Causal Ordering of Asynchronous Request Services

Christian Delbé[†]
INRIA - CNRS - Univ. Nice-Sophia Antipolis
2004, Route des Lucioles - BP93 - 06902 Sophia Antipolis Cedex France
Email: Christian.Delbe@inria.fr

***Abstract*— This paper presents a study that will allow us to design and prove correctness of fault-tolerance protocols for a particular middleware model called ASP. This study is based on Lamport's happened-before relation and its extension by Mattern. It takes into account specific relations between events in an asynchronous request/reply model.**

We introduce a *partial local order* between events that occur in a single process, and define a minimal characterization of an execution. After a definition of an ASP-convenient local order, we define *consistent enough cuts* in this context.

I. Introduction

Lamport introduces in [1] the concept of one event happening before another in a distributed system, and defines a partial ordering between events. His seminal paper is the first to emphasize the "causal domain", in opposition to the "time domain": notion of time and simultaneity are avoided. Mattern presents in [2] a formal model for characterizing a distributed execution. It consists in a set of events partially ordered by the Lamport's happened-before relation. This characterization is an important foundation for specifying and proving distributed programs, and particularly for designing and proving correctness of fault-tolerance protocols [3]. Indeed, proving rollback-recovery protocols needs a way to characterize distributed executions and states.

The Lamport's happened-before relation supposes that events that occur on the same process are *totally ordered*. Based on such local events ordering, Lamport defines a partial global order that also takes into account synchronization due to message passing between processes. In the context of asynchronous request/reply communication patterns, such total ordering of local events is too much restrictive: the asynchronous service of requests allows to safely exchange request receptions with some other events.

In this paper, we take into account the kind of events in the causality relation. A first distinction between message reception events is drawn in [4]: the authors propose an algorithm for identifying messages that are not influential in a computation, and then provide a message logging recovery scheme that takes into account message semantic. The same distinction is used in [5] to define the *significant precedence order*, and to provide a group protocol that supports the significantly ordered delivery of messages.

We add a concept similar to significant precedence into a framework based on asynchronous request services. ASP (Asynchronous Sequential Process) [6] defines a convenient model of such framework. Our main contribution is an event-based formalization and properties for designing and proving fault tolerance protocols. In the next section, we present the relevant aspects of the ASP calculus, and show why we introduce a partial order between events that occur on a single process. The section III show how a distributed execution can be then characterized. The section IV gives a convenient relation for ASP, and introduce *consistent enough cuts*.

[†]Join work with Ludovic Henrio, Denis Caromel and Françoise Baude.

II. Asynchronous request service model

We present in this section a quick overview of the ASP calculus. We consider mono-threaded activities (processes) communicating with an asynchronous request and reply mechanism. A request from an activity i to another activity j consists in adding an entry in the pending requests queue of the activity i. Requests are *served* by the destination activity: the destination activity is responsible for reacting later to this request and associate a result to be returned to the sender This mechanism is called asynchronous because this request will be served later by j and meanwhile i can continue its execution while it does not need the result of the request service. Later on, j will serve this request and send back to i the result.

In this paper, we focus only on the request service mechanism because it has been proved in [6] that, for the ASP model, the reply mechanism has an important property stating that replies can be sent in any order without any consequence on the execution. We will then consider four kinds of events: request sending, request reception, request service and internal operation. An internal operation is an event that manipulates only the internal state of an activity (it does not alter the pending request queue), and a request service event is the beginning of the service of a request.

The asynchronous service of requests introduces particular relations between events on a single process: a request reception event can be safely exchanged with an internal event, a request or reply sending, or a service of another request. The consequence is that a computation characterized using the Lamport's happened-before relation is too restrictive: the relative order of some events that occur on the same process is not relevant to characterize a distributed computation.

The fact that the happened-before relation defines a total order on the events inside each process makes it inadequate to an asynchronous request service model. So as to characterize computations in ASP, we consider a causality relation inside an activity (called *local causality relation*) that uniquely defines a *partial order* on local events.

III. A PARTIAL LOCAL EVENTS ORDER

This section first presents a model for characterizing distributed executions, then a partial order on local events is introduced in order to express only the minimal causality relation between events. Finally, this partial causality order is used to obtain a minimal representation for characterizing an execution.

In the following, i, j, k range over the activities of the system; e_i is an event that occurs on the activity i.

A. Characterizing Executions

Let $\mathcal{S}_i^k$ be a state of the activity i. Let us consider a distributed execution characterized by a set of lists of events $\mathbb{E} = \{E_0, ...E_n\}$. This execution allows the system to change from the global state $\mathcal{S}_0^{ini}|...|\mathcal{S}_n^{ini}$ to the global state $\mathcal{S}_0^{final}|...|\mathcal{S}_n^{final}$:

$$\mathcal{S}_0^{ini}|...|\mathcal{S}_n^{ini} \xrightarrow{\mathbb{E}=E_0...E_n} \mathcal{S}_0^{final}|...|\mathcal{S}_n^{final}$$

where each E_i occurs on activity i:

Notation 3.1 (local execution): Let E_i be a (totally ordered) list of events that occur on an activity i:

$$E_i = [e_i^1, ..., e_i^m]$$

We suppose that, in each activity, such sets of events are sufficient to characterize the local computation in the following manner:

Notation 3.2 (characterization of local execution): We denote by $+$ the local execution simulated only from the local computations E_i in the following manner:

$$\mathcal{S}_i^{ini} + E_i = \mathcal{S}_i^{final}$$

We suppose that this definition is sound, in other words $\mathcal{S}_i^{final}$ is unique in Notation 3.2. In the following, we will only write $S_i + E_i$ if there exists a distributed execution for which local execution from state S_i is characterized by E_i.

B. Partial Local Order

The precedence order is a local order corresponding to the local causality ordering as defined in Lamport relation:

Notation 3.3 (Precedence order): $e_i^x \rightarrow_i e_i^y$ for a computation $E_i = [e_i^1, ..., e_i^m]$ if and only if $x < y$.

Let σ denote permutations of events in local computations. In the following we will only consider permutations of computations that preserves a given order $\prec_i$:

$$e_i^x \prec_i e_i^y \Rightarrow \sigma(e_i^x) \prec_i \sigma(e_i^y)$$

We introduce a local causality relation $\prec_i$ characterized as follow:

Definition 1 (local causality relation): $\prec_i$ is a local causality relation if and only if it verifies:

$\forall\sigma$, if $e_i^x \prec_i e_i^y \Rightarrow \sigma(e_i^x) \prec_i \sigma(e_i^y)$, then

$$\left(\mathcal{S}_i^0 + E_i = \mathcal{S}_i^1 \wedge \mathcal{S}_i^0 + \sigma(E_i) = \mathcal{S}_i^2\right) \Rightarrow \mathcal{S}_i^1 = \mathcal{S}_i^2$$

In other words, all permutations of events preserving the order define by $\prec_i$ correspond to equivalent executions.

C. Minimal Characterization of Local Execution

Consequently, a local execution is no longer characterized by a totally ordered set of events but only by a partially ordered set, denoted by $(\mathcal{E}_i, \prec_i)$ where $\mathcal{E}_i$ is a set of events and $\prec_i$ a local causal order on these events.

Notation 3.4 (generalized local execution): $(\mathcal{E}_i, \prec_i)$ generalizes a given execution E_i if and only if

$$\begin{cases} e \in E_i \Leftrightarrow e \in \mathcal{E}_i \\ e \prec_i e' \Rightarrow e \rightarrow_i e' \end{cases}$$

In other words, a correct generalized local execution $(\mathcal{E}_i, \prec_i)$ is made of contiguous events (elements of $\mathcal{E}_i$) causally ordered by $\prec_i$. Then a minimal characterization of execution is the following one:

Notation 3.5 (minimal characterization of local execution):

$$S_i \oplus (\mathcal{E}_i, \prec_i) = S_i' \Leftrightarrow \begin{cases} E_i = [e^1 \dots e^k] \wedge \\ \mathcal{E}_i = \{e^l | 1 < l < k\} \wedge \\ e^l \prec_i e^m \Rightarrow l < m \wedge \\ S_i' = S_i + E_i \end{cases}$$

Indeed, in this definition, S_i' is unique because $\prec_i$ is causally correct (Definition 1).

IV. A CAUSALITY RELATION FOR ASYNCHRONOUS REQUEST SERVICES

This section applies the preceding results to the ASP calculus, and particularly to the specification of consistent global states. This is considered as a first step towards proving the correctness of a rollback recovery fault tolerance protocol.

Let M denote a request message, e_i is an event that occurs on the process i, following the classification of events given in II (request sending, request reception, request service and internal operation), we obtain:

$$e_i ::= \mathit{send}(M) | \mathit{rcv}(M) | \mathit{serve}(M) | \mathit{int}$$

Let Γ denote the corresponding pairs of communication events, Γ defines a bijection between request sending and request receptions:

$$\Gamma = \{(e_i^s, e_j^r) | i \neq j \wedge \exists M, e_i^s = \mathit{send}(M) \wedge e_j^r = \mathit{rcv}(M)\}$$

Let Σ denote the pairs associating request receptions with their services, Σ defines a bijection between request receptions and request services:

$$\Sigma = \{(e_i^r, e_i^x) | e_i^r \rightarrow_i e_i^x \wedge \exists M, e_i^r = \mathit{rcv}(M) \wedge e_i^x = \mathit{serve}(M)\}$$

A. Characterizing a request/service execution

In ASP a convenient local causal order is:

Definition 2: $\prec_i$ is a transitive *partial order* such that $e_i^x \prec_i e_i^y$ if and only if

$$\begin{cases} (e_i^x \neq rcv \wedge e_i^y \neq rcv \wedge e_i^x \rightarrow_i e_i^y) \vee & (a) \\ (e_i^x = rcv \wedge e_i^y = rcv \wedge e_i^x \rightarrow_i e_i^y) \vee & (b) \\ (e_i^x, e_i^y) \in \Sigma i \vee & (c) \\ \exists e_i, e_i^x \prec_i e_i \prec_i e_i^y & (d) \end{cases}$$

This causality relation expresses three kinds of causes:

(a) *Evaluation order*, that is the order directly given by the local computation. It is composed of internal, request sending and service events. Provided that activities are piecewise deterministic [7], such order does not need to be remembered to characterize execution because it is a consequence of the evaluation mechanism. This is mainly due to the fact that each activity is made of a single thread.
(b) *External event order*, that is the order of message reception. This order must be remembered if one wants to replay an execution: some events can occur in a different order during the replayed execution (e.g. reception of requests coming from independent activities) or always occur in a predefined order (e.g. reception of two requests coming from the same source activity).
(c) *Service order*, that is the order relating the reception and the service of a request.

The definition of the global partial order, as formalized by Lamport, is unchanged. But since it is based on a partial local order $\prec_i$, its signification is altered. This global relation is still defined as follow:

Definition 3 (partial global order): $\prec$ is a quasi ordering such that $e_i^x \prec e_j^y$ if and only if

$$\begin{cases} e_i^x \prec_i e_i^y \vee \\ (e_i^x, e_j^y) \in \Gamma \vee \\ \exists e_i, e_i^x \prec e_i \prec e_i^y \end{cases}$$

B. Consistent Enough Cuts

As defined in [2], a cut is a partially ordered set defined as follow:

Definition 4: A cut $\mathcal{C}$ of an event set $\mathbb{E}$ is a finite subset $\mathcal{C} \subseteq \mathbb{E}$ such that

$$e \in \mathcal{C} \wedge e' \prec_i e \Rightarrow e' \in \mathcal{C}$$

Checkpointing protocols are usually based on consistent cuts. A cut is consistent if it verifies:

Definition 5: A cut $\mathcal{C}$ is consistent if and only if

$$e \in \mathcal{C} \wedge e' \prec e \Rightarrow e' \in \mathcal{C}$$

The partial local order given in IV-A defines a more permissive notion of consistent cuts. Moreover, an interesting property of ASP is that requests can be safely added or removed from the pending request queue. Thus making a cut consistent can be achieved by adding or removing messages in the pending request queue but modifying the internal state of an activity is not possible. Consequently, a cut will be said to be *consistent enough* if it can be transformed into a consistent cut. That means that a cut is consistent enough if there is no served orphan request.

Definition 6 (consistent enough): A cut $\mathcal{C}$ is consistent enough if and only if

$$(e_i^x, e_i^y) \in \Sigma \wedge (e_j^z, e_i^x) \in \Gamma \wedge e_j^z \notin \mathcal{C} \Rightarrow e_i^y \notin \mathcal{C}$$

Such a consistency property allows to define new kinds of consistent global states from which a recovery could be performed. This is particularly useful because it allows much more flexibility in the placement of checkpoints: checkpoints synchronizations (i.e. checkpoints forced by a message reception) could then be delayed while the communication that should trigger the checkpoint *has no consequence on the internal state of the activity*.

V. Conclusion

This paper introduced a partial local order into the Lamport's happened-before relation, which leaded us to a minimal characterization of a distributed execution. The concept of consistent enough cuts generalizes, for asynchronous requests and replies models, the classical consistency of cuts.

Concerning replies, it has been proved in [6] that the order of reply receptions has no influence on the execution. Thus, the only causality relation concerning replies is that using a result necessarily happens *after* the reception of the corresponding reply. Adapting the preceding framework mainly relies on this remark and should not raise any technical difficulty. Indeed dealing with such out of order replies in a fault tolerance protocol does not require much causality informations.

This framework has been designed in order to prove the correctness of fault-tolerance protocols for ASP. However the partial events ordering introduced in this paper applies as soon as a message reception has a delayed consequence on the internal state of the activity (e.g. asynchronous RPC). Indeed this work could be applied to other systems as soon as a model provides the semantic of a message and particularly the first local consequence of a message reception. For example, such properties also seem easily identifiable in the context of a distributed shared memory where nodes communicate with asynchronous read and write messages.

Consequently, generalizing consistent enough cuts by taking into account a generic partial causality relation seems to be a promising perspective.

References

[1] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," in *Communications of the ACM*, vol. 21, no. 7, July 1978, pp. 558–565.

[2] F. Mattern, "Virtual time and global states of distributed systems," in *Parallel and Distributed Algorithms: proceedings of the International Workshop on Parallel And Distributed Algorithms*, M. C. et. al., Ed. Elsevier Science Publishers B. V., 1989, pp. 215–226.

[3] M. Elnozahy, L. Alvisi, Y. Wang, and D. Johnson, "A survey of rollback-recovery protocols in message passing systems," School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, Tech. Rep. CMU-CS-96-181, oct 1996.

[4] H. V. Leong and D. Agrawal, "Using message semantics to reduce rollback in optimistic message logging recovery schemes," in *International Conference on Distributed Computing Systems*, 1994, pp. 227–234.

[5] T. Enokido, H. Higaki, and M. Takizawa, "Significant message precedence in object-based systems," in *ICPADS*, 1998, pp. 284–291.

[6] D. Caromel, L. Henrio, and B. Serpette, "Asynchronous and deterministic objects," in *Proceedings of the 31st ACM Symposium on Principles of Programming Languages*. ACM Press, 2004.

[7] R. Strom and S. Yemini, "Optimistic recovery in distributed systems," in *ACM Transactions on Computer Systems*, vol. 3, 1985, pp. 204–226.