H, *and $\mathcal{T} = (T_0 ... T_n)$ the corresponding* outline. *For each state s, let t be a session such that $s \in pre(t) \wedge s \in post(t')$*

$$s \in cc(H, T_i) \Leftrightarrow (\mathcal{L}_T(t) \geq i \wedge \mathcal{L}_T(t') < i)$$

4 Design of a rollback recovery protocol

Each of the three protocols of interest — checkpoint, rollback and dispose — run a separate instance of the basic coordination scheme. The internal coordination of the three global processes, as well as their interactions, is under control of the basic coordination scheme.

Require: $\mathcal{L}$ is the level of the last session (see definition 12);
1: the process computes $\mathcal{B}$ as the baselevel of this session;
2: **if** $\mathcal{B} > \mathcal{L}$ **then**
3: **for all** $i \in [\mathcal{L}+1, \mathcal{B}]$ **do**
4: the description of the *local event* associated with ϵ_i is obtained from the participants that already implemented it;
5: the *local event* associated with ϵ_i is implemented;
6: **end for**
7: **end if**
8: **if** the process decides, in cooperation with other participants to the session, to trigger a new *global event*, with an associated *local event* **then**
9: $\mathcal{B} = \mathcal{B} + 1$;
10: the local event associated with $\epsilon_{\mathcal{B}}$ is implemented;
11: **end if**
12: $\mathcal{L} = \mathcal{B}$;
Ensure: $\mathcal{L}$ is the level of this session;

Table 1. The basic coordination scheme

To specialize the basic scheme for a specific activity, we need to specify:

- which **local event** corresponds to the the global event;
- which are the **parameters**, if any;
- which are the requirements about the **participants** to the forthcoming session, if any.

In the following we outline three simple specifications.

4.1 The global checkpointing process

The specification is summarized in table 2.

We use the *level* as an identifier of the checkpoint: we do not need any further information to indicate a recovery line. This identification is used during the *checkpointing* activity, to attach a significant label to the recorded state, during the *rollback*, to indicate which checkpoint is to be restored, and during the *disposal*, to indicate which checkpoint is to be removed.

4.2 The global rollback process

The specification is summarized in table 3.

In order to enforce the relation introduced in definition 11 we need to guarantee that the *global checkpoint event* precedes the *global rollback event*. To this purpose, we require that a *global rollback event* to a given recovery line can be triggered only by a subset of the processes that triggered the *global checkpoint event* that

local event	record a checkpoint
parameters	none
participants	unbound

Table 2. Specification for the checkpointing process

local event	restore a checkpoint
parameters	the *level* of the recovery line
participants	some of the triggers of the recovery line + the serializer process

Table 3. Specification for the rollback process

local event	dispose a checkpoint
parameters	the *level* of the recovery line
participants	the triggers of the recovery line

Table 4. Specification for the disposal process

recorded that recovery line. They will engage in a specific *rollback triggering* session, and issue an indication of the recovery line to use.

In the above scenario, it is necessary to avoid concurrent rollbacks: one solution consists in the introduction of a centralized process (for instance the same that performs the diagnosis), that is in charge of serializing rollback operations. This process should be called to participate to every *rollback triggering* session, but does not store any data about them. Therefore its substitution, in case of failure, is straightforward.

Unlike the checkpointing, that is transparent to the session, the local rollback event may interfere with the forthcoming session. The application controlling the session should be made aware of this event, and undertake the appropriate actions: for instance, abort and replan the session.

4.3 The global disposal process

The specification is summarized in table 4.

If the *rollback* and *dispose* global processes run in reciprocal isolation, there could be the case that one process triggers a *global rollback event* to a recovery line that has been already disposed by other processes. In order to avoid this, we should introduce another consistency requirement, similar to that introduced between checkpointing and rollback *global events*: a *global dispose event* can be triggered only by a session whose participants are those that triggered the *global checkpoint event*.

5 Conclusions

The coordination of an efficient rollback recovery scheme is one of the most challenging problems in distributed computing, and the integration of its components — checkpoint, rollback and disposal — is still an open problem.

This paper suggests a new model, that introduces complex sessions instead of plain packet delivery, and a unifying concept, the *basic coordination scheme*: together, they simplify the design of integrated solutions.

We are currently coding the formal issues using the Coq [5] proof assistant.

References

[1] L. Alvisi, E. Elnozahy, S. Rao, S. A. Husain, and A. De Mel. An analysis of communication induced checkpointing. In *Proceedings of the 29th IEEE Fault-tolerant Computing Symposium (FTCS-29)*, pages 242–249, Madison (Wisconsin - USA), June 1999.

[2] L. Alvisi and K. Marzullo. Message logging: pessimistic, optimistic, optimal. *IEEE Transactions on Software Engineering*, 24(2):149–159, Feb. 1998.

[3] A. Ciuffoletti. Error recovery in a system of communicating processes. In *Proc. of 7th International Conference on Software Engineering*, pages 6–17, Orlando (Florida), Mar. 1984.

[4] M. Elnozahy, L. Alvisi, Y.-m. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. Technical Report CMU-CS-99-148, Carnegie Mellon University, June 1999.

[5] H. Gerard, G. Kahn, and P.-M. Christine. The Coq proof assistant – a tutorial. Technical report, INRIA - Coq Project, 2000.

[6] J.-M. Helary, A. Mostefaoui, R. H. Netzer, and M. Raynal. Communication-based rpevention of useless checkpoints in distributed computations. *Distributed Computing*, 13:29–43, 2000.

[7] D. Manivannan and M. Singhal. Quasi-synchronous checkpointing: Models, characterization, and classification. *IEEE Transactions on Parallel and Distributed Systems*, 10(7), 1999.

[8] P. Merlin and B. Randell. State restoration in distributed systems. In *Proc. of International Symposium on Fault-Tolerant Computing*, 1978.

[9] J. Tsai, S.-Y. Kuo, and Y.-M. Wang. Theoretical analysis for communication-induced checkpointing protocols with rollback dependency trackability. *IEEE Transactions on Parallel and Distributed Systems*, 9(10):963–971, Oct. 1998.

[10] Y. Wang, P. Chung, I. Lin, and W. Fuchs. Checkpoint space reclamation for uncoordinated checkpointing in message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 6(5):546–554, 1995.