Towards Efficient Time-stamping for Autonomous Versioning

Carlos Baquero cbm@di.uminho.pt
Paulo Sérgio Almeida psa@di.uminho.pt

Distributed Systems Group
Departamento de Informática
Universidade do Minho, Braga, Portugal
http://gsd.di.uminho.pt/

Abstract

We sketch a decentralized versioning scheme that handles the detection of concurrent updates among an arbitrary number of replicas, overcoming the limitations that a centralized knowledge of that number imposes to Mobile Computing.

1 Introduction

The characterization of causal dependencies among distributed activities plays an important role on the analysis of distributed systems and, in particular, on the definition of versioning schemes for distributed persistent entities. In fact, it was a versioning scheme [5] that made one of the first presentations of *vector timestamps* as a mechanism that was later seen to be a full characterization of Lamport's *causality* [3][1].

Recently, it has been shown [1] that the advent of mobile computing, by stimulating cooperation among arbitrary groups of nodes, precludes the use of globally known sets of entities, thus invalidating the use of vector timestamps. This observation lead to a new causality definition coined *Autonomous Causality*, that tracks causality among arbitrary numbers of instances, while embedding Lamport's causality. A time-stamping scheme that characterizes autonomous causality was also introduced.

In this paper we will show that a less space demanding time-stamping scheme can be adopted if the goal is, not to track the causal dependency among all events that have occurred, but to compare only the present versions of entities related by autonomous causality. The resulting time-stamping scheme has the additional advantage of allowing

[1] These characterizations were independently done by Fidge [2] and Mattern [4]

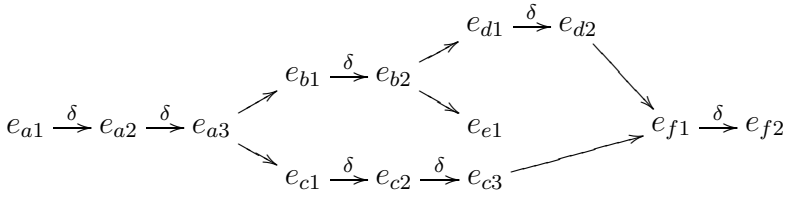

Figure 1: A set of partially ordered events, depicting causality.

strong simplifications upon the eventual convergence of deployed versions. A direct application of this mechanism is an autonomous versioning tool for the copying of files among fixed, mobile machines and transportable media.

2 Time-stamping Versions

Consider the set of events, in Figure 1, related by causality. A time-stamping scheme that partially orders all events in the distributed computation, should be able to relate, for instance, events e_{e1} and e_{b2}, and detect that $e_{b2} < e_{e1}$. In fact this is what the time-stamping mechanism presented in [1] does.

If we restrict our universe of analysis to those instances that are available at a given moment in time, we can observe that, since the instance e_{e1} is produced from duplication of instance e_{b2}, we will never be required to compare them. We may, however, be requested to compare the pairs of instances (e_{d1}, e_{c2}) or (e_{d2}, e_{c1}) since they may co-exist in a particular run. For a versioning system it would be useful to know that collapsing e_{d2} and e_{c1} into a common version would be straightforward, since they have a common ancestor (e_{a3}) and e_{c1} did not suffer any modification (depicted by δ arrows). On the contrary, collapsing e_{d1} and e_{c2} would require some conflict resolution since they depict concurrent evolutions on their state.

Our interest in only supporting the relations among instances that can co-exist in a given moment in time is motivated by the economy that it conveys to logged state. On the contrary, the representation strategy used in vector timestamps cannot exploit this kind of space saving, as it is designed so that all events in the system can be compared.

2.1 Adapting Boolean Expressions and Karnaugh Maps

The key to the time-stamping technique for versioning, is to view the set of all existing versions in a given point in time as a partition of the *universe*. Each version is (partially) represented by a boolean expression; the set of all existing versions corresponds to a set of expressions whose sum is *true*; forking versions corresponds to refining the representation by having one more boolean variable; and converging two versions corresponds to the sum of two expressions, upon which a simplification may be possible (where known techniques to simplify boolean expressions, such as Karnaugh maps, can be used.)

For instance, after one division the universe can be represented by the two components $\{a, \overline{a}\}$, and after a subsequent division of $[\overline{a}]$ would be seen as $\{a, \overline{a}b, \overline{a}\overline{b}\}$. In our notation,

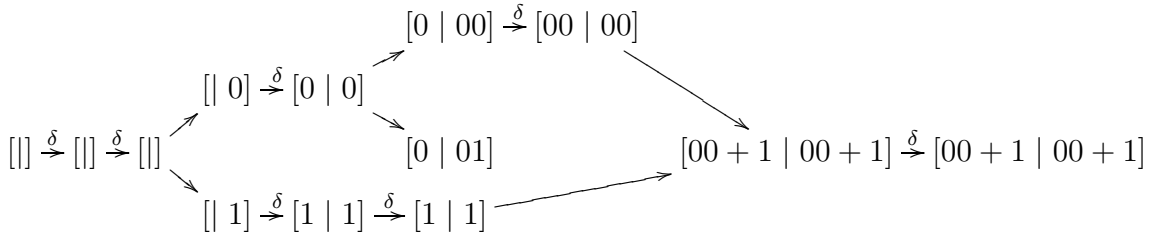

Figure 2: A set of partially ordered events with versioning timestamps.

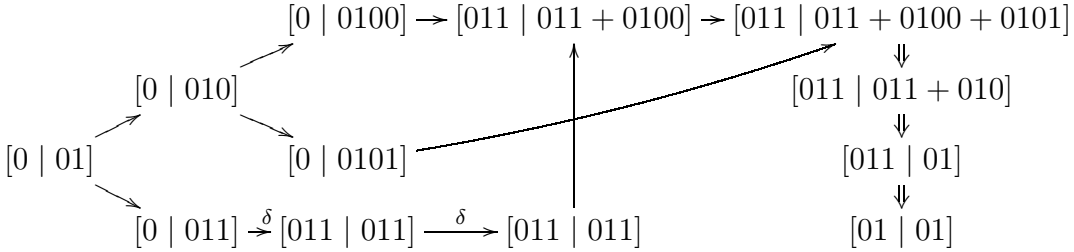

Figure 3: Replication and total convergence of instance [0 | 01].

we adopt a more systematic representation of the variables, by using strings of $\{0, 1\}$ where the position in the string identifies the variable and the digit's value its logical state (either plain or negated).

Figure 2 shows the time-stamping of the events previously depicted in Figure 1. Each time-stamp has two components, where the second identifies the instance, by describing the subset of the universe that it represents, and the first indicates the nearest instance in its past that has suffered modifications.

When comparing instances like $[00 + 1 \mid 00 + 1]$ and $[0 \mid 01]$ it is possible to check that $[0 \mid 01]$ has not suffered more updates than those already seen in $[0 \mid 0]$, that, taken together with the fact that $[0 \mid 0]$ is in the direct past of $[00 + 1 \mid 00 + 1]$ leads to the absence of concurrency conflicts among those instances. As a general rule, we state that it suffices to consider the first component when comparing any two instances with the purpose of checking update domination.

If we now derive a new version that replaces $[00 + 1 \mid 00 + 1]$ and $[0 \mid 01]$, we obtain the identifier $[00 + 1 \mid 00 + 1 + 01]$ that simplifies to $[00 + 1 \mid]$ (applying rules as in boolean expression simplification to the second component), and finally to $[|]$, since the first component must not represent a refinement of the second. In fact the simplification rules are capable of collapsing any branching that has been done in the past once they converge. This collapsing acts at any level of detail, which means that some intermediate collapsing can be done even if some outer branches do not happen to converge.

The second example, in Figure 3, introduces a slightly more complex case where a replica instance $[0 \mid 01]$ evolves into a set of branches that end up converging again into a single instance. It can be seen that only in one of the branches were there updates; therefore, this particular branch dominates all the others when convergence takes place.

In a practical setting this maps a case when several replicas are generated, in order to

optimistically allow updates to any of the replicas. Later when they are all available for cross reconciliation the system realizes that only one suffered updates, and consequently it is able to automatically choose that one to dominate over the others. Otherwise if conflicts where detected, there would be a need for user driven reconciliation and the generated replica would dominate its ancestors.

Finally, the rightmost part of Figure 3 shows a set of simplification steps (connected by $\Rightarrow$) that lead to the instance representation [01 | 01]. This means that, after convergence, all the intermediate branching can be forgotten and the whole case becomes equivalent to the single transition $[0 \mid 01] \xrightarrow{\delta} [01 \mid 01]$.

3 Conclusions

We have briefly sketched a time-stamping mechanism with built-in simplification rules upon convergence, adapted from the representation of boolean expressions and their simplification. This mechanism supports a versioning scheme with autonomous replication and collapsing of instances, and enables the comparison between any two existing instances. A formal treatment of the work here presented and of the general case of characterizing the partial order among all events will be addressed in the near future.

The presented mechanism can be applied to the construction of file copying constructs that keep track of dependencies and notify users when a given copy can be safely replaced by another, or has suffered a concurrent evolution and needs some prior reconciliation.

Acknowledgements The authors would like to thank Victor Fonte and the anonymous reviewers for their comments.

References

[1] Carlos Baquero and Francisco Moura. Causality in autonomous mobile systems. In *Third European Research Seminar on Advances in Distributed Systems*. Broadcast, EPFL-LSE, April 1999.

[2] Colin Fidge. Timestamps in message-passing systems that preserve the partial ordering. In *11th Australian Computer Science Conference*, pages 55–66, 1989.

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

[4] Friedemann Mattern. Virtual time and global clocks in distributed systems. In *Workshop on Parallel and Distributed Algorithms*, pages 215–226, 1989.

[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.