Breakpoints and Time in Distributed Computations

Twan Basten

Dept. of Computing Science, Eindhoven University of Technology, The Netherlands
email: tbasten@win.tue.nl

Abstract. This paper investigates how vector time can be used to set breakpoints in distributed computations for the purpose of analyzing and debugging distributed programs. A breakpoint is represented by a set of events in one or more processes. One interesting state in which a distributed computation can be halted is the *earliest* global state reflecting all events in a breakpoint. A simple expression in terms of vector time is derived to determine this state. Another state of interest is the global state reflecting only events preceding any of the breakpoint events, but reflecting none of the breakpoint events themselves. Two alternative expressions are presented for this state. The first one is in terms of vector time and a derived notion called reversed vector time. The second expression uses vector time and the convex closure of a set of events.

Key words: Breakpoints – Vector time – Global states – Causality – Distributed debugging

1 Introduction and Related Work

Despite considerable progress over the last decades, designing distributed applications is still more an art than a science. A correct and efficient implementation is, therefore, hardly ever achieved without extensive testing and debugging.

Analyzing and debugging distributed computations is an area that is itself complex and in the early stages of its development. Due to concurrency, lossy channels, and unpredictable communication delays, distributed computations are inherently non-deterministic and it is impossible to determine the global state of a computation reliably. To overcome the first problem, a distributed debugging environment should provide a *deterministic replay facility.* During the original execution, information essential for a deterministic replay is collected, minimizing interference with the non-deterministic program behavior. Deterministic replays will then be used to analyze program behavior. To overcome the need for determining the global state of a distributed computation, many researchers [3, 8, 20, 21] have chosen an *event-based* approach. Event information can be captured relatively easily compared to state information. In addition, states of a computation can always be restored from event information.

What is important in an event-based approach to debugging is how events are causally related to each other. An event might causally *precede* another event or two events might be causally unrelated. In the latter case, they are said to be *concurrent*.

An important feature of a debugging environment is the ability to set breakpoints in a computation. This allows the programmer to halt the computation and examine its state at interesting points. The two basic issues involved in designing a breakpoint facility are specifying and detecting breakpoints, and halting the computation in a meaningful global state. In this paper, we only discuss the second aspect. This does not mean that the first aspect is not important. On the contrary, specification and detection of breakpoints is at least as important as halting the computation and it definitely is a complex task. However, the literature already suggest several approaches. In [7, 16], algorithms are given for detecting arbitrary predicates in an event-based environment. In [1, 3, 12, 15, 19], specification formalisms and detection algorithms are described based on the notion of behavioral patterns. A behavioral pattern is a specification of a set of events and the causal relations among them. In both approaches a breakpoint can always be translated to a *set of events*. Therefore, in this paper a breakpoint is considered to be an arbitrary set of events. Given a set of events, we are interested in global states of the computation that might give the programmer interesting information about the breakpoint events. This paper intends to give a general strategy for halting a computation in such states that can be implemented in many distributed-debugging environments.

In sequential computations, a breakpoint has an implied reference to physical time. The computation is suspended as soon as a breakpoint condition is satisfied. This means that the computation is halted in the *earliest* state in which the condition is true. In distributed computations, there is no global notion of time. In particular, it is not straightforward to define the notion of earliest state.

Miller and Choi [19] and Haban and Weigel [12] use behavioral patterns to specify breakpoints. As soon as an occurrence of a pattern is detected, a halting algorithm is initiated. They use halting algorithms based on Chandy and Lamport's global snapshot algorithm [4]. As soon as the occurrence of an event in some process completes the detection of a breakpoint, a halting message is sent to all neighboring processes and the process itself is suspended. Every process that receives a halting message forwards it to its neighbors and then suspends its execution. In this way, every process will, eventually, be suspended. Obviously, the global state in which the computation is halted is, in general, not the *earliest* global state.

This observation has motivated Fowler and Zwaenepoel [11] to introduce the notion of *causal distributed breakpoints* as a natural extension of breakpoints in sequential programs. A causal distributed breakpoint consists of a single event. The global state in which the computation is restored is the earliest state that exactly reflects all events preceding the breakpoint. Manabe and Imase [15] propose a similar approach for patterns slightly more general than a single event, but still fairly restricted.

In this paper, we use vector time [10, 18] to enhance the results of Fowler and Zwaenepoel, and Manabe and Imase. Vector time can be used to represent global states of a distributed computation. Interesting global states can be found and restored using the following general strategy.

During the original execution, event information is collected, minimizing perturbation. Afterwards, vector timestamps are computed and stored with the event information. Breakpoint detection takes place in the first replay, and, during this process, vector timestamps are used to calculate expressions for global states that might be of interest to the programmer. Such states can then be restored in one additional replay. The whole process of finding and restoring a meaningful state of a computation thus takes two replays. The strategy outlined here can be used with any kind of breakpoint-detection algorithm.

The research described in this paper is part of a project at the University of Waterloo to design and implement a distributed debugger [21]. The current prototype records event information for the purpose of visualization. It has tools for visualizing events and their causal relationships. It also has some tools for determining and visualizing abstract program behavior [13]. Since most tools use vector time, much effort is put into an efficient algorithm for calculating and storing timestamps. An earlier version of the prototype also had a replay facility. It is planned to add such a facility to the current prototype as well. An implementation of the strategy described above is then straightforward. In a first implementation, breakpoint events could be selected manually by the programmer using the visualization tools of the debugger.

The paper is organized as follows. In Section 2, a model of distributed computations is given. Section 3 introduces the notions of global state and time. In Section 4, an expression in terms of vector time is derived for the earliest global state reflecting a set of breakpoint events. Section 5 presents two alternative expressions for another state that might be of interest to the programmer, namely the global state reflecting only events preceding any of the breakpoint events, but reflecting none of the breakpoint events themselves. The first expression for this state is in terms of vector time and a derived notion called reversed vector time. The second expression uses vector time and the convex closure of a set of breakpoint events. The appendix contains some formal proofs of results in Section 5. Finally, Section 6 summarizes the results.

2 Distributed Computations

The model of distributed computations used in this paper is based mainly on definitions of Mattern [17, 18], Charron-Bost [5, 6], and Fidge [10]. The precedence relation, which is defined below, essentially is the "happens before" relation introduced by Lamport [14].

A distributed computation consists of a set of local computations, one for each process. For the sake of simplicity, it is assumed that the number of processes is constant and known in advance. Note that in the context of a distributed debugger providing a replay facility, this is a valid assumption. The set of process identifiers, $\{0, \ldots, N-1\}$, where N is the number of processes, is denoted $\mathcal{P}$. The set of events E is the union of N mutually disjoint sets of events, $E_0, \ldots, E_{N-1}$, one for each process. It is assumed that E is *finite*. Again, in the context of debugging, this is not a real restriction.

A message communication is modeled by two events, a send event and a corresponding receive event. Two disjoint sets of events, $\mathcal{S}, \mathcal{R} \subseteq E$, denote the sets of send and receive events. A left- and right-unique relation, $\Gamma \subseteq \mathcal{S} \times \mathcal{R}$, relates send events to receive events. Every receive event has a corresponding send event. The converse is not necessarily true. This means that messages might be lost or might still be in transit. All message communications in Γ are asynchronous, except for a subset, Γ_s, that denotes the set of synchronous communications.

In our model, processes are sequential. Therefore, we assume that for every $i \in \mathcal{P}$, the set of events E_i is totally ordered by a relation $\prec_i$. The relation $\prec_l$ is the union of all $\prec_i$ and expresses the local ordering of events. The precedence relation $\preceq$ is the smallest reflexive and transitive relation that satisfies the following two conditions.

C1 The relation $\prec_l \cup \Gamma$ is a subset of $\preceq$.
C2 For every $(s, r) \in \Gamma_s$ and $e \in E \setminus \{s, r\}$, $e \preceq s \Leftrightarrow e \preceq r$ and $s \preceq e \Leftrightarrow r \preceq e$.

Since cyclic causal relationships cannot occur in a distributed computation, we require that the precedence relation is a partial order. It extends the "happens before" relation as defined by Lamport [14] to synchronous communication in a natural way. The condition **C2**, originally given by Fidge [9, 10], means that a synchronous communication can be interpreted as if it occurred instantaneously.

A distributed computation can be visualized in a so-called *time diagram*. In Figure 1, the vertical lines represent processes. Time increases along these lines from top to bottom. Events are depicted as dots. The arrows represent the relation Γ. Horizontal arrows represent synchronous communications.

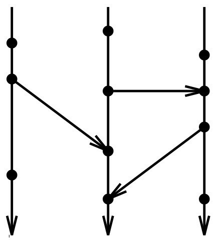

Fig. 1. The time diagram of a distributed computation.

3 Global States and Time

In this section, we formalize the notion of global states of a distributed computation and introduce a notion of time. The definitions and results in this section are due to Mattern [17, 18]. The event-based equivalent of a global state is a so-called *cut*.

Definition 3.1. (Cut) A set $C \subseteq E$ is called a *cut* of E if and only if for all events $e_0 \in C$ and $e_1 \in E$, $e_1 \preceq_l e_0 \Rightarrow e_1 \in C$, where $\preceq_l$ is the reflexive closure of the local ordering $\prec_l$. The set of all cuts is denoted $C_{\preceq_l}$.

If a cut contains some event in some process, then it must also contain *all preceding events in the same process.* In this way, a cut uniquely defines the local state for each process and, therefore, the global state of the computation. A cut can be seen as a single line in a time diagram dividing it into two parts. All events above the line belong to the cut.

Not every global state defined by the set of cuts can actually occur during the computation. For example, a cut can contain a receive event in one process and not contain the corresponding send event in some other process. Therefore, the notion of *consistent* cuts is introduced. If a consistent cut contains some event, then it also contains *all preceding events.*

Definition 3.2. (Consistent cut) A set $C \subseteq E$ is called a *consistent cut* of E if and only if for all events $e_0 \in C$ and $e_1 \in E$, $e_1 \preceq e_0 \Rightarrow e_1 \in C$. The set of all consistent cuts of a distributed computation is denoted $C_{\preceq}$.

Cuts and consistent cuts have a well-known mathematical structure. The proofs of the following two theorems are straightforward and, therefore, omitted.

Theorem 3.3. (Structure of cuts) *The pair* $(C_{\preceq_l}, \subseteq)$ *is a complete lattice. The infimum and supremum of sets of cuts are defined by* $\cap$ *and* $\cup$ *respectively.*

Theorem 3.4. (Structure of consistent cuts) *The pair* $(C_{\preceq}, \subseteq)$ *is a complete lattice.*

The ordering relation $\subseteq$ on cuts can be interpreted as "earlier." To formalize this interpretation, we introduce the following notion of time.

Definition 3.5. (Global time of a cut) The function $\mathcal{T} : C_{\preceq_l} \rightarrow \mathbb{N}^N$ defines the global time of a cut. For any cut C, component i, where $0 \leq i < N$, of the time vector is defined as $\mathcal{T}.C.i = |C \cap E_i|$. The set of all time vectors of a computation $\{\mathcal{T}.C \mid C \in C_{\preceq_l}\}$ is denoted $T_{\preceq_l}$.

The ordering $\leq$ (earlier) on time vectors is defined as follows. For two vectors $t_0, t_1 \in \mathbb{N}^N$, $t_0 \leq t_1 \Leftrightarrow t_0.i \leq t_1.i$ for all i, $0 \leq i < N$.

Corollary 3.6. *For any cuts* $C_0, C_1 \in C_{\preceq_l}$, $C_0 \subseteq C_1 \Leftrightarrow \mathcal{T}.C_0 \leq \mathcal{T}.C_1$.

Formally, Corollary 3.6 states that the function $\mathcal{T}$ is an isomorphism between the two complete lattices $(C_{\preceq_l}, \subseteq)$ and $(T_{\preceq_l}, \leq)$. This results in the following two theorems.

Theorem 3.7. (Structure of time vectors) *The pair* $(T_{\preceq_l}, \leq)$ *is a complete lattice that is isomorphic to the lattice* $(C_{\preceq_l}, \subseteq)$.

Let $T_{\preceq}$ denote the set of consistent time vectors $\{\mathcal{T}.C \mid C \in C_{\preceq}\}$.

Theorem 3.8. (Structure of consistent time vectors) *The pair* $(T_{\preceq}, \leq)$ *is a complete lattice that is isomorphic to* $(C_{\preceq}, \subseteq)$.

Theorem 3.7 implicitly defines the infimum and supremum of time vectors corresponding to a set C of cuts as $\mathcal{T}.(\cap\, c : c \in C : c)$ and $\mathcal{T}.(\cup\, c : c \in C : c)$ respectively. It is easy to verify that the former is equal to the componentwise minimum of the time vectors corresponding to the cuts in C. The latter is equal to the componentwise maximum of these time vectors. Let the quantifier INF on time vectors be defined as the componentwise minimum, and the quantifier SUP as the componentwise maximum. Consequently, for some set of cuts C, $\mathcal{T}.(\cap\, c : c \in C : c) = (\text{INF}\, c : c \in C : \mathcal{T}.c)$ and $\mathcal{T}.(\cup\, c : c \in C : c) = (\text{SUP}\, c : c \in C : \mathcal{T}.c)$.

Time vectors provide an efficient representation of cuts and, therefore, global states. The goal in the remaining sections is to identify interesting cuts for some given set of breakpoint events, and derive expressions for their global time. Such expressions can be used to restore a global state using a replay mechanism.

4 The Earliest Global State Reflecting a Breakpoint

In this section, an expression is derived for the time of the earliest global state reflecting a set of breakpoint events. In terms of cuts, this state can be defined as follows.

Definition 4.1. (Causal past) The function $\mathbf{p}. : E \cup 2^E \longrightarrow 2^E$ defines the causal past of events and sets of events as follows. For any event $e \in E$, $\mathbf{p}e = \{e_0 \in E \mid e_0 \preceq e\}$. For any $A \subseteq E$, $\mathbf{p}A = (\cup\, a : a \in A : \mathbf{p}a)$.

It follows from Definition 3.2 (Consistent cut) that the causal past of an event is a consistent cut. Since the union of consistent cuts is a consistent cut, the causal past of a set of events also is a consistent cut. Definitions 3.2 and 4.1 yield that the causal past of an event is the earliest consistent cut containing the event.

Corollary 4.2. [18] *For any event* e *and consistent cut* C, $e \in C \Leftrightarrow \mathbf{p}e \subseteq C$.

Definition 4.1 and Corollary 4.2 yield that the causal past of a set of events is the earliest consistent cut containing every element in the set.

Corollary 4.3. *For any set* $A \subseteq E$ *and consistent cut* C, $A \subseteq C \Leftrightarrow \mathbf{p}A \subseteq C$.

Thus, if we want to derive an expression for the time of the earliest global state reflecting a set of events, we must calculate a function $egs : 2^E \longrightarrow T_{\preceq}$ such that for any $A \subseteq E$, $egs.A = \mathcal{T}.\mathbf{p}A$.

The first step is to find an expression for the global time of the causal past of an event. For this purpose, vector timestamps as introduced by Mattern [17, 18] and Fidge [9, 10] can be used. A vector of size N is assigned to every event in E such that every component $i \in \mathcal{P}$ of the vector is equal to the number of predecessors of the event in process i.

Definition 4.4. (Causal past in a process) For any $i \in \mathcal{P}$, the function $\mathbf{p}_i. : E \longrightarrow 2^{E_i}$ defines the causal past in process i of an event as follows. For any $e \in E$, $\mathbf{p}_i e = \{e_0 \in E_i \mid e_0 \preceq e\}$.

Definition 4.5. (Timestamp function) The function $T : E \longrightarrow \mathbb{N}^N$ defines a timestamp for every event as follows. For any event $e \in E$ and process $i \in \mathcal{P}$, $T.e.i = |\mathbf{p}_i e|$.

An algorithm for calculating vector timestamps for the model of distributed computations used in this paper can be found in [2]. During a debugging session, vector timestamps can be calculated after a distributed program has been executed and event information has been recorded. This means that the timestamps are available when a computation is replayed.

Mattern and Fidge introduced vector timestamps to characterize the precedence relation. That is, they have shown that an event precedes another event if and only if its timestamp is at most the timestamp of the other event. For our purposes, the following corollary is of greater importance. It follows from Definitions 3.5 (Global time of a cut), 4.1 (Causal past), and 4.5 (Timestamp function).

Corollary 4.6. [18] *For any event $e \in E$, $\mathcal{T}.\mathbf{p}e = T.e$.*

Corollary 4.6 implies that for an event e, $egs.\{e\}$ is equal to $T.e$. This means that we have found a simple expression for the earliest global state reflecting a single event. This result is similar to the results given by Fowler and Zwaenepoel [11] for causal distributed breakpoints. The following derivation yields the desired result for arbitrary sets of events. Let A be a subset of E.

$$\begin{array}{ll} & \mathcal{T}.\mathbf{p}A \\ = & \{ \text{ Definition 4.1 (Causal past) } \} \\ & \mathcal{T}.(\cup\, a :\ a \in A :\ \mathbf{p}a) \\ = & \{ \text{ Theorem 3.8 (Structure of consistent time vectors) } \} \\ & (\text{SUP}\ a :\ a \in A :\ \mathcal{T}.\mathbf{p}a) \\ = & \{ \text{ Corollary 4.6 } \} \\ & (\text{SUP}\ a :\ a \in A :\ T.a) \end{array}$$

Theorem 4.7. (Earliest global state) *For any $A \subseteq E$, $egs.A = \mathcal{T}.\mathbf{p}A = (\text{SUP}\, a : a \in A : T.a)$.*

This theorem gives a simple expression in terms of vector timestamps for the earliest global state reflecting a set of breakpoint events. The expression can be calculated while trying to detect a breakpoint. The replay mechanism can then be used to restore the state in an additional replay.

An example of Theorem 4.7 is given in Figure 2. Let "sup" be the binary supremum operator. If A is a breakpoint, Theorem 4.7 yields $(2, 3, 3)$ as a representation of the earliest global state. This vector corresponds to the cut depicted by the solid line. It is easy to see that every component is exactly the number of predecessors of A in the corresponding process.

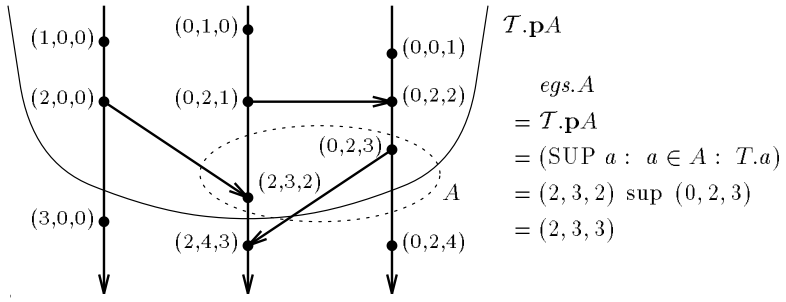

Fig. 2. The earliest global state reflecting a set of breakpoint events.

5 The Global State Preceding a Breakpoint

Another global state that might be of interest to the programmer is the state reflecting only events preceding any of the breakpoint events, but reflecting none of the breakpoint events themselves. Starting in this state, the programmer can investigate the effect of the breakpoint events on the state of the computation by executing them one at a time.

In this section, two expressions for the state preceding a breakpoint are given. The first result (Theorem 5.11) is in terms of reversed vector time. The second expression (Theorem 5.17) is based on the notion of convexity. The two results are meant as alternatives. Although the second alternative appears useful in most cases, it depends on the exact requirements of a distributed-debugging environment what alternative should be implemented.

A complete formal proof of the two main theorems can be found in the appendix. The proof of the second theorem is based on the first result. It is also possible to prove the second theorem directly. However, it is convenient to use the first one.

The global state preceding a breakpoint can be defined in terms of cuts as the state reflecting the causal past of a breakpoint minus the set of events succeeding any breakpoint event, where the successor relation $\succeq$ is defined as the dual of $\preceq$. That is, for any $e_0, e_1 \in E$, $e_0 \succeq e_1 \Leftrightarrow e_1 \preceq e_0$. Note that $\succeq$ is reflexive.

Definition 5.1. (Causal future) The function $\mathbf{f}.\ :\ E \cup 2^E \longrightarrow 2^E$ defines the causal future of an event or a set of events as follows. For any $e \in E$, $\mathbf{f}e = \{e_0 \in E \mid e_0 \succeq e\}$. For any $A \subseteq E$, $\mathbf{f}A = (\cup\ a:\ a \in A:\ \mathbf{f}a)$.

Using this definition, we want to calculate $\mathcal{T}.(\mathbf{p}A \setminus \mathbf{f}A)$, for some breakpoint $A \subseteq E$. Since only *consistent* global states of the computation can be restored, we must show that the cut $\mathbf{p}A \setminus \mathbf{f}A$ is consistent for any $A \subseteq E$. For this purpose, we can exploit the duality of the precedence and successor relations. The local-successor relation $\succeq_l$ is the dual of $\preceq_l$.

Definition 5.2. (Successor cut) A set $C \subseteq E$ is called a successor cut, or $\succeq$-cut, of E if and only if for all events $e_0 \in C$ and $e_1 \in E$, $e_1 \succeq_l e_0 \Rightarrow e_1 \in C$. The set of all $\succeq$-cuts is denoted $C_{\succeq_l}$.

Definition 5.3. (Consistent successor cut) A set $C \subseteq E$ is called a *consistent* $\succeq$-cut of E if and only if for all events $e_0 \in C$ and $e_1 \in E$, $e_1 \succeq e_0 \Rightarrow e_1 \in C$. The set of all consistent $\succeq$-cuts is denoted $C_{\succeq}$.

The next corollary states the relation between cuts and $\succeq$-cuts.

Corollary 5.4. *For $C \subseteq E$, $C \in C_{\preceq_l} \Leftrightarrow E \backslash C \in C_{\succeq_l}$ and $C \in C_{\preceq} \Leftrightarrow E \backslash C \in C_{\succeq}$.*

It follows from Definitions 5.1 and 5.3 that the causal future of an event or set of events is a consistent $\succeq$-cut. Therefore, Corollary 5.4 implies that for any $A \subseteq E$, the cut $E \setminus \mathbf{f}A$ is consistent. It follows that $\mathbf{p}A \setminus \mathbf{f}A$, which is the intersection of $\mathbf{p}A$ and $E \setminus \mathbf{f}A$, also is a consistent cut.

Now we have established that $\mathbf{p}A \setminus \mathbf{f}A$ is a consistent cut for any $A \subseteq E$, we can formalize the goal of this section as follows: Calculate the function $pgs : 2^E \longrightarrow T_{\preceq}$ (*p*receding *g*lobal *s*tate) such that for any $A \subseteq E$, $pgs.A = \mathcal{T}.(\mathbf{p}A \backslash \mathbf{f}A)$.

The idea behind calculating this function is to introduce a second timestamp for events representing the causal future like the timestamp T represents the causal past (Corollary 4.6 and Theorem 4.7).

Definition 5.5. (Reversed vector time of a successor cut) The function $\mathcal{T}^R : C_{\succeq_l} \rightarrow \mathbb{N}^N$ defines the *reversed vector time* of a $\succeq$-cut. For any $\succeq$-cut C, component i of the reversed time vector is defined as $\mathcal{T}^R.C.i = |C \cap E_i|$.

Definition 5.6. (Causal future in a process) For any $i \in \mathcal{P}$, the function $\mathbf{f}_i. : E \longrightarrow 2^E$ defines the causal future in process i of an event as follows. For any $e \in E$, $\mathbf{f}_i e = \{e_0 \in E_i \mid e_0 \succeq e\}$.

Definition 5.7. (Timestamp function T^R) The function $T^R : E \longrightarrow \mathbb{N}^N$ defines a timestamp in reversed vector time for every event as follows. For any event $e \in E$ and process $i \in \mathcal{P}$, $T^R.e.i = |\mathbf{f}_i e|$.

The timestamps defined by T^R can be obtained by applying a timestamp algorithm for ordinary vector timestamps while traversing event information backwards. The duality of the relations $\preceq$ and $\succeq$ yields the desired results.

Corollary 5.8. *For any event $e \in E$, $\mathcal{T}^R.\mathbf{f}e = T^R.e$.*

Corollary 5.9. *For any $A \subseteq E$, $\mathcal{T}^R.\mathbf{f}A = (\mathrm{SUP}\ a :\ a \in A :\ T^R.a)$.*

This corollary gives an expression in terms of reversed vector timestamps for the reversed time of the causal future of a set of events. It is the key to deriving an expression for the global time of the state preceding a breakpoint. The following corollary states the relation between vector time and reversed vector time. It follows from Definitions 3.5 (Global time of a cut) and 5.5 (Reversed time of a successor cut), and Corollary 5.4. The binary operator "$-$" on vectors is defined as componentwise subtraction.

Corollary 5.10. *For any successor cut $C \in C_{\succeq_l}$, $\mathcal{T}.(E \backslash C) = \mathcal{T}.E - \mathcal{T}^R.C$. For any cut $C \in C_{\preceq_l}$, $\mathcal{T}^R.(E \backslash C) = \mathcal{T}^R.E - \mathcal{T}.C$.*

The vectors $\mathcal{T}.E$ and $\mathcal{T}^R.E$ are constant and both equal to the vector whose i^{th} component is equal to the number of events in process i. In the following, this vector is denoted $\mathbf{E}$. The following theorem gives the desired expression for the state preceding a breakpoint. The proof can be found in the appendix. Let "inf" be a binary operator on vectors yielding the componentwise minimum.

Theorem 5.11. (Preceding global state) *For any* $A \subseteq E$,

$$pgs.A = \mathcal{T}.\mathbf{p}A \text{ inf } (\mathbf{E} - \mathcal{T}^R.\mathbf{f}A) = egs.A \text{ inf } (\mathbf{E} - (\text{SUP } a : a \in A : T^R.a)).$$

Theorem 5.11 gives an expression for *pgs* in terms of ordinary timestamps and reversed timestamps. In a replay environment, the two sets of timestamps can be calculated after collecting event information and before starting the detection of a breakpoint. Therefore, the above expression can be calculated during the detection of a breakpoint. Afterwards, the state can be restored in one more replay.

Theorem 5.11 is visualized in Figure 3. The computation is the same as in Figure 2. The solid line depicts the cut corresponding to the global state preceding breakpoint A. It is left as a simple exercise to calculate the reversed timestamps and verify the result.

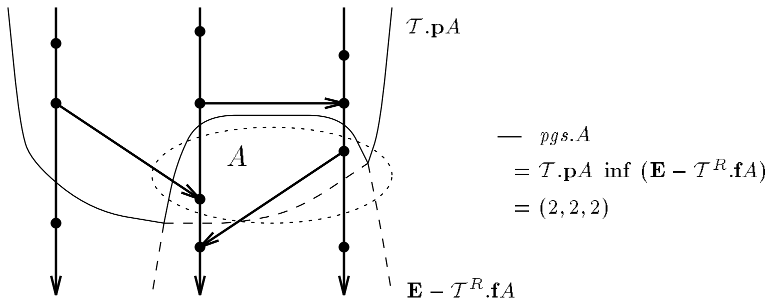

Fig. 3. The global state preceding a set of breakpoint events.

A disadvantage of the expression in Theorem 5.11 is that using two sets of timestamps is computationally expensive and requires extra storage. The following shows that at the cost of some extra effort during the detection of a breakpoint, one can overcome the need for calculating reversed timestamps. First, the notions of *location set*, *convexity*, and *convex closure* are introduced.

Definition 5.12. (Location set) The location set of a set of events is defined by a function $\boldsymbol{l}. : 2^E \longrightarrow 2^{\mathcal{P}}$ as follows. For any $A \subseteq E$, $\boldsymbol{l}A = \{i \in \mathcal{P} \mid A \cap E_i \neq \emptyset\}$.

Definition 5.13. (Convexity) A set of events A is called *convex* if and only if

$$(\forall\, a_0, a_1, e : a_0, a_1 \in A \ \wedge\ e \in E : a_0 \preceq e \ \wedge\ e \preceq a_1 \ \Rightarrow\ e \in A).$$

Definition 5.14. (Convex closure) The convex closure of a set of events A is defined by a function $\mathbf{c}. : 2^E \longrightarrow 2^E$ as the smallest convex set of events containing A.

Since for any $A \subseteq E$, $\mathbf{p}(\mathbf{c}A) = \mathbf{p}A$ and $\mathbf{f}(\mathbf{c}A) = \mathbf{f}A$, it follows that $pgs.A = pgs.\mathbf{c}A$. That is, for calculating pgs, we can use the convex closure of a breakpoint instead of the breakpoint itself. For convex event sets, the following two properties can be used to simplify the expression "$\mathcal{T}.\mathbf{p}A$ inf $(\mathbf{E} - \mathcal{T}^R.\mathbf{f}A)$" in Theorem 5.11. The result does no longer use reversed vector time. The proofs can be found in the appendix. Theorem 5.17 follows immediately.

Property 5.15. *For a convex set of events C and a process $i \in \mathcal{P}$,*
$i \in \boldsymbol{l}C \;\Rightarrow\; \mathcal{T}.\mathbf{p}C.i \geq \mathbf{E}.i - \mathcal{T}^R.\mathbf{f}C.i$, *and*
$i \notin \boldsymbol{l}C \;\Rightarrow\; \mathcal{T}.\mathbf{p}C.i \leq \mathbf{E}.i - \mathcal{T}^R.\mathbf{f}C.i$.

Property 5.16. *For a convex set of events C and a process $i \in \boldsymbol{l}C$,*
$\mathbf{E}.i - \mathcal{T}^R.\mathbf{f}C.i = (\mathrm{MIN}\ c :\ c \in C \cap E_i \ :\ T.c.i) - 1$.

Theorem 5.17. (Preceding global state) *For any $A \subseteq E$,*

$$pgs.A.i = \begin{cases} (\mathrm{MIN}\ a :\ a \in \mathbf{c}A \cap E_i \ :\ T.a.i) - 1, & \textit{for } i \in \boldsymbol{l}(\mathbf{c}A) \\ egs.A.i, & \textit{for } i \notin \boldsymbol{l}(\mathbf{c}A) \end{cases}$$

Note that $egs.A = egs.\mathbf{c}A = (\mathrm{SUP}\ a :\ a \in A \ :\ T.a)$. Since the expression for pgs does not depend on reversed timestamps, only ordinary timestamps must be calculated and stored with the event information. During the detection of a breakpoint, its convex closure is calculated and used to compute the preceding global state. For the example in Figures 2 and 3, it is easily verified that Theorem 5.17 yields the correct result. Note that the set A is convex, i.e., $\mathbf{c}A = A$.

6 Concluding Remarks

In this paper, we investigated how to restore a meaningful global state of a distributed computation after detecting a breakpoint. A breakpoint is represented by a set of events. Vector time is used as a concise representation of states.

One meaningful state is the earliest global state reflecting a set of breakpoint events. An expression in terms of vector timestamps has been derived for this state. This expression can be calculated during the detection of a breakpoint. The corresponding state of the computation can be restored in the next replay.

Another interesting state is the state reflecting only predecessors of any breakpoint event, but reflecting none of the breakpoint events themselves. Two alternative expressions have been derived for this state. The question of which expression should be used can only be answered within the context of a particular debugging environment. If storage is not a problem, and if response time is important, the alternative using reversed vector time is probably preferred. Reversed timestamps require extra storage, but can be calculated before starting the detection of a breakpoint. During the detection, calculating the expression for the preceding global state is straightforward. However, if storage is a problem, or if response time is less important, one probably wants to use the alternative expression using the convex closure of a set of breakpoint events. For this expression, reversed timestamps are not necessary, reducing the amount of storage needed. Calculating the convex closure, however, slows down the detection of a breakpoint.

The results derived in this paper can also be used for other purposes. One important application is event abstraction. In [2], a theoretical treatment of the relationship between vector time and causality among abstract events is given. The expressions derived for the earliest and preceding global state of a breakpoint can also be used to determine causality among abstract events. In [13], an event-abstraction tool is described that uses a timestamp for convex abstract events that is essentially the same as our second expression for the global state preceding a breakpoint. A prototype of this tool has been implemented and the results appear to be very promising.

Acknowledgements

Part of this work was done while the author was staying at the University of Waterloo. The author would like to thank Jay Black, Mike Coffin, Thomas Kunz, and David Taylor for their critical comments and the helpful discussions.

References

1. A.A. Basten. Event Abstraction in Modeling Distributed Computations. In K. Ecker, editor, *Proceedings of the Workshop on Parallel Processing*, Lessach, Austria, September 1993. Informatik-Bericht 94/1, Technische Universität Clausthal, Germany, January 1994.
2. A.A. Basten, T. Kunz, J.P. Black, M.H. Coffin, and D.J. Taylor. Time and the Order of Abstract Events in Distributed Computations. Computing Science Note 94/06, Eindhoven University of Technology, Department of Mathematics and Computing Science, Eindhoven, The Netherlands, February 1994. Submitted 01-02-1994 to *Distributed Computing*, 29pp in ms.
3. P.C. Bates. Debugging Heterogeneous Distributed Systems using Event-Based Models of Behavior. *ACM SIGPLAN Notices*, 24(1):11–22, January 1989. Proceedings of the ACM SIGPLAN/SIGOPS Workshop on Parallel and Distributed Debugging, Madison, Wisconsin, May, 1988.
4. K.M. Chandy and L. Lamport. Distributed Snapshots: Determining Global States of Distributed Systems. *ACM Transactions on Computer Systems*, 3(1):63–75, February 1985.
5. B. Charron-Bost. Combinatorics and Geometry of Consistent Cuts: Application to Concurrency Theory. In J.-C. Bermond and M. Raynal, editors, *Distributed Algorithms*, volume 392 of *Lecture Notes in Computer Science*, pages 45–56. Springer Verlag, Berlin, Germany, 1989. Proceedings of WDAG '89, Nice, France, September 1989.
6. B. Charron-Bost, F. Mattern, and G. Tel. Synchronous and Asynchronous Communication in Distributed Computations. Technical Report LITP 91.55, Institut Blaise Pascal, Université Paris 7, Paris, France, September 1991.
7. R. Cooper and K. Marzullo. Consistent Detection of Global Predicates. In *Proceedings of the ACM/ONR Workshop on Parallel and Distributed Debugging*, pages

163–173, Santa Cruz, California, May 1991. The proceedings appeared also as *ACM SIGPLAN Notices*, 26(12), December 1991.
8. I.J.P. Elshoff. A Distributed Debugger for Amoeba. In *Proceedings of the ACM SIGPLAN/SIGOPS Workshop on Parallel and Distributed Debugging*, pages 1–10, Madison, Wisconsin, May 1988.
9. C.J. Fidge. *Dynamic Analysis of Event Orderings in Message-Passing Systems.* PhD thesis, Australian National University, Department of Computer Science, Canberra, Australia, 1989.
10. C.J. Fidge. Logical Time in Distributed Computing Systems. *IEEE Computer*, 24(8):28–33, August 1991.
11. J. Fowler and W. Zwaenepoel. Causal Distributed Breakpoints. In *Proceedings of the 10th International Conference on Distributed Computing Systems*, pages 134–141, Paris, France, May/June 1990.
12. D. Haban and W. Weigel. Global Events and Global Breakpoints in Distributed Systems. In *Proceedings of the 21st Annual Hawaii International Conference on System Sciences*, volume II, pages 166–175, Kailua-Kona, Hawaii, January 1988.
13. Thomas Kunz. *Abstract Behaviour of Distributed Executions with Applications to Visualization.* PhD thesis, Department of Computer Science, Technical University of Darmstadt, Darmstadt, Germany, May 1994.
14. L. Lamport. Time, Clocks and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 1978.
15. Y. Manabe and M. Imase. Global Conditions in Debugging Distributed Programs. *Journal of Parallel and Distributed Computing*, 15(1):62–69, May 1992.
16. K. Marzullo and G. Neiger. Detection of Global State Predicates. In S. Toueg, P.G. Spirakis, and L.Kirousis, editors, *Distributed Algorithms*, volume 579 of *Lecture Notes in Computer Science*, pages 254–272, Berlin, Germany, 1991. Springer Verlag. Proceedings of WDAG '91, Delphi, Greece, October 1991.
17. F. Mattern. Virtual Time and Global States of Distributed Systems. In M. Cosnard et al., editor, *Parallel and Distributed Algorithms*, pages 215–226. Elsevier Science Publishers B.V., Amsterdam, North-Holland, The Netherlands, 1989. Proceedings of the International Workshop held in Gers, France, October, 1988.
18. F. Mattern. On the Relativistic Structure of Logical Time in Distributed Systems. *Bigre*, 78:3–20, March 1992. Proceedings of the workshop: Datation et Contrôle des Exécutions Réparties, December 4th, 1991, Rennes, France.
19. B.P. Miller and J.-D. Choi. Breakpoints and Halting in Distributed Programs. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 316–323, San Jose, California, June 1988.
20. E.T. Smith. Debugging Tools for Message-Based Communication Processes. In *Proceedings of the 4th International Conference on Distributed Computing Systems*, pages 303–310, San Fransisco, California, 1984.
21. D.J. Taylor. A Prototype Debugger for Hermes. In *Proceedings of the 1992 CAS Conference, Volume I*, pages 29–42, Toronto, Ontario, Canada, November 1992. IBM Canada Ltd. Laboratory, Centre for Advanced Studies.

A Appendix: Proofs

Theorem 5.11. *For any* $A \subseteq E$,
$$pgs.A = \mathcal{T}.\mathbf{p}A \text{ inf } (\mathbf{E} - \mathcal{T}^R.\mathbf{f}A) = egs.A \text{ inf } (\mathbf{E} - (\text{SUP } a : a \in A : \mathcal{T}^R.a)).$$

Proof.

$$
\begin{array}{ll}
 & pgs.A \\
= & \{ \text{ Definition } pgs \ \} \\
 & \mathcal{T}.(\mathbf{p}A \setminus \mathbf{f}A) \\
= & \{ \text{ Set calculus } \} \\
 & \mathcal{T}.(\mathbf{p}A \cap (E \setminus \mathbf{f}A)) \\
= & \{ \text{ Theorem 3.8 (Structure of consistent time vectors) } \} \\
 & \mathcal{T}.\mathbf{p}A \ \text{inf} \ \mathcal{T}.(E \setminus \mathbf{f}A) \\
= & \{ \text{ Corollary 5.10 } \} \\
 & \mathcal{T}.\mathbf{p}A \ \text{inf} \ (\mathbf{E} - \mathcal{T}^R.\mathbf{f}A) \\
= & \{ \text{ Definition } egs\text{; Corollary 5.9 } \} \\
 & egs.A \ \text{inf} \ (\mathbf{E} - (\text{SUP } a : \ a \in A : \ T^R.a))
\end{array}
$$

□

For the proofs of Properties 5.15 and 5.16, we need two new definitions and some intermediate results. Duals of results are omitted.

Definition A.1. (Causal past of a set of events in a process) For any $i \in \mathcal{P}$, the function $\mathbf{p}_i. : 2^E \longrightarrow 2^{E_i}$ defines the causal past in process i of a set of events as follows. For any $A \subseteq E$, $\mathbf{p}_i A = (\cup\, a : \ a \in A : \ \mathbf{p}_i a)$. Note that $\mathbf{p}_i A = \mathbf{p}A \cap E_i$.

Definition A.2. (Causal future of a set of events in a process) For any $i \in \mathcal{P}$, the function $\mathbf{f}_i. : 2^E \longrightarrow 2^{E_i}$ defines the causal future in process i of a set of events as follows. For any $A \subseteq E$, $\mathbf{f}_i A = (\cup\, a : \ a \in A : \ \mathbf{p}_i a)$.

The following corollary is a direct result from Definitions 3.5 (Global time of a cut), 4.1 (Causal past), and 4.4 (Causal past in a process).

Corollary A.3. *For any process $i \in \mathcal{P}$ and event $e \in E$,*

$$\mathcal{T}.\mathbf{p}_i e.j = \begin{cases} \mathcal{T}.\mathbf{p}e.j, & \text{for } j = i \\ 0, & \text{otherwise} \end{cases}$$

For sets of events, we have a similar result.

Corollary A.4. *For any process $i \in \mathcal{P}$ and $A \subseteq E$,*

$$\mathcal{T}.\mathbf{p}_i A.j = \begin{cases} \mathcal{T}.\mathbf{p}A.j, & \text{for } j = i \\ 0, & \text{otherwise} \end{cases}$$

The following property for convex sets of events is easily verified. The proof is left to the reader.

Property A.5. *For a convex set of events C and a process $i \in \mathcal{P}$,*

$i \in \boldsymbol{l}C \ \Rightarrow \ \mathbf{p}_i C \cap \mathbf{f}_i C \neq \emptyset$, *and*

$i \notin \boldsymbol{l}C \ \Rightarrow \ \mathbf{p}_i C \cap \mathbf{f}_i C = \emptyset$.

Property 5.15. *For a convex set of events C and a process $i \in \mathcal{P}$,*

$i \in \boldsymbol{l}C \ \Rightarrow \ \mathcal{T}.\mathbf{p}C.i \geq \mathbf{E}.i - \mathcal{T}^R.\mathbf{f}C.i$, *and*

$i \notin \boldsymbol{l}C \ \Rightarrow \ \mathcal{T}.\mathbf{p}C.i \leq \mathbf{E}.i - \mathcal{T}^R.\mathbf{f}C.i$.

Proof. For $i \in \boldsymbol{l}C$,

$$\begin{array}{ll} & \mathcal{T}.\mathbf{p}C.i \\ = & \{ \text{ Corollary A.4 } \} \\ & \mathcal{T}.\mathbf{p}_i C.i \\ \geq & \{ \text{ Property A.5: } i \in \boldsymbol{l}C \;\Rightarrow\; \mathbf{p}_i C \supseteq E \setminus \mathbf{f}_i C \} \\ & \mathcal{T}.(E \setminus \mathbf{f}_i C).i \\ = & \{ \text{ Corollary 5.10 } \} \\ & \mathbf{E}.i - \mathcal{T}^R.\mathbf{f}_i C.i \\ = & \{ \text{ The dual of Corollary A.4 } \} \\ & \mathbf{E}.i - \mathcal{T}^R.\mathbf{f}C.i \end{array}$$

For $i \notin \boldsymbol{l}C$, the derivation is almost identical and, therefore, omitted. □

For proving Property 5.16, we need the following property of convex event sets.

Property A.6. *For a convex set C and a process $i \in \boldsymbol{l}C$, $\mathbf{p}_i C = \mathbf{p}_i(C \cap E_i)$.*

Proof. First, we show that $\mathbf{p}_i(C \cap E_i) \subseteq \mathbf{p}_i C$. It follows immediately from $C \cap E_i \subseteq C$. Second, we show that $\mathbf{p}_i C \subseteq \mathbf{p}_i(C \cap E_i)$. Assume $c \in \mathbf{p}_i C \setminus \mathbf{p}_i(C \cap E_i)$. Note that $c \in \mathbf{p}_i C$ implies $c \in E_i$. So $c \notin \mathbf{p}_i(C \cap E_i)$ implies $c \notin C$. It follows from $i \in \boldsymbol{l}C$, that $C \cap E_i \neq \emptyset$. Given that $c \notin \mathbf{p}_i(C \cap E_i)$, it follows that there is a $c_0 \in C \cap E_i$, such that $c_0 \prec c$. Since $c \in \mathbf{p}_i C$, there also exists an event $c_1 \in C$ such that $c \preceq c_1$. This contradicts the convexity of C. Therefore, $\mathbf{p}_i C \subseteq \mathbf{p}_i(C \cap E_i)$. □

In addition, we need the following special case of Corollary 5.10.

Corollary A.7. *For any $i \in \mathcal{P}$ and $e \in E_i$, $\mathcal{T}^R.\mathbf{f}_i e.i - 1 = \mathbf{E}.i - \mathcal{T}.\mathbf{p}_i e.i$*

Property 5.16. *For a convex set of events C and a process $i \in \boldsymbol{l}C$,*
$\mathbf{E}.i - \mathcal{T}^R.\mathbf{f}C.i = (\text{MIN } c : \; c \in C \cap E_i \; : \; T.c.i) - 1$.

Proof.

$$\begin{array}{ll} & \mathcal{T}^R.\mathbf{f}C.i \\ = & \{ \text{ The dual of Corollary A.4 } \} \\ & \mathcal{T}^R.\mathbf{f}_i C.i \\ = & \{ \text{ The dual of Property A.6 } \} \\ & \mathcal{T}^R.\mathbf{f}_i(C \cap E_i).i \\ = & \{ \text{ Definition A.2 (Causal future) } \} \\ & \mathcal{T}^R.(\cup\, c : \; c \in C \cap E_i \; : \; \mathbf{f}_i c).i \\ = & \{ \text{ The dual of Theorem 3.8 } \} \\ & (\text{SUP } c : \; c \in C \cap E_i \; : \; \mathcal{T}^R.\mathbf{f}_i c).i \\ = & \{ \text{ SUP is the componentwise maximum } \} \\ & (\text{MAX } c : \; c \in C \cap E_i \; : \; \mathcal{T}^R.\mathbf{f}_i c.i) \\ = & \{ \text{ Corollary A.7 } \} \\ & (\text{MAX } c : \; c \in C \cap E_i \; : \; \mathbf{E}.i - \mathcal{T}.\mathbf{p}_i c.i + 1) \\ = & \{ \text{ Corollaries A.3 and 4.6; Domain is not empty } \} \\ & \mathbf{E}.i - (\text{MIN } c : \; c \in C \cap E_i \; : \; T.c.i) + 1 \end{array}$$

□