Bounded Time-Stamps

Amos Israeli[*]
Electrical Engineering Department
The Technion — Haifa, Israel

Ming Li[†]
Computer Science Department
University of Waterloo
Waterloo — Canada

January 15, 1999

Abstract

Time-stamps are labels which a system adds to its data items. These labels enable the system to keep track of the temporal precedence relations among its data elements. Many distributed protocols and some applications use the natural numbers as time-stamps. The natural numbers however are not useful for bounded protocols. In this paper we develop a theory of *bounded time-stamps*. Time-stamp schemes are defined and the complexity of their implementation is analyzed. This indicates a direction for developing a general tool for converting time-stamp based protocols to bounded protocols.

[*]Supported in part by The Weizmann fellowship and NSF Grant DCR-86-00379.

[†]Supported in part by ONR Grant N00014-85-k-0445 and Army Research Office Grant DAAL03-86-K-0171 at Harvard University, by NSF Grant DCR-86-06366 at Ohio State University, and by NSERC Operating Grant OGP0036747. Most of this work was done when the authors were at Aiken Computation Laboratory at Harvard University. The authors also acknowledge the hospitaity of the computer science department at York University, Canada.

1 Introduction

Time-Stamps are used quite extensively in many areas of computer science as well as in practical applications. A time-stamp is a label attached to any of the system's data items. The data item to which a time-stamp is attached is called its *owner*. Traditionally, in a single processor system, a time-stamp value is obtained by reading the system's clock. This value indicates the time at which the owner of the time-stamp was created, accessed, updated, etc. Time-stamps are used for many of the operating system functions such as cross-reference, naming and accounting.

Time-stamps may be obtained by computation rather than by clock reading. For example, a processor may scan all existing time-stamps and choose a number which is greater than all the scanned numbers as the new time-stamp. A common example is the process naming method of the UNIX operating system. In this method the name of a new process is the value of a variable which is incremented by 1 whenever a new process is initialized.

In a multi-processor asynchronous system there is no global clock, hence there is no computable notion of global time. Local clocks might be set to different times and their paces might be skewed. In such an asynchronous system it is not the absolute occurrence time of events that is important, but rather the temporal order of occurrence of different events. The crucial role of time-stamps in asynchronous environments is to maintain the ordering of various events of the system's execution. The system represents events as data items, (e.g. messages). At any given time all data items concerning a certain protocol might be gathered in some set. As execution of the protocol proceeds elements are added to and removed from this set. Time-stamps enable the system to keep-track of the order in which elements were added to the set (i.e., the order of occurrence of the events represented by this set).

It is common use in theoretical computing as well as in real-life applications that time-stamps are generated and used as *unbounded* numbers. For example we cite the works of Lamport in [La-74] on mutual exclusion, Vitanyi and Awerbuch in [VA-86] on multi-writer atomic register, Chor Israeli and Li in [CIL-87] on wait-free consensus and the aforementioned UNIX process naming convention. In many theoretical works bounded solutions are sought. For this purpose ordinary time-stamps cannot be used since they are unbounded. All aforementioned problems were solved by ad-hoc bounded solutions. Our goal is to identify the common mechanism underlying in these works and to supply a mechanism bounded in memory with similar properties.

In this paper we develop a theory of *Bounded Time Stamps*. We first define sequential bounded time-stamp schemes and explore the possibility of implementing them. Towards this end we introduce a new pebble game and use it to analyze the complexity of such an implementation. Using the game we prove that time-stamp schemes for n elements should have at least $\Omega(2^n)$ labels. We proceed by giving a nonconstructive proof of the existence of time-stamp schemes with $O((2+\epsilon)^n)$ labels and by constructing a time-stamp scheme with $O(3^n)$ labels.

The work of Katseff in [Ka-78] presents a bounded protocol for mutual exclusion which has the First Comes First Serve (or in short *FCFS*) property. The *FCFS* property requires keeping a temporal order among the processors seeking mutually exclusive access to a common resource. This protocol implicitly contains a a bounded time-stamp scheme at its core, though the author does not use this abstraction. Another time-stamp scheme can be extracted from the FCFS mutual exclusion of Lamport in [La-86a, La-86b].

A similar problem of *bounded sequence-numbers* was discussed in message passing based distributed systems; for example see the works of Finn [Fi-79], Segall in [Se-83], Soloway and Humblet in [SH-87] and Katz and Shmueli in [KS-87]. All these papers use bounded schemes in which in some circumstances a processor cannot pick up a sequence-number. In these circumstances the processor *waits* until some sequence-numbers are eliminated from the system by other processors. This type of solutions is acceptable in message passing environments in which waiting is imposed by the assumption of unbounded delays over links. In shared memory systems we seek for wait-free protocols in which a processor's progress is determined only by its own pace; in this environment these simple methods are ruled out because they require waiting.

The rest of this paper is organized as follows: In Section 2 bounded time-stamp schemes are defined and the complexity of their implementation is analyzed by presenting almost tight lower and upper bounds. Section 3 contains some concluding remarks

2 Time-Stamp Schemes

Let $G(V, E)$ be a directed graph with no self-loops. A node $u \in V$ *dominates* a node $v \in V$ if there is an edge from u to v. The graph G is a *tournament* if for every two nodes $v, u \in G$ either v dominates u or u dominates v but not both. The graph G is a *partial tournament* if G has no directed cycles of length 2. An *ordered set* of a partial tournament $G(V, E)$ is a set of nodes $U \subseteq V$, such that the graph induced by U is an acyclic tournament.

Let S be a dynamically changing set of owners. Elements can be added to S or removed from it in any arbitrary order, as long as the number of elements in S is bounded by n. Our aim is to label each element anew upon its entry to S. The labels determine the order in which the active elements (i.e. still in S) were added to S. The labeling algorithm may use all existing labels as input, but is not allowed to change any of them, since they might be used as means of referencing their owners. Once an element is labeled, its label stays until the element is removed from S.

Deciding the relative order of two elements of the set S should depend only on their *own* labels. To see this imagine that all other elements are removed from S. The relative order of the two remaining elements does not change by removing the others, thus their order must be determinable by looking only at their labels. The fact that every pair of labels is ordered without any dependence on other labels suggests the use of a directed graph. In this graph nodes will correspond to labels. If v and u are two nodes of the graph, i.e. two labels, then the fact that v precedes u will be denoted by a directed edge from u to v. The task of finding a label for a new element of S will now correspond to finding a dominating node (label) to the nodes (labels) of all elements already in S.

One way to do this is to label each new element by computing the maximum label of all active elements and adding 1 to it. This is the *natural* time-stamp scheme. In this scheme, the length of the labels is unbounded and depends on the number of element additions that are done during the life time of the system. Its graph, which is infinite, can be described as follows: The set V is the set of natural numbers. Every pair of nodes, n, m $(n < m)$ is connected by an edge directed from m to n. A dominator v for any given finite set of natural numbers,

$U = \{u_1, u_2, \ldots, u_l\}$, is computed by $v := \max(U) + 1$. Note that the natural time-stamp scheme can serve sets of any (finite) size.

In this paper we restrict ourselves to *sequential* time-stamp schemes. In Sequential schemes the set S is updated sequentially. At any given time at most one update action (either element removal or element addition) takes place. Time-stamps may be chosen by several processes which communicate with each other only by reading these time-stamps as long as each update action is terminated before another one starts. The label of a newly entered element is more recent than all other label and hence it dominates all of them. Thus in a sequential scheme every pair of nodes is ordered and so is the entire set. To analize these schemes we introduce a game for 2 players and n pebbles that models the problem of finding new labels.

2.1 The Pebble Game

The game is played on a partial tournament G. The graph G is augmented by a special node called the *initial node* of G and denoted by $\perp$. The initial node is dominated by all other nodes of G. For convenience of notation we exclude the initial node from the graph G. In the beginning of the game the n pebbles reside on the initial node $\perp$. The game proceeds in steps as follows: In each step player $\mathcal{A}$, the adversary, chooses a pebble. Player $\mathcal{L}$, the labeler, has to move the chosen pebble to a node that dominates all the other nodes on which the other $n-1$ pebbles reside. In this way by the end of each move of the labeler, all pebbles are linearly ordered according to the the sequence in which they made their last move.

A *sequential time-stamp scheme* of order n is a pair (G, P) where G is a partial tournament on which the game is played and P is a *pebble moving protocol* for the labeler, $\mathcal{L}$. The protocol P should supply $\mathcal{L}$ with an infinite sequence of answers, produced on-line, for every infinite sequence of choices made by $\mathcal{A}$. By letting the adversary choose which pebble is moved we ensure the correctness of the protocol in all possible situations. Note, the set of nodes with pebbles is always ordered.

A time-stamp scheme of order n accommodates a dynamic set S of up to n elements in the following way: The nodes of the graph, except the $\perp$, are the possible labels (time-stamps) of the scheme. The nodes of the graph on which the pebbles reside (except $\perp$) are the labels of existing elements. An active/inactive bit is added to each pebble. All pebbles on $\perp$ are marked *inactive*. Removing an element from S is done by marking the corresponding pebble inactive while keeping it on its node. Adding an element to S is done by moving an inactive pebble to a dominating node and making it active.

After formulating the notion of sequential time-stamp schemes we turn to time-stamp schemes of *bounded size*. Are there any time-stamp schemes with a bounded size graph? What are the bounds on the size of the graph? What is the computational complexity of the protocol P in these bounded schemes?

2.2 Lower Bound

A partial tournament is *n–good* if it is the graph of a sequential time-stamp scheme of order n.

Theorem 1: If a graph $G(V, E)$ is n–good then it has at least $2^n - 1$ nodes.

Proof: The proof is by induction on n, the number of pebbles.

Base: If n=1 then $|V|$ has at least one node.

Induction step: Assume correctness of the theorem for $k < n$ and let $G(V, E)$ be an n–good graph. We may assume that every node $v \in V$ is used by some sequence of pebble movements, If this is not the case we will just remove v from G. We claim that the in-degree of every node $v \in V$ is at least $2^{n-1} - 1$. This will immediately imply our theorem since for $|V| = N$ we get:

$$\begin{aligned}
|E| = \sum_{v \in V} d_{in}(v) \;&\geq\; N \cdot (2^{n-1} - 1) \\
\frac{N \cdot (N-1)}{2} \geq |E| \;&\geq\; N \cdot (2^{n-1} - 1) \\
N \;&\geq\; 2^n - 1.
\end{aligned}$$

In order to show that the indegree of an arbitrary node v in G is $2^{n-1} - 1$ we show that the graph G_v, induced by all nodes that dominate v is $(n-1)$–good. By the induction hypothesis this implies the claim immediately. Let H be an isomorphic copy of G_v. The graph G is n–good and hence has a labeling protocol $\mathcal{P}$ for a labeler, $\mathcal{L}_G$. We now define a pebble moving protocol $\mathcal{Q}$ for a labler $\mathcal{L}_H$ to play the game on the graph H. The protocol $\mathcal{Q}$ uses the protocol $\mathcal{P}$ as an oracle. It starts by running $\mathcal{P}$ until a pebble is put on node v. By our previous assumption this is always possible (otherwise v is removed from G). Once a pebble is on v $\mathcal{L}_H$ is ready to play on H. A $(1:1)$-mapping is formed between the pebbles on the initial node $\perp$ of H and all pebbles on G, except the one residing on v. Whenever $\mathcal{A}_H$ asks $\mathcal{L}_H$ to move a certain pebble on H, $\mathcal{L}_H$ points, as an adversary, to the corresponding pebble on G_v, watches the reaction of $\mathcal{L}_G$ which must keep the moved pebble on G_v, to dominate v, and copies this movement on H. Since G is n–good there is an answer for every sequence of choices of $\mathcal{A}_G$, so every choice of $\mathcal{A}_H$ can also be answered. Therefore H is $(n-1)$–good. $\square$

2.3 Upper Bound

The problem of existence of bounded-size time-stamp scheme is related to a problem raised by Schutte and solved by Erdös [Er-63]. A tournament T is said to have the property S_k if every set of k nodes of T is dominated by a single node. Schutte had defined this property and asked if there are such graphs at all. Erdos proved, using a nonconstructive probabilistic argument that there is a tournament of size $O(2^k \cdot k^2)$ with the property S_k. E. Szekeres and G. Szekeres in [Sz-65], prove that if $T(V, E)$ is S_k then $|V| = \Omega(2^k \cdot k)$. Another result presented in [Sz-65] is a graph with 19 nodes which is S_3. Explicit construction of S_k graphs, for all k is given in [GS-71]. The size of an S_k graph in this construction is $k^2 2^{2k-2}$.

It is obvious that if T_k is an S_k tournament then it is $(k+1)$–good. In T_k each set U, $|U| \leq k$ has a dominator. The labeling protocol P operating on any set U should simply return its dominator. Hence the existence proof of [Er-63] proves the existence of bounded size sequential time-stamp schemes of any order. Unfortunately it is not clear whether the structure of T_k in the construction of [GS-71] allows an efficient computation of a dominator for all sets of k nodes. Presumably, computing a dominator for any set would require keeping all of T_k in memory. The complexity of computing a dominator appears to be at least linear in the degree of each node, i.e., exponential in n, the number of elements in the set S.

We now present a method to construct bounded time-stamp schemes in which dominators can be computed efficiently: The *composition* of two tournaments, G and H, denoted by $G \circ H$, is the following noncommutative operation: Replace each node of G by a copy of H. If (v_1, v_2) is an edge of G, directed from v_1 to v_2, then every node in H_1, the graph substituted for v_1, dominates all the nodes of H_2, the graph substituted for v_2. The following theorem shows how graphs obtained by tournament composition can be used to implement efficient time-stamp schemes.

Theorem 2: Let (G_1, P_1) and (G_2, P_2) be time-stamp schemes of orders $(n+1)$ and $(m+1)$ respectively. Then there is a protocol P such that $(G = G_1 \circ G_2, P)$ is a time-stamp scheme of order $(n+m+1)$. Furthermore if t_1 and t_2 are the time complexities of P_1 and P_2 respectively then the time complexity of the protocol P is $O(t_1 + t_2)$.

Proof: For any ordered set S, of the graph G, let the *dominator* of S be the element of S which dominates all its other elements. Define the copy of G_2 in which the dominator of S resides to be the *dominating copy* of G_2. We prove the theorem by presenting a labeling protocol P for the game played on the graph G augmented by an initial node $\perp$. The labeling protocol P is defined as follows: In the beginning of the game all pebbles reside on $\perp$. Assume that the adversary points to a pebble pb. Consider the following cases:

Case 1: pb is not on the dominating copy.

- **Case 1.1: There is no dominating copy**. This happens when all pebbles are on $\perp$. In this case use P_1 to find a new dominating copy of G_2. Once a new dominating copy is found use P_2 to determine the new dominator in the dominating copy. When P_2 is used assume pb is on $\perp_2$ the initial node of G_2.
- **case 1.2: The dominating copy has $m+1$ pebbles.** This means that after removing pb there are at most n copies of G_2 with pebbles on them. Use P_1 to find a new dominating copy of G_2. Then use P_2 to find a new dominator inside the new dominating copy and put pb on it.
- **case 1.3: The dominating copy does not have $m+1$ pebbles on it**. In this case use P_2 to find a dominator of all pebbles on the dominating copy and put pb on it.

Case 2: The pebble pb is on the dominating copy of G_2. In this case move pb, using the protocol P_2, to a dominating node of the dominating copy.

The correctness of the protocol P is immediate. □

Define the graph B^n inductively as follows:

1. B^2 is the directed cycle with 3 nodes.

2. $B^{k+1} = B^k \circ B^2$, for all $k \geq 1$.

Using Theorem 2 we can present an actual bounded size time-stamp scheme of order n. Its graph is B^n. The complexity of computing a new dominator for any set of pebbles is linear in n, the number of elements in S. The number of nodes in the graph is 3^{n-1} but the only nodes which need to be kept in memory of $\mathcal{L}$ are the nodes of (up to n) elements of S. The following theorem proves the existence of more space efficient time-stamp schemes.

Theorem 3: For any $\epsilon > 0$, for large enough n, there is an n–good graph of size $(2+\epsilon)^n$. The time complexity of the pebble moving protocol is polynomial in n.

Proof: By [Er-63] for any large enough integer n there exists a graph G of size $c^2(n(\log n)^2)$ with property $S_{\log n}$ where c is some constant. By the previous discussion G is $(\log n + 1)$–good. Finding a dominator for any arbitrary set of $\log n$ elements in G can be done exhaustively in $O(n(\log n)^3)$ time. For any $\epsilon > 0$, take $n > n_0$ such that

$$\frac{2 \log c \log n_0}{\log n_0} < \log(1 + \frac{1}{2} \cdot \epsilon). \tag{1}$$

By Theorem 2 an n–good graph is obtained by composing G with itself $n/\log n$ times to get $G^{n/\log n}$. Finding a dominator in $G^{n/\log n}$ is easily done in time polynomial in n. The size of $G^{n/\log n}$ is:

$$c^2(n(\log n)^2)^{\frac{n}{\log n}} = 2^n(c \log n)^{\frac{2n}{\log n}}$$

By (1) we conclude

$$(c \log n)^{2n/\log n} < (1 + \epsilon/2)^n$$

This completes the proof. □

3 Concluding Remarks

We have motivated, defined and analized *bounded time-stamp schemes* which we believe will play a central role in the theory of bounded distributed systems. Unfortunately our original aim of presenting automatic conversion of time-stamp based protocol to bounded protocols was not fully achieved in this paper. The original presentation of this paper in [IL-87] contained a concurrent labeling protocol which enabled the implementation of a multi-writer, multi-reader atomic register which uses sequential bounded time-stamps as a subroutine. Some other applications of (sequential) bounded time-stamps were proposed in [AMS89, AGTV92] and others. Concurrent bounded time-stamp schemes were defined and implemented in [DS-89]. More efficient implementations of concurrrent bounded time-stamp schemes were presented in [DW92, DH92, IP92].

Acknowledgements

The authors wish to thank Paul M. B. Vitányi for stimulating discussions and continuous help during the work on this paper. We also thank Mihaly Geréb-Graus Danny Krizanc and John Tromp for helpful discussions.

References

[AGTV92] , Y. Afek, E. Gafni, J. Tromp and P. Vitanyi, "Wait-Free Test-and-Set", Lecture Notes in Computer Science 649: Distributed AlgorithmsP (Proceedings of the Fifth International Workshop on Distributed Algorithms, Haifa, Israel, November 1992), A Segall and S. Zaks, Editors, pp. 95-109, Springer-Verlag, 1992.

[AMS89] B. Awerbuch, Y. Mansour and N. Shavit: "Polynomial End-to-End Communication", proceedings of 30th Anjnual Symposium on Foundations of Computer Science, 1989, pp. 358-363.

[CIL-87] B. Chor, A. Israeli, and M. Li, "On Processor Coordination Using Asynchronous Hardware", 1987 ACM PODC, pp. 86-97.

[DS-89] D. Dolev and N. Shavit "Bounded concurrent time-stamp systems are constructible," Proceedings of the 21st Annual ACM Symposium on Theory of Computing (1989) pp. 454-466.

[DH92] C. Dwork, M.Herlihy, S.A. Plotkin and O. Waarts, "Time-lapse snapshots," Proceedings of the Israeli Symposium on the Theory of Computing and Systems (1992) pp. 154-170.

[DW92] C. Dwork and O. Waarts "Simple efficient bounded concurrent timestamping or Bounded concurrent timestamp systems are comprehensible!," Proceedings of the 24th Annual ACM Symposium on Theory of Computing (1992) pp. 655-666.

[Er-63] P. Erdos, "On a Problem of Graph Theory", Math Gazette, vol. 47, 220-223, 1963.

[Fi-79] S.G. Finn, "Resynch Procedures ana a Failsafe Network Protocol", IEEE Transactions on Communication, vol. COM-27, no. 6, pp. 840-846, June 1979.

[GL-92] R. Gawlick, N. Lynch and N. Shavit, "Concurrwent Timestamping Made Simple," Proceedings of the Israeli Symposium on the Theory of Computing and Systems (1992) pp. 171-183.

[GS-71] R.L. Graham and J.H. Spencer, "A Construction to a Tournament Problem", Canadian Math. Bulletin, vol. 14(1), pp. 45-48, 1971.

[IL-87] A. Israeli and M. Li, "Bounded Time-stamps", 28th Annual IEEE Symp. on Foundations of Computer Science, 1987, pp. 371-382.

[IP92] A. Israeli and M. Pinhasov, "A Concurrent Time-Stamp Scheme which is Linear in Time and Space", Lecture Notes in Computer Science 649: Distributed Algorithms (Proceedings of the Fifth International Workshop on Distributed Algorithms, Haifa, Israel, November 1992), A Segall and S. Zaks, Editors, pp. 95-109, Springer-Verlag, 1992.

[Ka-78] H.P. Katseff, "A New Solution to the Critical Section Problem, *Conference Record of the 10th Annual ACM Symposium on the Theory of Computing* San Diego May 1978, pp.86-88.

[KS-87] S. Katz and O. Shmueli, "Cooperative Distributed Algorithms for Dynamic Cycle Prevention", IEEE Trans. on Software Engineering, Vol SE-13, No. 5, May 1987, pp. 540-552.

[La-74] L. Lamport, "A New Solution of Dijkstra's Concurrent Programming Problem, Communications of the ACM, no. 8, Vol. 17, August 1974.

[La-86a] L. Lamport, "The Mutual Exclusion Problem. Part I: A Theory of Interprocess Communication," J. ACM 33, 2 1986, pp.313-326.

[La-86b] L. Lamport, "The Mutual Exclusion Problem. Part II: Statement and Solutions," J. ACM 33, 2 1986, pp. 327-348.

[Se-83] A. Segall, "Distributed Network Protocols", IEEE Transactions on Information Theory, Vol. IT-29 no. 1, pp. 23-35, January 1983.

[SH-87] S.R. Soloway and P.A. Humblet: "On Distributed Network Protocols for Changing Topologies", preprint.

[Sz-65] E. Szekeres and G. Szekeres, "On a Problem of Schutte and Erdos", Math. Gazette 49, pp. 290-293, 1965.

[VA-86] P.M.B. Vitányi and B. Awerbuch, "Atomic Shared Register Access by Asynchronous Hardware", 27th Annual IEEE Symposium on Foundations of Computer Science, 1986, pp. 233-243.