Networks

Carlos Baquero **Francisco Moura**
cbm@di.uminho.pt *fsm@di.uminho.pt*
Computer Science Department at Minho University, Braga, Portugal
http://gsd.di.uminho.pt/

Vector clocks, or their compressed representations, have played a central role in the detection of causal dependencies between events in a distributed system. When adapting these techniques to a mobile network, bounding the vector clock size to the number of mobile nodes does not provide a satisfactory approach. This paper builds on previous techniques for efficient causality logging in mobile networks and presents a lighter logging mechanism. The technique is based on a particular partial order that is generated by the interleaving of events on mobile hosts that are mediated by the same support station.

I. Introduction

Mobile computing systems are frequently designed as a network of fixed nodes, mobile support stations (MSS), that give connectivity to a set of mobile hosts (MH) [1]. The MSSs are interconnected by a high bandwidth wire-line network where communication costs are inexpensive. In contrast, the MHs always communicate with the mediation of a hosting MSS using a low-bandwidth wireless or phone channel where costs are at issue. This class of mobile computing systems, although excluding direct communication among MHs, models a vast range of existing systems that include wireless networks of MHs bound to local cells, and nomadic MHs that bind to different MSSs as access points to a wide area network.

Distributed applications that build on this class of mobile computing systems are often modeled as a set of concurrent activities distributed among different MHs. Tracking the causal relationships among these concurrent activities is a basic mechanism for the analysis and debugging of distributed applications and a step towards the design of message delivery and replica consistency policies. It is well established [7] that in a distributed system the causal dependency can be fully characterized by the use of vector clocks [7, 2, 5]. However vector clocks are very sensitive to scalability issues since the vector size is bound to the number of activities, which are here bound to the number of MHs.

Clearly, a dependency tracking mechanism that is bound to the number of MSSs and avoids expensive communication on the MH-MSS link would lead to a more scalable and efficient implementation on this class of distributed systems. With this objective in mind, Prakash and Singhal have proposed two alternatives to vector clocks, for mobile computing systems, namely dependency sequences and hierarchical clocks [6]. In this paper we build on their work, showing how hierarchical clocks can be further simplified, and give a deeper insight into the existent equivalences among the two alternatives.

The organization of the paper reflects its incremental basis, and we follow, whenever possible, the notation used in Prakash and Singhal work.

II. System Model

In order to establish a causal ordering among events, we start by modeling the distributed computation as a set of activities that execute in a group of MHs. We will be concerned with only one distributed computation and, at most, one activity per MH is allowed. Additionally, FIFO communication is required in the MH-MSS channel.

The causality relation ($\rightarrow$) expresses the partial order of events that emerges from the distributed execution. This relation assumes that events that belong to the same activity (same MH) exhibit a total order among them. Events from different activities are related by message send and receive event pairs. This relation is the same as the *happened before* relation defined by Lamport [4] for distributed computations. As this relation is independent from the actual time of event occurrence, as observed by Schwarz and Mattern [7], it is better described by the expression *causes* rather than *happened before*. We follow the definition as established in [7]:

Definition II.1 $\rightarrow$ *is the smallest transitive relation satisfying:*

- $e_i \rightarrow e_j$*, if* e_i *and* e_j *are events in the same activity and* e_i *occurred before* e_j*.*
- $e_i \rightarrow e_j$*, if* e_i *is the event of sending a message and* e_j *is the corresponding event of receiving that message.*

Notice that the first condition assumes a total ordering of events for any given activity. This assumption will have a prime role in the upcoming analysis.

It is well established that there is a simple one-to-one mapping between vector clocks and the causal relation [7].

Definition II.2 $e_i \rightarrow e_j$ *iff* $V(e_i) < V(e_j)$*, when* $V(e)$ *is the vector clock associated to event* e*.*

Having vector clocks with one entry for each MH would provide sufficient information to characterize the causal relation. In contrast, having one entry for each MSS leads to a loss of causality information since it implicitly serializes events at the MSSs. Using one logical clock in each MSS leads to a centralized indexing of the events that occur on the served

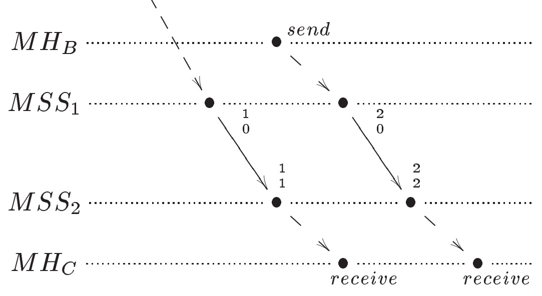

Figure 1: Loss of causality information by implicit total ordering. Since $\frac{1}{0} < \frac{2}{0}$ the send event on MH_B appears to depend on the send event on MH_A, when in fact they are unrelated.

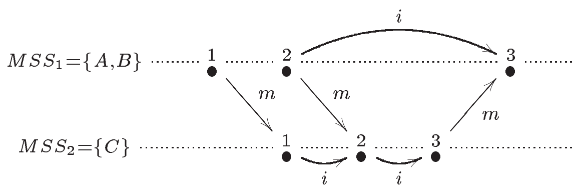

Figure 2: A system with two MSSs and three MHs, showing total event indexing in each MSS. The relations i and m depict causality induced by *internal* events and *messages*, respectively.

MHs (see figures 1 and 2), thus assuming phantom causal dependencies among the events of different MHs, served by the same MSS. For instance, the send event of MH_B in figure 1 is wrongly assumed as having the send event of MH_A in its causal past. This intuition was easily proved by counter example in [6].

Since this loss of information is not admissible it is necessary to complement the information that is stored in these reduced version vectors. The relation labeled by m in figure 2 is captured by vector clocks, but the relation i would be lost, unless some extra information is kept. Relations i and m indicate, respectively, the causality due to internal events and the causality that derives from send/receive event pairs.

III. Dependency Sequences as Causal Histories

This approach complements the loss, due to serialization in the MSSs, by storing, with each event, a set with the events that causally precede that event. Since the numbering of events is local to each MSS in order to register the causality that arises from messaging among MSSs, each event will actually need one set of causal predecessors for each MSS.

It is interesting to observe that this is in fact the registering of a causal history for each event. Building from [7]:

Definition III.1 *Let $E = E_1 \cup \cdots \cup E_N$ denote the set of events in the distributed computation, where N is the number of MSSs. E_i is the set of events that are serialized by a given support station MSS_i and occur in one of MHs that it supports. For a event $e \in E$ occurring in the distributed computation, the* causal history *of e, denoted $C(e)$, is defined as*

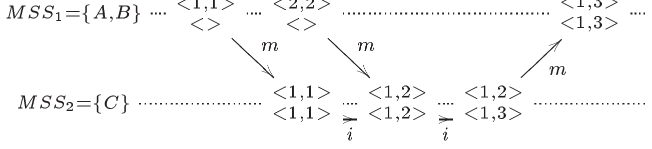

Figure 3: Dependency Sequences. Each vector (here just a pair) $\substack{<>\\<>}$ shows the known causal past. Here the 1st element shows the known events from MSS_1 and the 2nd the ones from MSS_2. Each two entries in the vector $< \ldots, a, b, \ldots >$ indicates a range $(a - b)$ of consecutive events.

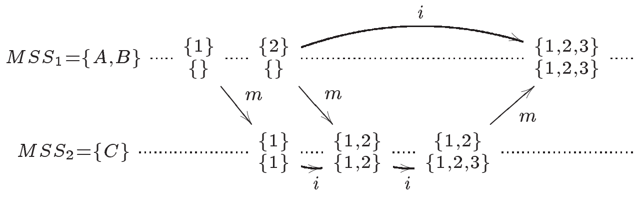

Figure 4: Causal Histories projected in two sets. The notation shows a vector of sets with as many sets as MSSs. Each set depicts the known events as numbered in the respective MSS.

$C(e) = \{e' \in E \mid e' \to e\} \cup \{e\}$.

The projection of $C(e)$ on E_i, denoted $C(e)[i]$, is defined by $C(e)[i] = C(e) \cap E_i$.

This definition assumes that E gathers all events in the distributed computation. Such events could be distinctively labeled by a pair of naturals that identify the MSS that registers the event and the order in which he is registered in that MSS.

An alternative modeling of the causal history is created by associating to each event N sets of causally preceding events, where each set gathers the events that are registered in a particular MSS. In this representation the events in each set can be represented by their ordering index (a natural) since they all belong the same MSS. The projection, defined above, shows how to obtain this hashing of the global events.

Causal histories are shown in [7] as a step towards vector clocks for the representation of a distributed computation among totally ordered processes. If the events on each MSS exhibited a true total order, causal histories would have had redundant information and could be simplified to vector clocks. However, since we need to model the partial ordering of events in a MSS (due to the existence of multiple MHs in the MSS) this simplification is no longer possible.

Schwarz and Mattern's presentation of causal histories proves that the causal history of two events is sufficient to characterize the causality relation among them. It also shows a procedure to constructively create the causal history of every event in the computation. Informally, the algorithm shows that internal events and send events inherit the causal history of its immediate predecessor, and receive events inherit the causal histories of both their predecessors (see figure 4).

We can now observe that the dependency sequences introduced by Prakash and Singhal are in fact compressed representations of causal histories.

Dependency sequences DS represent each projection of the causal history as a set of sequences of consecutive events, as utive events $e_p, \ldots, e_q$ are represented by the pair of naturals p, q. As a consequence, these sequences, have always a even number of elements.

For example, for the sequence of projections of a given causal history $C(e)$ in a system with two MSSs, such as the one depicted by $\langle\{1,2,4\},\{2,4,5,6,7,9\}\rangle$, the corresponding $DC(e)$ would be $\langle\langle 1,2,4,4\rangle,\langle 2,2,4,7,9,9\rangle\rangle$. This can also be represented by bit vectors, which have been used in the implementation of hierarchical clocks (discussed in the next section).

We demonstrated above the equivalence of sequences and sets. A classical representation for sets of N is, as a total function from N into a set with two elements, traditionally denoted as $N \rightarrow 2$ or as 2^N. This total function is thus representable by a binary sequence, in other words a bit vector, with size equal to the greatest integer that exists on the represented set. The above example would be represented as $\langle\langle 1101\rangle,\langle 010111101\rangle\rangle$.

The algorithm described in [6] for the management of dependency sequences is consistent with the constructive technique for the creation of causal histories. To informally argue this equivalence it is necessary to first recall that the MSSs do not keep information about the internal events on each MH, only send and receive events are numbered and registered on the MSSs[1].

The algorithm for the management of dependency sequences ensures that send events inherit the dependency sequence of the last event from that MH, and that this new DS is sent with the message. Receive events inherit the DS of the last event from the receiving MH and merge it with the DS that is received with the message, thus ensuring that both dependencies are combined. It is easy to establish the equivalence of this procedure with the causal history construction procedure, described in the last section.

Independently of the chosen encoding (sets, sequences or bit vectors) all the representations implement causal histories and consequently allow the determination of the causal relation by the appropriate comparison of the data associated to the events. The algorithm for the construction of event logs is also easy to adapt in order to fit each alternative representation. It suffices to implement the appropriate methods for adding an element and joining two representations.

IV. Hierarchical Clocks to Interleaving Clocks

The capability, that dependency sequences exhibit, of storing with the events enough information to compare them, leads to the exchange of considerable amounts of data when logging the system. Hierarchical clocks are presented [6] as an alternative that exhibits lower data exchange needs while constructing the logs. As expected, this also means that the data needed to compare two events is found distributed along other events. This implies, in the general case, communication among MSSs when comparing events.

[1]If tracking of internal events proves to be a needed feature, it suffices to notify them to the MSSs and asks it to number and register the event. While still avoiding local logging on the MH, this would imply a considerable burden on the MH-MSS channel.

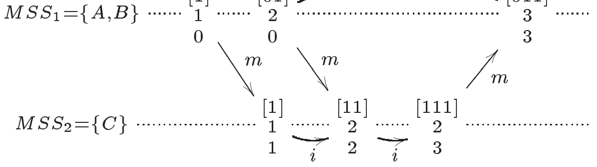

Figure 5: Hierarchical Clocks. These clocks introduce a bit sequence $[\ldots]$ that complements the information logged in the vector clocks (shown under the sequence). Each MSS hosts one bit sequence and this sequence indicates, to each event, which of the other events in that MSS are in fact in his past.

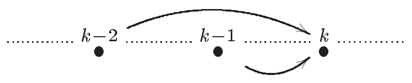

Figure 6: Dependence from two direct causal predecessors. This is a relation pattern dot does not show up in these models of MH/MSS networks, thus leading to simplification opportunities in the logging approach.

Our description of hierarchical clocks will keep building on the notion of complementing the information that is captured when using standard vector clocks with one entry for each MSS.

A vector clock with entries for each MSS is able to capture the causality among communications between MSSs, but assumes a total order on the events seen on each MSS, which we have seen as not appropriate (figure 1). One way to correct this implicit serialization of events, from different MHs, is to tag each event with its local causal history (in other words, with the projection of the causal history for that MSS). As seen in the previous section, this causal history can be represented in a number of ways. Prakash and Singhal use the bit vector representation, which comes as a legacy from the initial description of hierarchical clocks [3] (depicted in figure 5). Hierarchical clocks were initially presented as a technique for the characterization of the causal relation in systems where the processes do not exhibit a total order [2]. This new causal relation, denoted by $\xrightarrow{*}$, has the causal relation $\rightarrow$ as a special case (when the partial order restricts to a total order).

We can now argue that hierarchical clocks are not optimal for an efficient mapping of the system under consideration. This is a natural consequence of the origin of hierarchical clocks, as they where designed to describe a system that exhibited a unrestricted partial order[3]. When a unrestricted partial order is considered, it is necessary to account for situations such as having an event that causally depends on two causally unrelated events (figure 6). These cases are well represented by causal histories and their representations, in particular bit vectors. However this is "overkill" for the system under consideration.

MSSs assign a fictitious total order to their MH events, but

[2]Hierarchical clocks offer a distributed logging alternative to the earlier concept of *bit-matrix clocks*, a clock notation that provided representation for causal histories.

[3]By *unrestricted partial order* we mean a partial order that is not subject to other invariants that further restrict the partial order definition. For instance a *total order* or a *interleaving of total orders* cannot be said to be *unrestricted partial order*.

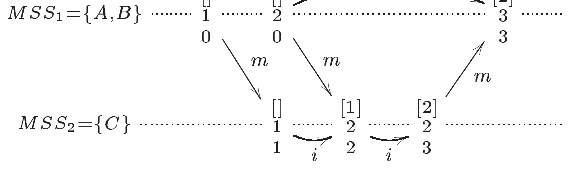

Figure 7: Interleaving Clocks. Hierarchical Clocks can be safely simplified by replacing in each event the bit sequence with the index, in the MSS, of the last event from its MH. This can be visually checked by following the i relations in the opposite direction.

each MH events are themselves totally ordered. What the MSS does is centralize the assignment of monotonic indexes to observed events, thus interleaving the unrelated total orders. Any representation technique that enables the reconstruction of each MH total order, from their joint indexing, would suffice. This can be achieved by storing with each event the index of the last event in its MH, or nil for the first events. We will name this approach as *interleaving clocks* (figure 7), and show how it adds the missing information to MSS vector clocks.

Other known algorithms for the transitive calculation of vector clocks also build on the identification of the most recent predecessors of the incoming event chains [7]. In this case it suffices to identify one predecessor since each event is preceded by a single event chain, the one that links events in the same MH.

The construction algorithm, described bellow, uses a temporary index $L()$ that identifies in each MSS the last event from each of its MHs. For instance in a given MSS and at a given point in time, $L(x)$ will indicate the index in that MSS of the last event that concerns MH_x.

In contrast the other variables, $V()$ and P, are bound to events. They represent the logged state (that is shown in figure 7) so that, in the notation $\overset{[\mathbb{N}]}{\nu}$, where ν is a vertical vector of naturals, we have P represented as $[\mathbb{N}]$ and the vector $V()$ as ν. The algorithm is based on the derivation of the new event state from the last logged event and the temporary index $L()$.

Definition IV.1 *Let $V()$ be a vector clock that tags events on a group of N MSSs, and P a natural that indicates the local predecessor of a given event. Let $L()$ indicate the last event index for a given MH.*

- *Initially, $V(k) := 0$ for $k = 1, \dots, N$, and $L()$ is initialized to nil.*
- *When a message send request is received from MH_i in a hosting MSS_k, $V(k) := V(k) + 1$, $P := L(i)$ and then $L(i) := V(k)$. The message is sent with the new vector $V()$.*
- *When a message is received with $V_m()$ aimed at MH_j by MSS_l, $V(l) := V(l) + 1$, $P := L(j)$ and then $L(j) := V(l)$. Next, $V()$ is updated by taking the pairwise maximums between the local $V()$ and the received $V_m()$.*

Interleaving clocks when compared with hierarchical clocks offer a different balance between local and remote state, that

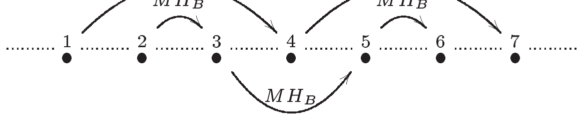

Figure 8: Interleaving of two MHs on a MSS. Each MH events follows an independent path along the index.

is to be used when comparing two events. Hierarchical clocks keep as a bit vector the set of the local causal predecessors in the same MH, while interleaving clocks just keep the last local causal predecessor and recursively reach the other local predecessors. This simplification is possible since the partial order among events seen by a MSS is a interleaving of total orders, of MH events, rather than a unrestricted partial order (see figure 8). The calculation of the set of local predecessors, that would be shown if using hierarchical clocks, can be determined within the local state of the MSS. A higher use of recursion in the local computation, with associated time penalties for some traces, is the tradeoff for logging a more compact representation. Nevertheless, if the local calculation is done first, then both approaches would engage in the same amount of communication among MSSs, which is the most expensive portion, but considerable space would have been spared when using interleaving clocks.

As with hierarchical clocks, handoffs must be managed by a convenient state transfer among the two MSSs. The handoff procedure for interleaving clocks can easily be expected to be of the same magnitude as hierarchical clocks handoffs.

V. Conclusions

Our analysis of the alternatives for efficient causality logging in mobile computing networks, originally proposed in [6] focused on bringing evidence to the underlying concept, *causal histories*. Sets, bit vectors and sequences are shown to be alternative representations of the same concept. Once notation is ruled out, the two alternatives differ by the tradeoffs between local storage with redundancy, thus enabling local calculation, versus distributed storage with distributed recursive calculation of event dependencies.

Dependency sequences are shown to be an implementation of causal histories and their construction algorithm can be derived from causal history construction. The choice of the best representation can now be seen as a separate issue which opens the potential for hybrid representations and for translations among representations. Each alternative exhibits different storage tradeoffs for different samples of data, for instance sets are more economic than sequences for causal histories with many isolated events.

We have shown that hierarchical clocks are overly expressive for the modeled context, thus raising simplification opportunities. Once established that the modeled partial order is restricted to a interleaving of total orders it was possible to replace the bit vectors (or other set representation) by a single natural. The extra calculation uses only the local state, which leads to a very favorable tradeoff between saved space and ex-

VI. Acknowledgment

The authors would like to thank Ravi Prakash, Victor Fonte and the anonymous referees for providing valuable feedback that helped to improve this work.

References

[1] B. R. Badrinath, Arup Acharya, and Tomasz Imielinski. Structuring distributed algorithms for mobile hosts. In *14th International Conference on Distributed Computing Systems*, June 1994.

[2] Colin Fidge. Logical time in distributed computing systems. In *Computer, Issue on Distributed Computing Systems*, number 24(8), pages 28–33. IEEE, 1991.

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

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

[5] 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.

[6] 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.

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

Biographies

Carlos Baquero received his MSc degree in Computer Science from Minho University at Braga Portugal, in 1994. He is currently a Teaching Assistant in Distributed Systems and PhD candidate at Minho University. His primary research interests encompass replicated data management in mobile computing and mobility modeling.

Francisco Moura holds a degree in Electrical Engineering since 1977 and received his MSc and PhD degrees in Computer Science at Manchester University, UK in 1982 and 1985, respectively. He is Associate Professor at Minho University and leads the Distributed Systems Group at the Computer Science Department.