Plausible Clocks with Bounded Inaccuracy

Brad T. Moore and Paolo A.G. Sivilotti

The Ohio State University, Columbus OH 43210, USA
{mooreb, paolo}@cse.ohio-state.edu

Abstract. In a distributed system with N processes, time stamps of size N (such as vector clocks) are necessary to accurately track potential causality between events. Plausible clocks are a family of time-stamping schemes that use smaller time stamps at the expense of some accuracy. To date, all plausible clocks have been designed to use fixed-sized time stamps, and the inaccuracy of these schemes varies from run to run. In this paper, we define a new metric, imprecision, that formally characterizes the fidelity of a plausible clock. We present a new plausible clock system that guarantees an arbitrary constant bound on imprecision. This bound is achieved by allowing time stamps to grow and shrink over the course of the computation. We verify the correctness of our algorithm, present results of a simulation study, and evaluate its performance.

1 Introduction

The events of a distributed system can be ordered by potential causality: whether one event might have affected another. Determining this ordering between events is of fundamental importance to a variety of distributed algorithms. For example, a global snapshot consists of a set of events such that no pair is causally related [1,2,3]. Cache-coherence protocols can maintain consistency by ordering updates to a shared object by potential causality [4,5,6] Resource allocation algorithms can use this relation to resolve contention for a shared resource [7,8].

Many logical time-stamping schemes exist to track potential causality between events. Lamport clocks [9], for example, time stamp each message and event with an integer, while vector clocks [10,11] time stamp each message and event with an array of integers. Although Lamport clocks require less overhead, they carry limited information: Lamport clocks order *all* events that are causally related but may also order events that are not causally related. On the other hand, the larger time stamps of vector clocks permit them to be completely accurate: Vector clocks order *all* and *only* events that are causally related.

This trade-off between space and accuracy is inherent to the problem. For a system with N processes, time stamps of size N are both sufficient and necessary for complete accuracy [12]. This result means that no time-stamping scheme can simultaneously guarantee small time stamps and perfect accuracy.

A *plausible clock* is a time-stamping system that satisfies the requirement that all events that are causally related be ordered, but not necessarily the additional requirement that *only* events that are causally related be ordered [13]. Because

P. Fraigniaud (Ed.): DISC 2005, LNCS 3724, pp. 214–228, 2005.

plausible clocks do not guarantee perfect accuracy, they can be implemented with small time stamps. Several plausible clock schemes have been developed [13,14] that use constant-sized time stamps. The accuracy achieved by these schemes varies from run to run and even over the course of a single execution. Their performance, therefore, is quantified in terms of their expected-case inaccuracy and is generally assessed through simulation.

In this paper, we introduce a new performance measure: *imprecision*. Informally, the imprecision of a time stamp is the maximum number of incorrect orderings permitted by such a stamp. Thus, imprecision reflects an upper bound on the inaccuracy of a time-stamping system. Existing plausible clock algorithms are parameterized by the size of time stamps used. For any chosen size less than N, however, the imprecision of such an algorithm can be quite high. In contrast, we describe a new algorithm that is parameterized by the amount of imprecision. This algorithm allows time stamps to grow and shrink over the course of the computation as necessary to maintain the desired level of precision. To our knowledge, this is the first guaranteed precision plausible clock algorithm.

We quantify the performance of our algorithm in two ways. Firstly, we study the expected-case time-stamp size through simulation. That is, we determine the average time-stamp size needed to maintain a given level of precision, under a variety of circumstances. Secondly, we examine the expected-case accuracy achieved by our algorithm through simulation. That is, we determine how close the actual inaccuracy comes to the upper bound reflected in the selected level of imprecision. We compare the performance of our algorithm with that of two existing plausible clock algorithms.

2 Background and Definitions

2.1 The Model

A distributed system consists of N processes. Processes communicate by message-passing, which is asynchronous, point-to-point, and fault-free. The execution of a process p_i is a finite sequence of events denoted H_i. Each event is either a *local*, *send*, or *receive* event. There is a one-to-one correspondence between send events and their matching receive events. The execution of the system is the set of all events from the individual histories, $H = (\cup i : 1 \leq i \leq N : H_i)$.[1]

The *happens before* ($\rightarrow$) relation [9] orders the events in H by their potential causal relationship. For two events $a \in H_i$ and $b \in H_j$, $a \rightarrow b$ if and only if:

1. $i = j$ and a occurs before b on p_i,
2. a is a send event and b is the corresponding receive event, or
3. there exists an event $c \in H$ such that $a \rightarrow c$ and $c \rightarrow b$.

[1] The notation we use for quantification throughout this paper is $(op\ vars : ranges : exp)$, where op is an associative and commutative operator with an identity element, $vars$ is the set of bound variables, $ranges$ is a predicate restricting the ranges of the bound variables, and exp is the expression to be quantified.

Two events are *concurrent* when neither happens before the other:

$$a \parallel b \equiv \neg(a \rightarrow b) \wedge \neg(b \rightarrow a) \ .$$

2.2 Logical Clocks

A *time-stamping system* X [13] is a tuple $(\langle S, \overset{X}{\rightarrow}\rangle, G, X.\mathbf{stamp}, X.\mathbf{tag})$, where:

S is a set of logical time values used locally (*time stamps*),
$\overset{X}{\rightarrow}$ is an irreflexive transitive relation on time stamps,
G is a set of logical time values appended to messages (*time tags*),
$X.\mathbf{stamp}$ is the time stamping function mapping events to stamps, and
$X.\mathbf{tag}$ is the tagging function mapping event time stamps to message time tags.

The relation $\overset{X}{\rightarrow}$ is irreflexive and transitive, therefore $\langle S, \overset{X}{\rightarrow}\rangle$ is a strict partial order. This strict partial order induces further relations: for $r, s \in S$,

$$\begin{aligned} r \overset{X}{=} s &\equiv r = s \\ r \overset{X}{\parallel} s &\equiv \neg(r \overset{X}{\rightarrow} s) \wedge \neg(s \overset{X}{\rightarrow} r) \wedge \neg(r \overset{X}{=} s) \ . \end{aligned}$$

For convenience, we will overload the definitions of these relations to allow them to directly compare events of H. For instance, given two events $a, b \in H$,

$$a \overset{X}{\rightarrow} b \equiv X.\mathbf{stamp}(a) \overset{X}{\rightarrow} X.\mathbf{stamp}(b) \ .$$

In practice, the function $X.\mathbf{stamp}$ is guaranteed to be locally computable by defining it inductively. First, time stamps are defined for all initial events. Then, a function on $S \times G$ is given that determines the time stamp of an event based upon the most recent local time stamp and the most recently received message time tag. When the time-stamping system is clear from context, we will omit the name and write simply **stamp** and **tag**.

2.3 Example: Vector Clocks

The vector clock is a logical clock that *characterizes* the happens before relation (i.e., $a \rightarrow b \equiv a \overset{\text{Vector}}{\rightarrow} b$). A time stamp $s \in S$ is a vector of N integers ($S = \mathbb{Z}^N$). The **stamp** function is defined inductively. The time stamp of an initial (local) event on p_i is all 0's except for the i^{th} entry, which is 1. A subsequent local or send event on p_i has the same stamp as its immediate predecessor on p_i, except the i^{th} entry is incremented. Finally, the time stamp of a receive event is the element-wise max of the current stamp (with the i^{th} entry incremented) and the incoming tag.

Time tags are identical to time stamps ($G = S$). The **tag** function is the identify function: The time tag appended to a message is the time stamp of the corresponding send event. The $\overset{\text{Vector}}{\rightarrow}$ relation is defined to be[2]

[2] The bound variables i and j will be understand to range from 1 to N and so the range can be omitted.

$$r \overset{\text{Vector}}{\rightarrow} s \;\equiv\; (\forall\, i \;::\; r[i] \le s[i]\,) \;\wedge\; (\exists\, j \;::\; r[j] < s[j]\,)\;.$$

A vector clock maintains the two important properties. First, events on a given process are mapped to a strictly increasing sequence of integers. That is, for two time stamps $r = \mathbf{stamp}(a)$ and $s = \mathbf{stamp}(b)$ on process p_i, $a \rightarrow b \;\equiv\; r[i] < s[i]$. Second, a stamp records the *most recent* happens before event from each process. For instance, consider a time stamp $r = \mathbf{stamp}(a)$ on process p_i. For each entry $r[j]$ where $j \neq i$, there exists a time stamp $s = \mathbf{stamp}(b)$ on p_j such that $s[j] = r[j]$. This event b is the most recent event on p_j that happens before a. More formally, $b \rightarrow a \;\wedge\; \neg(\exists\, c \in H_j \;::\; b \rightarrow c \;\wedge\; c \rightarrow a\,)$.

2.4 Plausible Clocks

A time-stamping system P is *plausible* if and only if it satisfies, for all $a, b \in H$,

$$a \rightarrow b \;\Rightarrow\; a \overset{P}{\rightarrow} b \tag{1}$$

$$a = b \;\equiv\; a \overset{P}{=} b\;. \tag{2}$$

Equation (1) requires that a plausible clock's ordering relation on time stamps be *consistent* with the happens before relation between events: Every pair of causally-related events is correctly ordered by the plausible clock, although some unrelated (i.e., concurrent) events may *also* be ordered. Equation (2) requires that a plausible clock's time stamps can be used to distinguish different events.

2.5 Inaccuracy

The *inaccuracy* of a plausible clock is the ratio of the number of incorrectly ordered event pairs to the number of concurrent event pairs in the system [14]. Formally, we define C as the set of concurrent pairs in the system, and M as the set of incorrectly ordered pairs. The inaccuracy of a plausible clock P on a history H, $\rho(P, H)$, is therefore defined by

$$C = \{\, (a,b) \in H \times H \;:\; a \parallel b \;:\; (a,b)\,\}$$

$$M = \{\, (a,b) \in H \times H \;:\; a \parallel b \wedge \neg(a \overset{P}{\parallel} b) \;:\; (a,b)\,\}$$

$$\rho(P,H) = \frac{|M|}{|C|}\;.$$

Accuracy can then be defined as $1 - \rho(P, H)$. Note that $\parallel$ and $\overset{P}{\parallel}$ are both symmetric, so a single pair of events is counted twice in both C and M.

2.6 Imprecision

Our goal is to create a plausible clock that can guarantee an arbitrary bound on inaccuracy. To be practical, there should be no presumption of global information

nor should the clock modify the underlying computation (e.g., by sending extra messages). Our approach is to bound the inaccuracy by controlling the maximum possible error permitted by individual time stamps. To this end, we redefine inaccuracy in terms of this error.

First, we define the *local error* of a plausible clock to be the number of mistakes it makes with respect to a given event. More precisely, it is the number of (concurrent) events that are mistakenly ordered *before* the event in question. Formally, we define the local error of P with respect to an event b by

$$\delta(P, H, b) = |\{\, a \in H \;:\; a \parallel b \wedge a \xrightarrow{P} b \;:\; a \,\}| \;.$$

We can now define the total number of mistakes in terms of the local error for each event (note, we do not double-count pairs in the definition of local error):

$$|M| = 2 * (\sum b \in H \;::\; \delta(P, H, b)) \;.$$

Therefore, inaccuracy can be written as

$$\rho(P, H) = 2 * \frac{(\sum b \in H \;::\; \delta(P, H, b))}{|C|} \;.$$

Our definition of $\rho(P, H)$ is still problematic. We would like to define inaccuracy in terms of the local error per event; however, it is currently the ratio between the sum of local error and the total number of concurrent event pairs. To this end, we define $\epsilon(H)$ as the ratio between the total number of concurrent pairs and the total number of events:

$$\epsilon(H) = 1/2 * \frac{|C|}{|H|} \;.$$

For computations that exhibit regular communication patterns and whose processes are not partitioned, the value of $\epsilon(H)$ remains constant as H is extended with new events. If the processes were partitioned (say one process ceases to communicate), this ratio would increase without bound as H is extended with new events. For the remainder of the paper, we will assume fault-free executions where all processes actively communicate within the system. Rewriting the total number of concurrent pairs in terms of this concurrency ratio, we have

$$\rho(P, H) = 1/\epsilon(H) * \frac{(\sum b \in H \;::\; \delta(P, H, b))}{|H|} \;.$$

Since we assume that $\epsilon(H)$ is a constant, we need only to bound the mean value of δ in order to bound the inaccuracy. Unfortunately, we cannot use δ directly in our algorithm; the **stamp** function is defined inductively over time stamps and not histories. Therefore, we define a new metric that is based on time stamps and hence can be used directly by a plausible clock to reason about fidelity. We call this metric *imprecision*. The imprecision of a time stamp generated by a plausible clock is an upper bound on the number of ordering mistakes made for

an event with that time stamp. More formally, let $\mathcal{H}(P, s)$ be the set of histories for which the plausible clock P generates the time stamp s:

$$\mathcal{H}(P, s) = \{\, H \;:\; (\exists\, a \in H \;::\; P.\mathbf{stamp}(a) = s\,) \;:\; H \,\}\ .$$

Imprecision, $\psi(P, s)$, then is defined by

$$\psi(P, s) = (\,\mathbf{Max}\, H \in \mathcal{H}(P, s),\ a \in H \;:\; P.\mathbf{stamp}(a) = s \;:\; \delta(P, H, a)\,)\ .$$

Intuitively, imprecision is the worst-case value of δ for an event with a given time stamp. Note that imprecision is independent of history and therefore is a function of the information contained within a time stamp. If we guarantee that all time stamps generated during a computation have an imprecision below some arbitrary bound, K, then the mean value of δ is also below that bound. The resulting bound on inaccuracy is given by

$$\rho(P, H) \leq 1/\epsilon(H) * \frac{(\sum b \in H \;::\; \psi(P, P.\mathbf{stamp}(b))\,)}{|H|} \leq K/\epsilon(H)\ .$$

3 A Guaranteed Precision Plausible Clock

3.1 Logical Time Intervals

With vector clocks, the time stamps of events on process p_i all differ in their i^{th} entry. This one entry orders these events and distinguishes between them. The other entries serve a different purpose: Each one uniquely identifies the most recent happens-before event on the corresponding remote process. Our approach is conceptually similar. Time stamps are vectors where the i^{th} entry orders and distinguishes between events on p_i, while the other entries indicate the most recent happens-before events on remote processes. The difference is that a *range* of values, rather than a single one, is used as an entry in the array and hence the most recent happens-before events are not uniquely identified.

At the core of our algorithm is the concept of a time *interval*. A time interval is a tuple $\langle beg, end\rangle$ where beg and end are integers and $beg \leq end$. Unlike the integer entry of vector clocks which corresponds to a single event, a time interval corresponds to a set of events. The event of interest is within this range. Thus, when comparing two time intervals, we can conclude something about the ordering of the respective events of interest only when the ranges do not overlap. The ordering between two intervals m and n is given by

$$\begin{aligned}
m \overset{\text{int}}{<} n &\equiv m.end < n.beg \\
m \overset{\text{int}}{\approx} n &\equiv \neg(m \overset{\text{int}}{<} n) \wedge \neg(n \overset{\text{int}}{<} m) \\
m \overset{\text{int}}{\lessapprox} n &\equiv (m \overset{\text{int}}{<} n) \vee (m \overset{\text{int}}{\approx} n)\ .
\end{aligned}$$

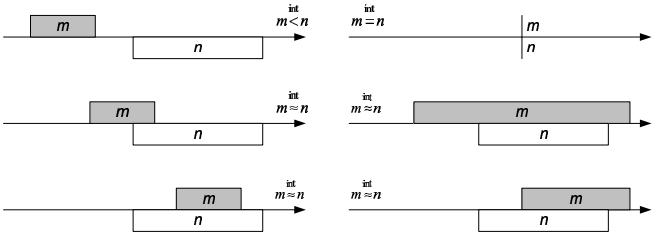

Fig. 1. Examples of time interval comparison

We define a *precise* interval to be one in which the begin and end points are equal. In the case of precise intervals, an overlap reflects exact equality:

$$\begin{aligned} \mathbf{precise}(m) &\equiv m.beg = m.end \\ m \overset{\text{int}}{=} n &\equiv \mathbf{precise}(m) \wedge \mathbf{precise}(n) \wedge m = n \ . \end{aligned}$$

3.2 Definition of S and G

A time stamp $s \in S$ is a vector of N time intervals. Let $\mathbf{min\_beg}(s)$ ($\mathbf{min\_end}(s)$) be the minimum begin (end) point in s. Like vector clocks, our time stamps map the events of a given process to an increasing sequence. That is, for two time stamps $r = \mathbf{stamp}(a)$ and $s = \mathbf{stamp}(b)$ on process p_i,

$$\mathbf{precise}(r[i]) \wedge \mathbf{precise}(s[i]) \tag{3}$$

$$a \rightarrow b \equiv r[i] \overset{\text{int}}{<} s[i] \ . \tag{4}$$

A time stamp s also satisfies several additional properties. First, all imprecise intervals of s share the same end value. Second, all precise intervals of s are greater than the imprecise intervals. Both properties are captured by:

$$(\forall i \ : \ \neg\mathbf{precise}(s[i]) \ : \ s[i].end = \mathbf{min\_end}(s) \ ) \ . \tag{5}$$

Like time stamps, a time tag t is also a vector of N time intervals. It satisfies all the properties of time stamps and, in addition, the property that imprecise intervals have the same begin point:

$$(\forall i \ : \ \neg\mathbf{precise}(s[i]) \ : \ s[i].beg = \mathbf{min\_beg}(s) \ ) \ . \tag{6}$$

Thus, $G \subset S$. See Fig. 2 for an illustration of a time stamp and a time tag.

Ordering. The comparison of time stamps in our algorithm is similar to that of vector clocks. The $\overset{P}{\rightarrow}$ relation is formally defined by:

$$r \overset{P}{\rightarrow} s \ \equiv \ (\forall i :: r[i] \overset{\text{int}}{\lessapprox} s[i] \ ) \ \wedge \ (\exists j :: r[j] \overset{\text{int}}{<} s[j] \ ) \ .$$

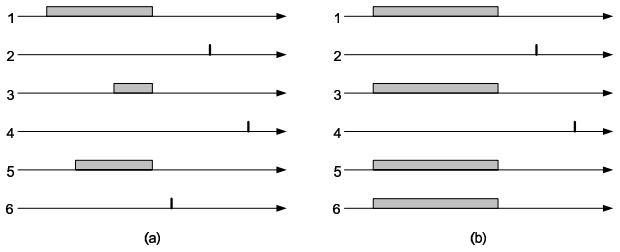

Fig. 2. A time stamp (a) and a time tag (b) in a system with 6 processes. Imprecise entries in a time tag share a common interval.

Space Complexity. Since precise intervals can be encoded with a single integer and all imprecise intervals share the same end point, time stamps can be encoded with $N+1$ integers (i.e., N begin values and one common end value).

For a time tag with R precise intervals, $R \log N$ bits are required to encode the mapping between precise intervals and their respective processes. Since all imprecise intervals are the same, a time tag requires $R(L + \log N) + 2L$ bits, where L bits are used to encode a single integer.

Imprecision. The size of the intervals determines the imprecision of a time stamp. In any given history, the local error possible for a stamp s with respect to some process p_i is the size of the i^{th} interval of s. Hence, the imprecision is the sum of these interval lengths:

$$\psi(s) = (\sum i :: s[i].end - s[i].beg) \ .$$

3.3 Definition of Stamp

The **stamp** function is defined inductively for process p_i as follows. Initially, all time stamp entries are precise intervals equal to $\langle 0, 0 \rangle$ except for the i^{th} entry which is set to $\langle 1, 1 \rangle$. During a local/send event, the i^{th} entry is incremented. Thus, if r is the old stamp on p_i, the new stamp s is defined by

$$\begin{aligned} &\mathbf{precise}(s[i]) \wedge s[i].end = r[i].end + 1 \\ &(\forall j : j \neq i : s[j] = r[j]) \ . \end{aligned}$$

Upon receiving a time tag, the max of the *beg* and *end* points of each entry is taken and the i^{th} entry is incremented. Thus, if r is the old stamp on p_i and t is the time tag of the incoming message, the new stamp s is defined by

$$\begin{aligned} &\mathbf{precise}(s[i]) \wedge s[i].end = max(r[i].end, t[i].end) + 1 \\ &(\forall j : j \neq i : s[j] = \langle \max(r[j].beg, t[j].beg), \max(r[j].end, t[j].end) \rangle) \ . \end{aligned}$$

Algorithm 1: **stamp**

Data: r is old stamp on p_i, s is new stamp on p_i, t is incoming tag
INITIALLY:
  for $j := 1$ **to** N **do** $s[j] := \langle 0, 0 \rangle$
  $s[i] := \langle 1, 1 \rangle$

LOCAL or SEND EVENT:
  for $j := 1$ **to** N **do** $s[j] := r[j]$
  $s[i].end := s[i].end + 1$
  $s[i].beg := s[i].end$

RECEIVE EVENT:
  for $j := 1$ **to** N **do**
    $s[j].end := \max(r[j].end, t[j].end)$
    $s[j].beg := \max(r[j].beg, t[j].beg)$
  end
  $s[i].end := s[i].end + 1$
  $s[i].beg := s[i].beg$

3.4 Definition of Tag

The goal of the **tag** algorithm is to construct the smallest possible time tag while not exceeding its bound on imprecision. Informally, the time tag is constructed by iteratively adding the greatest precise intervals until the error of the time tag is below the imprecision bound, K. The common imprecise interval of the time tag is formed by taking the max *end* value and the min *beg* value of the remaining intervals not in the time tag. The pseudo-code for **tag** is Algorithm 2. The function **ith_max**(i, s) returns the index of the i^{th} largest precise interval in time stamp s.

3.5 Example

Figure 3 depicts two examples of a process executing three events: a local event, a receive event, and a send event. In these example, the process is p_3, there are a total of 6 processes, and the bound on imprecision is 30. Observe that the time stamps satisfy property (5), while the time tags satisfy (5) and (6). Also note that the imprecision of each stamp (and tag) is less than the bound.

When a message is received, the new stamp is calculated as the element-wise max of the old stamp and the incoming message. The result of this operation is a valid time stamp (i.e., it satisfies (5)). In Fig. 3(a) the imprecision of the local stamp increases as the result of an incoming message (from 6 to 15), while in Fig. 3(b) it decreases (from 6 to 3).

Message tags are constructed from time stamps using the largest possible common interval such that the imprecision of the tag is less than the bound. In

Algorithm 2: **tag**

Data: r is the time stamp of the send event, t is the outgoing tag
for $j := 1$ **to** N **do** $t[j] := \langle 0, 0 \rangle$
$minbeg :=$ (**Min** $j : 1 \leq j \leq N : r[j].beg$)
$i := 1$
$k :=$ **ith_max**(i, r)
while $(N - i + 1) * (r[k].end - minbeg) > K$ **do**
  $t[k] := r[k]$
  $i := i + 1$
  $k :=$ **ith_max**(i, r)
end
for $j := 1$ **to** N **do**
  if $t[j] = \langle 0, 0 \rangle$ **then**
    $t[j].end = r[k].end$
    $t[j].beg = minbeg$
  end
end

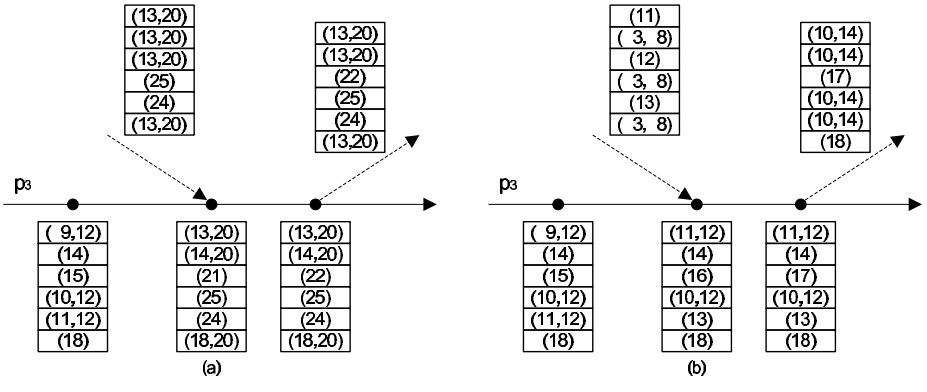

Fig. 3. Sample executions illustrating **stamp** and **tag**

Fig. 3(b), for example, entries 1,2,4, and 5 are part of the common interval, giving an imprecision of $4 * (14 - 10) = 16$. The next largest common interval would include entry 3 and so would be $\langle 10, 17 \rangle$. The resulting imprecision, however, would be $5 * (17 - 10) = 35$ which exceeds the bound.

4 Proofs of Correctness

In this section, we sketch the proof of correctness for our time-stamping scheme. Only the main theorems and lemmas are given, while details are available in [15].

We first define two operators on time stamps, then use these operators to show plausibility and boundedness of imprecision.

4.1 The Join ($\bowtie$) and Expand Operators

We define the *join* ($r \bowtie s$) of two time stamps by

$$(\forall i :: \quad (r \bowtie s)[i].beg = \mathbf{max}(r[i].beg, s[i].beg) \\ \wedge\ (r \bowtie s)[i].end = \mathbf{max}(r[i].end, s[i].end)\ )\ .$$

The **stamp** function for receive events can be redefined in terms of join. Figure 4 is a graphical representation of this operator.

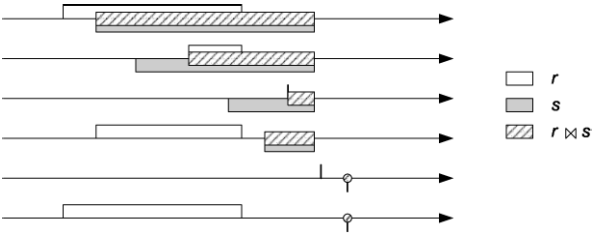

Fig. 4. The join ($\bowtie$) operator

An important property is that S is closed under join. That is, the join of two time stamps, r and s, satisfies (5). This closure property allows us to prove, inductively, that all stamps generated by **stamp** are indeed elements of S and all tags generated by **tag** are indeed elements of G.

Next, we define the **expand** of a time stamp by

$$\mathbf{expand}(s) = (\sum i : \neg\mathbf{precise}(s[i]) : \mathbf{min\_end}(s) - \mathbf{min\_beg}(s)\ )\ .$$

Thus, $\mathbf{expand}(s)$ is the size of the *smallest* (i.e., shortest common interval) time tag that can be generated from s.

We can show that the join operation does not increase the worst-case error of the system. That is, the **expand** of the join of two time stamps is less than or equal to the max of their respective **expand**'s:

$$\mathbf{expand}(r \bowtie s) \leq \mathbf{max}(\mathbf{expand}(r), \mathbf{expand}(s))\ .$$

This property allows us to prove inductively that the **expand** of all stamps generated by **stamp** and all tags generated by **tag** is below a specified bound.

4.2 Proof of Plausibility

In order to prove that P is plausible, we begin by showing that the $\xrightarrow{P}$ relation holds between pairs of events on the same process as well as send-receive pairs.

Theorem 1. *If a and b both occur on a process p_i, $a \rightarrow b \ \Leftrightarrow\ a \xrightarrow{P} b$.*

Theorem 2. *If* a *is a send event and* b *is the corresponding receive,* $a \xrightarrow{P} b$.

Another property of P is that the i^{th} (precise) interval can be used to order events on p_i with other events.

Theorem 3. *If* a *and* b *occur on processes* p_i *and* p_j *respectively,* $a \neq b \wedge \mathbf{stamp}(a)[i] \overset{\text{int}}{=} \mathbf{stamp}(b)[i] \Rightarrow a \rightarrow b$, *and* $\mathbf{stamp}(a)[i] \overset{\text{int}}{<} \mathbf{stamp}(b)[i] \Rightarrow a \rightarrow b$.

Theorem 4. P *is plausible.*

Proof. There are two proof obligations: properties (1) and (2) of plausible clocks.

Property (1): $a \rightarrow b \Rightarrow a \xrightarrow{P} b$. Assume $a \rightarrow b$. Therefore, there exists a chain of events $c_0, c_1, ..., c_n$ where $c_0 = a$ and $c_n = b$ and $(\forall k : 0 \leq k < n : c_k \rightarrow c_{k+1})$ such that adjacent pairs in this chain are either on the same process or matching send/receive events. From Theorems 1 and 2, we have $(\forall k : 0 \leq k < n : c_k \xrightarrow{P} c_{k+1})$. Furthermore, since end intervals are non-decreasing along this chain, $\xrightarrow{P}$ is transitive along this chain. Therefore, $a \xrightarrow{P} b$.

Property (2): $a = b \equiv a \overset{P}{=} b$. The forward direction follows immediately from the definition of $\overset{P}{=}$. For the reverse direction, assume $a \overset{P}{=} b$. Let p_i (p_j) be the process on which a (b) occurs and let r (s) be $\mathbf{stamp}(a)$ ($\mathbf{stamp}(b)$). Since $r \overset{P}{=} s$, $r[i] \overset{\text{int}}{=} s[i]$ and $r[j] \overset{\text{int}}{=} s[j]$. From (3), both $r[i]$ and $s[j]$ are precise. From Theorem 3, neither $r[j]$ nor $s[i]$ are precise. Since precise intervals are greater than imprecise intervals, these two intervals cannot be simultaneously equivalent unless $a = b$. □

4.3 Proof That Imprecision is Bounded

Theorem 5. $\psi(\mathbf{stamp}(a)) = (\sum i :: \mathbf{stamp}(a)[i].end - \mathbf{stamp}(a)[i].beg)$

Proof. Consider an event b that occurs on process p_j. From Theorem 3, a and b are correctly ordered when the j^{th} intervals of their stamps are ordered by $\overset{\text{int}}{=}$ or $\overset{\text{int}}{<}$. Hence, ordering mistakes can only occur when the j^{th} intervals are related by $\overset{\text{int}}{\approx}$. From (3) and (4), at most $\mathbf{stamp}(a)[j].end - \mathbf{stamp}(a)[j].beg$ such events exist. Thus, $(\sum i :: \mathbf{stamp}(a)[i].end - \mathbf{stamp}(a)[i].beg)$ is an upper bound on the number of incorrectly related events ($\xrightarrow{P}$ but not $\rightarrow$). This bound is tight since, for any time stamp s a history can be constructed in which there is an event whose stamp is s and for which $(\sum i :: \mathbf{stamp}(a)[i].end - \mathbf{stamp}(a)[i].beg)$ ordering mistakes are made. □

5 Experimental Evaluation

We consider a client-server system with 2 clients and 98 servers. A client performs local events and sends messages to a random server. While waiting for a response, a client performs only local events. Servers reply to messages in FIFO order, and only perform local events if there are no outstanding requests from clients. Event arrivals follow a negative exponential distribution.

For our analysis, we consider only events from the *middle* of the computation (i.e., events that have are causally related to some event from each process which, in turn, is also causally related to some event from each process). We exclude events at the beginning and end since plausible clocks (including our own) perform better during the startup of a computation than in steady-state.

The two primary characteristics we evaluate are: the relationship between message size and inaccuracy; and the difference between the (worst-case) inaccuracy bound determined by imprecision and the actual (expected-case) inaccuracy achieved. The former allows a comparison between our algorithm and other plausible clock algorithms in terms of trading off accuracy for message overhead. The latter is unique to our algorithm, where imprecision can be controlled.

Figure 5 depicts the relationship between message size and resulting inaccuracy. The plausible clocks considered are R-Entries Vector (REV) and Comb (a combination of REV and k-Lamport) [13]. We fix the k-Lamport component of Comb to 5 entries while varying the size of its REV component. The results show that, on average, our algorithm (labeled "Common Interval" in the figure) yields better accuracy than either of the other two.

Although this figure compares these plausible clocks directly, it is worth remembering that these clocks differ in a fundamental way: Our algorithm does not guarantee a constant message overhead, while other plausible clock algorithms do not guarantee any level of accuracy.[3] Therefore, while it is useful to compare their performance in trading off accuracy for message overhead, the choice of which algorithm to use will likely be driven by the primary performance property being optimized.

Figure 6 depicts the relationship between the inaccuracy bound (derived from imprecision) and the observed inaccuracy. We see that the resulting inaccuracy is significantly less than the inaccuracy bound. Imprecision measures the worst-case error per time stamp. Actual runs, however, may not generate events that result in error equal to each time stamp's imprecision. Thus, while our algorithm provides a guarantee on the worst case behavior, it does not do this at the expense of degrading the expected case inaccuracy.

6 Related Work

Several algorithms have been proposed as scalable solutions to vector clocks. For instance, in [16] Baldoni and Melideo proposed k-dependency vectors. Their al-

[3] Notice that the data points for our algorithm have error bars in the horizontal axis since time tag size varies.

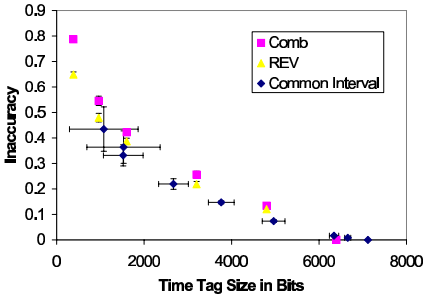

Fig. 5. Performance comparison with other plausible clocks

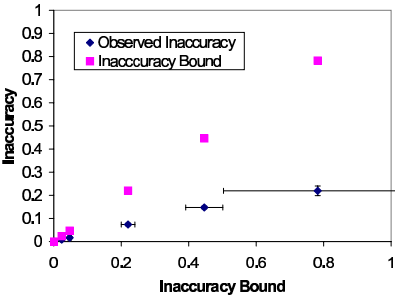

Fig. 6. Actual observed inaccuracy compared to upper bound

gorithm affixes a constant-size vector of integers to application messages. The trade-off for this approach is that extra computation may be required to detect the causal relationship between events. The algorithm requires a dedicated checker process to determine the causal order of events.

The definition of plausible clocks was formalized in [13], where it was also shown that such clocks could be combined to improve accuracy. Two plausible clock algorithms were presented: REV and k-Lamport, along with their combination, Comb. In [13,17], the performance of Comb was analyzed through simulation. The results of those studies showed good performance of Comb and also the dependency of that performance on several factors (e.g., local history size, communication pattern, system size). However, formal analysis of the expected behavior of a plausible clock algorithm was left for future work.

NUREV clocks were proposed in [14]. This plausible clock uses a fixed vector size and a dynamic mapping of processor ids to vector entries. Several mappings were proposed to minimize the ordering errors produced by this clock, and hence maximize the expected accuracy. This work did not consider the cost of encoding the dynamic map in the time tag.

Unlike these other approaches, our notion of imprecision–since it reflects a worst-case bound–permits the evaluation of plausible clock performance independent of a particular history or set of histories.

7 Conclusion

The contribution of this work is threefold. Firstly, we have defined a new metric, *imprecision*, which quantifies the worst-case accuracy of a plausible clock time-stamping system. This metric characterizes the system itself, independent of any particular history. Existing plausible clocks are parameterized by message time tag size and (even those with good average-case accuracy) have unbounded imprecision. Our second contribution is a new plausible clock algorithm which is parameterized by imprecision. This algorithm guarantees a maximum, bounded imprecision by varying the size of time tags as needed during a computation.

Finally, we provide an experimental evaluation of this algorithm's performance. We find that the expected message size for our algorithm compares favorably with existing plausible clocks. We also note that since imprecision is a conservative upper bound on inaccuracy, the actual inaccuracy for a given history may be considerably less than this guaranteed bound.

References

1. Chandy, K.M., Lamport, L.: Distributed snapshots: Determining global states of distributed systems. ACM Transactions on Computer Systems **3** (1985) 63–75
2. Netzer, R., Xu, J.: Necessary and sufficient conditions for consistent global snapshots. IEEE Transactions on Parallel and Distributed Systems **6** (1995) 165–169
3. Elnozahy, M., Alvisi, L., Wang, Y.M., Johnson, D.B.: A survey of rollback-recovery protocols in message passing systems. Technical Report CMU-CS-96-181, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA (1996)
4. Ahamad, M., Neiger, G., Burns, J.E., Kohli, P., Hutto, P.W.: Causal memory: Definitions, implementation, and programming. Distributed Computing **9** (1995) 37–49
5. Prakash, R., Raynal, M., Singhal, M.: An adaptive causal ordering algorithm suited to mobile computing environments. Journal of Parallel and Distributed Computing **41** (1997) 190–204
6. Fernández, A., Jiménez, E., Cholvi, V.: On the interconnection of causal memory systems. In: Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing. (2000) 163–170
7. Ricart, G., Agrawala, A.K.: An optimal algorithm for mutual exclusion in computer networks. Communications of the ACM **24** (1981) 9–17
8. Maekawa, M.: A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. ACM Transactions on Computer Systems **3** (1985) 145–159
9. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Communications of the ACM **21** (1978) 558–565
10. Fidge, C.J.: Timestamps in message-passing systems that preserve the partial ordering. In: Proc. of the 11^{th} Australian Computer Science Conf. (1988) 55–66
11. Mattern, F.: Virtual time and global states of distributed systems. In: Proceedings of the International Workshop on Parallel & Distributed Algorithms. Elsevier Science Publishers B. V. (1989) 215–226
12. Charron-Bost, B.: Concerning the size of logical clocks in distributed systems. Information Processing Letters **39** (1991) 11–16
13. Torres-Rojas, F.J., Ahamad, M.: Plausible clocks: constant size logical clocks for distributed systems. Distributed Computing **12** (1999) 179–196
14. Gidenstam, A., Papatriantafilou, M.: Adaptive plausible clocks. In: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04), IEEE Computer Society (2004) 86–93
15. Moore, B.T.: Plausible clocks with bounded inaccuracy. Master's thesis, The Ohio State University (2005) available as technical report OSU-CISRC-7/05-TR52.
16. Baldoni, R., Melideo, G.: k-dependency vectors: A scalable causality-tracking protocol. In: Proceedings of the 11th Euromicro Conference on Parallel, Distributed and Network-Based Processing. (2003)
17. Torres-Rojas, F.J.: Performance evaluation of plausible clocks. In: Proceedings of the 7th Euro-Par Conference. (2001) 476–481