Causality in Autonomous Mobile Systems

Carlos Baquero
cbm@di.uminho.pt

Francisco Moura
fsm@di.uminho.pt

Distributed Systems Group
Computer Science Department
Minho University, Braga, Portugal
http://gsd.di.uminho.pt/

Abstract

The analysis of causal relations among events in a distributed computation plays a central role in distributed systems modeling. Existing models for causal time-stamping are based on a known set of entities, either processes or data repositories, where events can occur. The advent of mobile computing settings that stimulate cooperation among arbitrary groups of nodes, possibly in isolation, precludes the use of a pre-established set of entity identifiers. Addressing this problem, the article proposes a causality definition and a time-stamping model that allows the analysis of those environments, while retaining compatibility with the classic causality model.

1 Introduction

Distributed computations are often modeled as a set of concurrent activities. These activities are distributed among computing nodes and perform communication by message passing over a communication network. In such systems physical time can only be approximated and potential causality plays a fundamental role in the analysis and understanding of these computations[12, 13].

Recently, mobile systems have emerged as a distinctive kind of distributed systems raising demand for the adaptation of distributed systems theory and techniques. When compared to stationary nodes served by a local network, mobile nodes exhibit different communication patterns and availability, often experiencing long periods of disconnection. Mobility itself can lead to changes on the cardinality and identity of the activities involved in distributed computations.

Taken together, these factors will influence the determination of causality in mobile environments, raising the need for an adaptation of the existing techniques. This article addresses this issue and presents a causality logging mechanism that fits environments where the number of concurrent activities is unbound, and the lifespan of each process is often inferior to that of the distributed computation.

The next section will summarize the present techniques for causality logging and reference some recent adaptations that fit mobile networks. Building on these notions, a following section will introduce a more general mechanism that addresses the concerns that were raised by the introduction of mobile nodes. Sections 4 and 5 present a model of autonomous causality and the associated time-stamping scheme. Section 6 closes the article with conclusions.

2 Causality Modeling

2.1 in Distributed Systems

Following the definition of causality as a partial order among events [6], logical vector time-stamping techniques have been proposed [3, 13, 8] for an efficient description of this partial order. These representations have been shown to provide a correct modeling of the partial order that was initially defined by Lamport as a *happened-before* relation among primitive events [6, 13].

Definition 2.1 *Lamport causality* $\overset{\text{lam}}{\rightarrow}$ *is defined as the smallest transitive relation on the set of events* E, $\overset{\text{lam}}{\rightarrow} \subseteq E \times E$, *satisfying:*

- $e_{ai} \overset{\text{lam}}{\rightarrow} e_{aj}$, *if* $e_{ai}, e_{aj} \in E_a$ *occur in the same process* P_a *and* e_{ai} *occurs before* e_{aj}, *verifying* $i < j$.
- $e_{ai} \overset{\text{lam}}{\rightarrow} e_{bj}$, *if* $e_{ai} \in E_a$ *is a send event and* $e_{bj} \in E_b$ *the corresponding receive event.*

From this definition it can be seen that in Lamport causality it is assumed that events on the same process are related in a total order. This assumption, which is crucial for the validity of description techniques based on vector time-stamping, is reasonable for the majority of the modeled systems. Nevertheless, some systems do not exhibit a total ordering among events in the same activity. In such cases, stronger description techniques based on causal histories [13, 4] are required and vector time-stamping may no longer be sufficient.

2.2 in Mobile Systems

Recently, the need to support efficient causality logging, in mobile networks with a mix of fixed support stations and mobile nodes, called for the re-evaluation of causality logging for activities exhibiting non total ordering [10, 1]. In these configurations, space saving and reliability issues suggest the use of partial ordering techniques for the delegation of logging tasks from the mobile nodes to their allocated support stations.

Although mobile networks, with support stations and mobile nodes, do represent a major portion of current mobile systems architectures, it is now clear that a new trend favoring direct interaction among mobile nodes is rapidly emerging. The leading factors are: the increased ubiquity of mobile devices capable or seaming-less inter-communication; the standardization efforts towards close range radio communication among mobile devices [7, 5]; and the development of data sharing protocols for point to point collaboration [2].

The new class of mobile systems that comes with this trend calls for a new approach to causality modeling. An approach that copes with the increased dynamism and unpredictability of interaction peers and the short lifespan of each individual activity.

3 A Data Driven View

3.1 Modeling Process Causality

When logging a distributed computation performed by a set of processes, causality connections can be introduced both by internal events local to a process, and by send/receive event pairs.

The modeled causality in the first case is bound to the occurrence of events within the processes. It is expected that these events induce changes in the local state of the process, thus influencing each other.

On the second case, causality connections are due to communication. When send and receive events perform a communication act among two processes, the causality link between the two separate process states is performed by way of a data item that is communicated. This data item can be seen as a transient portion of state that is separate from the local state of the processes.

In both cases, the description can be adapted in order to express causality connections by relating portions of data (state and messages) that exhibit causal dependencies. Under this vision, we can introduce an alternative picture of internal events and send/receive event pairs.

Tagging by $\mathcal{P}$ the state instance that represents a process in a given point in time, and by $\mathcal{M}$ an arbitrary message, we can observe three patterns that respectively represent the occurrence of internal, send and receive events.

This representation helps to enlighten some characteristics of process causality modeling. For instance, the fact that all three transformations must operate over a given $\mathcal{P}$ shows that the number of processes that will engage in the distributed computation must be known *a priori*[1]. In fact most causality models assume that the appropriate number of processes exist, and do not incorporate explicit primitives for processes creation and destruction [13]. The lack of these primitives invalidates the two associated patterns.

[1]This is sometimes circumvented by replacing fixed size vector time-stamps by associative arrays. In such cases it is the set of process names that is fixed.

We can also observe that messages are immutable and cannot be combined or duplicated, thus prohibiting three additional patterns.

Before delving deeper into a more data driven vision of causality it should be recalled that a duality of process causality and replica versioning is taken into account. In fact the comparable concepts of vector clocks[3, 13] and version vectors [8] have evolved together as representations of causality. Both exhibit the same dependence on the identity and number of the modeled units (that in our description we have restricted to processes), and both can be expected to benefit from the evolution that we are about to describe.

3.2 Towards Autonomous Operation

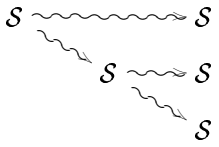
The set of transitions allowed in process causality have been adequate for the distributed computations that build on a set of communicating processes. A fixed number of processes enables the use of the indexed integer vectors that base vector time-stamping schemes. Even on environments where the number of participating processes changes with time, it is often possible to uniquely identify each process so that an associative array from process identifiers to integers can be used as a base for vector time-stamps [11].

Nevertheless, we argue that, on environments of extreme mobility and free interaction among mobile nodes, these approaches are no longer adequate. Several factors concur to the validity of this statement:

- The number of participating instances is often impossible to determine and even impossible to uniquely identify in an uniform way. This is the case when we allow the creation of new participants by duplication in isolation. Such actions are needed if collaboration with unexpected peers is to be supported.
- The distinction between messages and process state is fuzzier in these environments, thus suggesting their eventual unification into a single concept. For example, in the IETF *iCalendar*[2] specification, the structure of an event scheduling message and the persistent event itself are alike.
- The adoption of asynchronous communication schemes that are implemented with persistent messages and make use of epidemic diffusion raises the opportunity to replicate and combine the actual messages. Consequently, causality dependencies among messages becomes relevant.

In order to address causality modeling in these environments, the causality connections are better described by linking dependent portions of data. The patterns that are relevant in such a model show the occurrence of internal events that change a state, events that derive new instances of a state, and events that combine them.

By restricting the multiplication and combination actions to pairs the model can be made simpler, without loss of generality, since it will still be possible to model multi-part links by combining the basic links.



The model of causality that is presented in the next section formalizes this viewpoint, introducing a vision of causality that is centered on the state. This vision should be able to generalize the classical process causality[3] and support the analysis of autonomous mobile systems.

[2] See: IETF Calendaring and Scheduling Working Group at `http://www.imc.org/ietf-calendar/`

[3] As defined by Lamport[6].

4 Autonomous Causality

Causality is modeled over a set of instances $S = \{S_a, S_b, \dots\}$ of which a subset will be active at any specific point in time. We consider the occurrence of internal events in a given instance, as well as duplication and combination events that change the set of active instances. Internal events can be sequentially indexed in the appropriate instance as they occur in a total order.

For each instance $S_x \in S$, we represent by E_x the set of events that occur in it. This is a totally indexed set, as depicted by $E_x = \{e_{x1}, e_{x2}, \dots\}$. The set of system events E gathers all the events that occur in the instances in S, $E = E_a \cup E_b \cup \cdots$.

Duplication and combination events are respectively allocated as the last event of the duplicated instance and as the initial events of the combined instance. The first and the last event of a given instance S_x are conveniently denoted as e_{x1} and $e_{x\multimap}$.

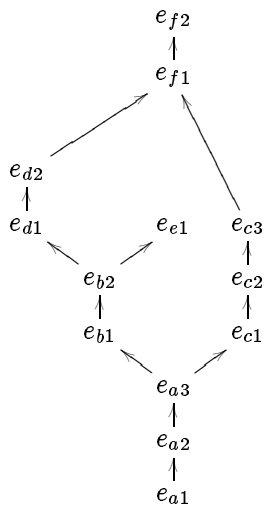


Figure 1: A partial order among events, depicting causality.

Definition 4.1 *The causality relation $\rightarrow$ is defined as the smallest transitive relation on the set E of events, $\rightarrow \subseteq E \times E$, satisfying:*

- *$e_{xi} \rightarrow e_{xj}$, if $e_{xi}, e_{xj} \in E_x$ occur in the same instance S_x and e_{xi} occurs before e_{xj}, thus verifying $i < j$.*
- *$e_{x\multimap} \rightarrow e_{y1}$ if $e_{x\multimap} \in E_x$ and $e_{y1} \in E_y$, with E_x, E_y as the events of S_x, S_y respectively, and one of the following conditions holds:*
  - *$e_{x\multimap}$ is a duplication event and S_y is created from S_x by that event.*
  - *e_{y1} is a combination event and S_y is created from S_x, and from another instance of S, by that event.*

In Figure 1 we show a set of related events, that exemplify this partial order of causality.

As usual, two events are concurrent when they are not causally related.

Definition 4.2 *The concurrency relation, $\nrightarrow \subseteq E \times E$, among distinct events from E is defined as: $e \nrightarrow e'$ if and only if $\neg(e \rightarrow e')$ and $\neg(e' \rightarrow e)$.*

The expressiveness of this causality model allows the representation of process causality, as defined by Lamport, and shown in definition 2.1. Informally, an encoding could be obtained by the following set of transformations: Both processes and messages are represented by generic instances; Internal events are simply registered in the associated instance; Send events lead to a duplication that yields two instances, one associated to the sending process and the other to the message; Likewise, receive events combine the instance of the message and that of the receiving process and derive a new process instance.

This expressiveness has a price, and it comes to face when developing time-stamping schemes — the significant point is the absence of pre-defined unique identifiers for tagging the instances. The identification problem will lead, as shown in the next section, to the use of a composition technique that uses the available identifiers when a new instance is to be created. In fact, similar composition techniques are also needed in actual systems that resort to epidemic propagation [9], when it comes to the autonomous creation of unique identifiers.

Even though the associated time-stamping technique would prove to be overly expensive for modeling process causality, it embodies the necessary expressiveness for systems that exhibit autonomous mobility.

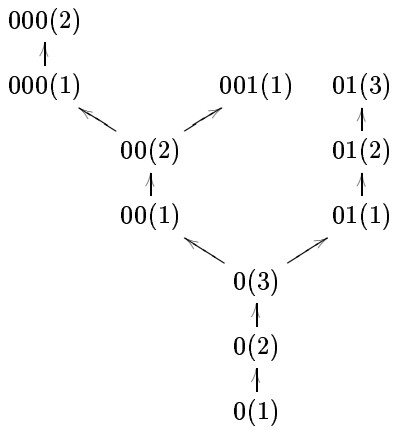


Figure 2: Event time-stamps with duplication only. Each label tags an event with the instance identifier followed by the event sequence number in braces.

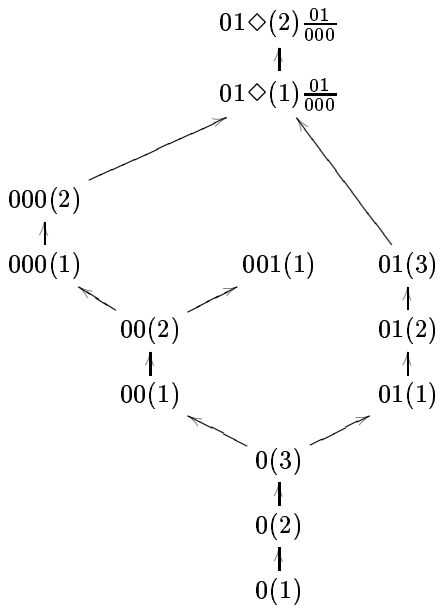


Figure 3: Event time-stamping with two bifurcations and one combination.

5 Time-stamping scheme for Autonomous Causality

The definition of adequate time-stamps for the implementation of this causality relation must start with the selection of adequate instance identifiers. These identifiers are only bound to the represented data instance and must be independent of the computing node that might be operating on that instance. By fulfilling these requirements, those identifiers will be able to persist with the instance, track causality links that origin in data interchange by transportable media, and even allow the creation of new dependencies while data is in dissemination.

Fitting these requirements, the selected identifiers can be constructed as incremental sequences of three alternative symbols. Representing the set of three symbols by $\mathbf{3}$, with the elements $\{0, 1, \diamond\}$, the possible sequences from this set can be represented by $\mathbf{3}^*$. The use of these elements is due to the fact that upon duplication two alternatives need to be distinguished, and that an additional distinction is needed in order to represent combinations.

In fact, if only duplications were to be considered, it would suffice to use the symbols $0, 1$ and a natural counter to tag the total oder of events in each instance. This example is shown in Figure 2.

When combinations are considered, the symbol $\diamond$ is put to use when creating the new identifier, and an additional entry is made to register the combined instance identifiers. This new entry is a table that gathers the pairs of identifiers that have been combined. It is represented by $\mathbf{2}^{\mathbf{3}^* \times \mathbf{3}^*}$, which is the algebraic representation of a subset of pairs formed from $\mathbf{3}^*$. The resulting time-stamps are now of the format $\mathbf{3}^* \times \mathbb{N} \times \mathbf{2}^{\mathbf{3}^* \times \mathbf{3}^*}$. This format specifies that the event time-stamps are composed by a instance identifier $\mathbf{3}^*$, a natural index $\mathbb{N}$ that shows the event order within the instance, and a set of pairs of identifiers that have been combined in the past $\mathbf{2}^{\mathbf{3}^* \times \mathbf{3}^*}$.

Figure 3 shows an example of event time-stamping with duplications and combinations[4]. The combined pairs are represented with the notation $\frac{id1}{id2}$ to depict that the identifiers $id1$ and $id2$ have been combined in the past. As such, the subset $\mathbf{2}^{\mathbf{3}^* \times \mathbf{3}^*}$ is representable by the pairs that it holds, as in $\frac{id1}{id2}\,\frac{id5}{id8} \cdots$. Finally, the identifiers for combined instances are obtained by adding a $\diamond$ to one of the affluent instances.

The formal description of this time-stamping method starts by the definition of the event registering procedure.

[4] All these examples start from a common root. If more than one root is needed, in a given system, an initialization procedure should perform the number of duplications that lead to their creation.

Definition 5.1 *Consider that I_j, N_j, Eq_j are the three components of the time-stamps that tags events of an instance j. These components are, respectively, the identifier, the natural index and the set of identifier pairs that keeps the combined identifiers. The value of the identifiers is shown in double quotes and will be subject to concatenations with the operator $+$. The set Eq_j can be indexed so that $Eq_j(id_1)$ yields id_2 if it contains the pair $\frac{id1}{id2}$ or $\frac{id2}{id1}$, yielding nil otherwise.*

- *Initially a first instance i holds a virtual event with, $I_i := "0", N_i := 0, Eq_i := \{\}$.*
- *Upon occurrence of an internal event on an instance j, the natural index is incremented, $N_j := N_j + 1$, deriving the new time-stamp.*
- *Upon a duplication event over the instance j, with the derivation of instances k and l, this event is stored as the last event of j with $N_j := N_j + 1$. The new instances, k and l, are initialized with $I_k := I_j + "0", I_l := I_j + "1"$ and the set of combined identifiers is inherited by $Eq_k := Eq_j, Eq_l := Eq_j$. Finally a virtual initial event initializes the counters $N_k := 0, N_l := 0$.*
- *Upon a combination event, that joins two instances m and n and derives a new instance o, this event is registered as the first event of the new instance with $N_o := 1$. The new identifier can be obtained by $I_o := I_m + "\diamond"$ or alternatively by $I_o := I_n + "\diamond"$. The set of combined instances is inherited and actualized with the recent merge by $Eq_o := Eq_m \cup Eq_n \cup \{\frac{I_m}{I_n}\}$.*

With this procedure the events that occur in a distributed computation can be identified by a time-stamp consisting on the triplet (I, N, E_q). These time-stamps hold the necessary information for comparing two events with respect to their partial order of causality. The definition bellow introduces the appropriate procedures.

Definition 5.2 *Consider I_e, N_e, Eq_e as the values that are associated to a given event e. Making use of a function $\mathcal{S}()$ that applied to an identifier I_x derives the set of identifiers (I_x included) that are initial segments of the identifier I_x, we define a recursive function $\mathcal{A}()$ that derives the set of ancestor identifiers of I_x as:*

- $\mathcal{A}(nil) = \{\}$
- $\mathcal{A}(I_x) = \mathcal{S}(I_x) \cup \bigcup_{I_t \in \mathcal{S}(I_x)} \mathcal{A}(Eq_x(I_t))$

With this function we define:

- $e_a = e_b$ *iff* $I_{e_a} = I_{e_b} \wedge N_{e_a} = N_{e_b}$
- $e_a < e_b$ *iff* $I_{e_a} = I_{e_b} \wedge N_{e_a} < N_{e_b}$ *or* $I_{e_a} \neq I_{e_b} \wedge I_{e_a} \in \mathcal{A}(I_{e_b})$.
- $e_a \parallel e_b$ *iff* $\neg(e_a < e_b) \wedge \neg(e_b < e_a)$

The essential element of this definition is the derivation of the set of past identifiers from the inspection of the I and E_q components of a time-stamp. The recursive procedure uses the initial segments of the identifier to derive the direct ancestor identifiers that lead to it. This first set can then be used to introduce the identifiers from converging branches and recursively find their direct ancestors and other converging branches. After the recursive construction the resulting set will hold all the ancestor identifiers of a given identifier, thus enabling a direct determination of the partial ordering of the time-stamps under consideration.

This technique is akin to vector time-stamps in the sense that any pair of time-stamps hold the necessary information to assess their relation with respect to the partial order. The recursive calculation is confined to the time-stamp data and does not involve the inspection of other time-stamps. Other time-stamping schemes for process causality, that use recursive reconstruction, use it in order to collect a distributed state across several time-stamps [10, 1]. In contrast, this system holds all the needed information locally.

Assessing that this time-stamping scheme is a correct characterization of autonomous causality as stated in definition 4.1 can be achieved by proving lemma 5.1 . Space restrictions, however, condition us to the presentation of the lemma under consideration, together with a simple synopsis of the actual proof.

Lemma 5.1 *Taking any two distinct events e, e' the following correspondence holds:*

- $e \rightarrow e'$ *iff* $e < e'$.
- $e \nrightarrow e'$ *iff* $e \parallel e'$.

Proof Synopsis The proof is established by assessing the validity of the first expression and them inferring the second from the definitions. As for the validity of the first expression the proof can be decomposed on proving the two expressions $e < e' \Rightarrow e \rightarrow e'$ and $\neg(e < e') \Rightarrow \neg(e \rightarrow e')$. Both can be properly derived from the analysis of the definitions 5.1 and 5.2. □

6 Conclusions

Current techniques for causality logging, both for distributed computations and for tagging data dependencies in data based environments, have been essentially based on a fixed number of units. Even when the number of units is allowed to change, the unique identity of the intervening units is either pre-established or assigned in a centralized way.

The emergence of new computing settings where the number and the identity of the nodes is uncertain, calls for the separation of the time-stamping mechanism from the node identity. Additionally, in such systems the data lifespan is often superior to that of the processes that manipulate it, thus reinforcing the insufficiency of processes as the driving force in the causality model.

The resulting causality definition expresses causality links by relating data instances, which can have autonomous existence or be only a representation of a process state. The extended expressiveness of this definition allows the description of communication patterns that resort to epidemic propagation, as well as *ad hoc* interaction among mobile hosts and data repositories. Doing this, our model departs from both process causality and replica versioning schemes, respectively based on version vectors and vector time-stamping.

The associated time-stamping mechanism fulfills the requirements for a correct characterization of the causality definition and allows localized logging of the causality relation. The resulting time-stamps require however a considerable amount of state in order to allow a local characterization of the partial ordering of any two events.

Improving the state consumption in actual implementations is more at issue here than it was for process causality. Further research on the use of known compression techniques for vector time-stamping, as well as the potential for the creation of aggregate time-stamps at points of confluence of several branches, can be expected to derive lighter implementations of this scheme.

As for assessing that the time-stamping scheme is minimum, it rests as an open issue.

7 Acknowledgment

The authors would like to thank Victor Fonte and the anonymous referees for their feedback on early versions of this article.

References

[1] Carlos Baquero and Francisco Moura. Improving causality logging in mobile computing networks. *ACM Mobile Computing and Communications Review*, 2(4):62–66, October 1998.

[2] Maria Butrico, Henry Chang, Norman Cohen, and Dennis G. Shea. Data synchronization in mobile network computer reference specification. In *WMR'98, ECOOP'98 Workshop Reader*. Springer Verlag, 1998.

[3] Colin Fidge. Logical time in distributed computing systems. *IEEE Computer*, 24(8):28–33, August 1991.

[4] A. Gahlot and M. Singhal. Hierarchical clocks. Technical Report OSU-CISRC-93-TR19, The Ohio State University, Computer and Information Science Research Center, 1993.

[5] Jaap Haartsen, Mahmoud Naghshineh, Jon Inouye, Olaf Joeressen, and Warren Allen. Bluetooth: Vision, goals, and architecture. *ACM Mobile Computing and Communications Review*, 2(4):38–45, October 1998.

[6] Leslie Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[7] Kevin Negus, John Waters, Jean Tourilhes, Chris Romans, Jim Lansford, and Stephan Hui. HomeRF and SWAP: Wireless networking for the connected home. *ACM Mobile Computing and Communications Review*, 2(4):28–37, October 1998.

[8] D. Stott Parker, Gerald Popek, Gerard Rudisin, Allen Stoughton, Bruce Walker, Evelyn Walton, Johanna Chow, David Edwards, Stephen Kiser, and Charles Kline. Detection of mutual inconsistency in distributed systems. *Transactions on Software Engineering*, 9(3):240–246, 1983.

[9] Karin Petersen, Mike J. Spreitzer, Douglas B. Terry, Marvin M. Theimer, and Alan J. Demers. Flexible update propagation for weakly consistent replication. In *Sixteen ACM Symposium on Operating Systems Principles*, Saint Malo, France, October 1997.

[10] Ravi Prakash and Mukesh Singhal. Dependency sequences and hierarchical clocks: Efficient alternatives to vector clocks for mobile computing systems. *Wireless Networks*, (3):349–360, 1997. also in Mobicom96.

[11] David Ratner, Peter Reiher, and Gerald Popek. Dynamic version vector maintenance. Technical Report CSD-970022, Department of Computer Science, University of California, Los Angeles, 1997.

[12] Michel Raynal and Mukesh Singhal. Logical time: Capturing causality in distributed systems. *IEEE Computer*, 30:49–56, February 1996.

[13] R. Schwarz and F. Mattern. Detecting causal relationships in distributed computations: In search of the holy grail. *Distributed Computing*, 3(7):149–174, 1994.