A Basic Unit of Computation in Distributed Systems

Mohan Ahuja and Ajay D. Kshemkalyani
Dept. of Computer and Information Science

Timothy Carlson
Dept. of Mathematics

The Ohio State University, Columbus, OH 43210

ABSTRACT

A ground state in a distributed system is a global state in which there are no messages in transit. In this paper, we define basic units of computation in distributed systems, whether communicating synchronously or asynchronously, as comprising of indivisible logical units of computation that take the system from one ground state to another. We analyse their properties. This definition helps in understanding the nature of the communication pattern in a computation and how the communication pattern is related to the concurrency in the system. The basic unit of computation is potentially useful in checkpointing, distributed debugging and asserting about attainment of stable properties.

1 Introduction

It is widely accepted that an event on a node is the unit of computation on the node. Even though there is no singly accepted definition of an event, it is commonly agreed upon that an event is an action local to a node in the distributed system (DS). Though the notion of an event is very useful, as a unit of computation it does not capture the dynamics of a distributed computation. We identify a distributed basic unit of computation and explore its properties.

The evolution of our basic unit has been guided by the pragmatics of a "basic unit" in any system. We view a "basic unit" as an entity that is "indivisible" and has minimal dependence on external agents, in the context in which it is defined in a system. It should also preserve some useful invariant properties in the system. Our system is a DS and the context is one in which we can get a better understanding of the interactions in a DS.

For any computation in a DS, the initial global state [3] and the final global state (if the computation terminates) do not have any messages in transit. A global state in which there are no messages in transit is called a ground state. This has also been referred to as a synchronized global state [11]. The importance of the ground state has been recognized long ago in [16] which created a strong type of ground state through the use of "conversations" for domino-free rollback recovery. [5] created ground states for checkpointing in a distributed transaction system. A ground state is a meaningful observation point for the DS because the future computation depends only on the state of each node in that ground state. A computation takes a DS from an initial global state through a succession of global states [3]. We propose that during the computation, the DS may naturally pass through some possible global states which are ground states. This observation is true of any DS, irrespective of any parameters such as the communication synchrony and any program run on the DS. The sub-parts of a computation that take the DS from one possible ground state to another possible ground state are each units of the computation, which we call molecules. A molecule does not interact with the rest of the computation but could be divisible into more than one molecules. A *basic unit* of computation which we call an atom, is a molecule such that no smaller part of it is itself a molecule, i.e., an atom is indivisible. The basic units are distributed across the nodes of the system and are formed at run-time. A computation can be seen as a partial order over all the atoms, the state of the DS after any prefix of this partial order being a ground state.

The **contribution of this work** is the understanding it provides of the communication patterns in a computation and how the communication patterns need to be controlled for achieving power of reasoning in a DS. The structure of the basic units of computation reflects the communication patterns. The allowed concurrency depends on factors like the nature of communication channels, topology of the DS, communication synchrony and the program itself and determines the richness of possible scenarios in a computation. When concurrency is controlled, some possible scenarios in the computation are prevented, leading to simplified reasoning about the DS. This is reflected in the simplified structure of the basic units of computation.

In Section 2, we define our distributed system model. Section 3 defines the basic unit of computation and gives its properties. Section 4 explains how a computation can be viewed as a partial order over the basic units of the computation. Section 5 considers the problem of detecting the basic units. Detecting them as they naturally occur in

CH2878-7/90/0000/0012$01.00 © 1990 IEEE

a computation is (comparatively) difficult, detecting them even in retrospect is not easy. But because the ground states at the boundaries of these units have useful properties, it is useful to create them artificially. Even this is seen to have high overheads. One algorithm to create ground states during a computation in an asynchronously communicating system with FIFO channels is given, and an existing algorithm [2] that implicitly creates ground states in a synchronously communicating system is referenced. Section 6 explains the significance of the basic unit and gives its applications. Section 7 gives the conclusions.

2 A Distributed System Model

The system consists of a set of nodes N, communicating through a set of reliable unidirectional channels that interconnect them. A node involved in a computation does internal processing and communicates with other nodes. States of each node are modelled such that a state transition occurs only on the sending(receipt) of a message to(from) another node. An event at a node is either a send or a receive of a message at the node and is an atomic action at that node. An event is defined by the node number on which it occurs, the logical sequence number of the event on that node, the source and the destination of the message corresponding to the event and the message body. An event, denoted as e_{ij}, is the j^{th} event on node i. We define $|N|$ dummy events $e_{i,0}\, \forall\, i \in N$ that represent the initial events, one on each node, and are useful in the characterization of the basic unit of computation.

The model given above pertains to a distributed system as a whole. States could be modelled differently from the above, internally at a node, i.e., each node may have internal events which could figure in the state transitions internal to that node. However, any such modelling is at a level lower than that of the proposed model and hence is transparent. This approach is easily justified. In a DS, the transitions at a node between any consecutive pair of communications are not relevant to the system as a single unit since they are not visible to the other nodes. By modelling in this manner, no knowledge is lost as explained by the Knowledge Transfer Theorem [4]. Since events in the proposed model do not include internal events at all, the number of events and states in the system is reduced. This reduces the possible-world space [6] for a given computation without losing reasoning power.

The model is general and includes all possible DSs. It does not make the distinction between synchronous and asynchronous communication, and does not consider whether channels are FIFO in the latter case. Thus, the results about the basic unit of computation are valid in all distributed computations.

The definitions of the causality (happens before) relation among events in a computation [10], the global snapshot/global state [1, 3], and the consistent cut [15] will be used extensively and are reproduced here.

Definition 1 *Define "happens before" or the dependence relation $e_{i,s} \rightarrow e_{j,t}$ between two events $e_{i,s}$ and $e_{j,t}$ if (1) $i = j$ and $t > s$ or; (2) $i \neq j$ and $e_{i,s}$ is a send and $e_{j,t}$ is the corresponding receive, or; (3) $\exists e_{k,u} | e_{i,s} \rightarrow e_{k,u} \wedge e_{k,u} \rightarrow e_{j,t}$.*

Definition 2 *A global state [3] is a set of states (1) one for each node, such that every message that a node receives before it reaches its state would have been sent by the sender of that message before it reached its state; (2) one for each channel, as a set [1] (sequence for a FIFO channel [3]) of messages sent along that channel before the sender node reached its state less the messages received by the receiver node before it reached its state.*
A global snapshot is a recorded global state.
The set of instants of recording the node states is termed a consistent cut [15].

3 Units Of Computation

Here we give a definition of a basic unit and explore its properties. The definitions and theorems are illustrated by timing diagrams (which use horizontal lines representing time lines of nodes) for a six node DS communicating asynchronously over non-FIFO channels. Systems with asynchronous communication over FIFO channels and with synchronous communication are special cases of the above and are considered in Sections 5 and 6.

Definition 3 *A ground state of a DS is a global state in which all channels are empty.*

Let C be the possibly infinite set of all the events in a computation, plus the $|N|$ dummy events $e_{i,0}$, one on each node $i \in N$.

Definition 4 *$M(e_{i,j})$ is a one-to-one function from C to C that maps $M(e_{i,0}) = e_{i,0}$ and for $j > 0$, $M(e_{i,j}) = e_{k,l}$ where $e_{k,l}$ is a send(receive) corresponding to $e_{i,j}$.*

Definition 5 *A set S is a dipole set if $S = \{e_{i,j} \mid e_{i,j} \in S \implies M(e_{i,j}) \in S\}$.*

Dipole sets are closed under : set union, intersection, difference and complementation.

Definition 6 *A wave W is a set of events $e_{i,j}$ in C such that*
(1) $\forall i \in N\, \exists e_{i,j} \in W$
(2) W is a dipole set.
(3) $e_{i,j} \in W \Rightarrow ((j = 0) \bigvee (e_{i,j-1} \in W))$
$W_0 = \{e_{i,0}\, \forall i \in N\}$ is the dummy initial wave.

Waves are closed under union and intersection. A wave is a pre-fix of a computation such that every send and receive in the wave has the corresponding receive and send in the wave. The term "wave" captures the manner in which the computation progresses. The system is in a ground state after the execution of a wave. W_0, W_1 and W_2 in Figure 1

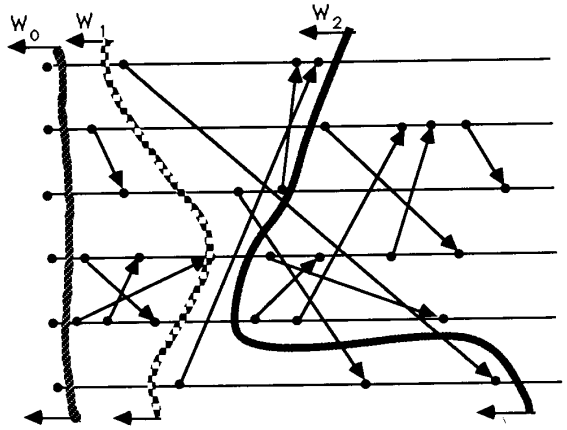

Figure 1: Some examples of waves and wavefronts in an asynchronous computation. Horizontal lines represent time lines of nodes.

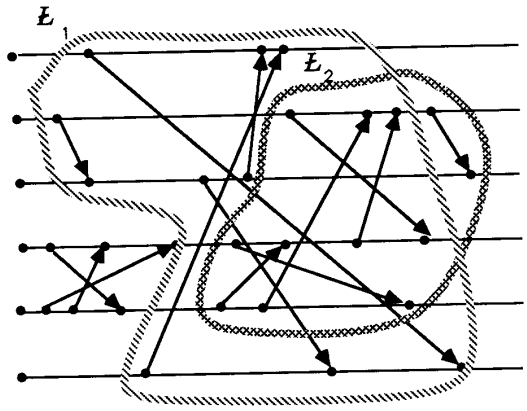

Figure 2: Some molecules in the computation of Figure 1. Horizontal lines represent time lines of nodes.

are examples of waves in the computation shown. Clearly, C is a wave, and if C is finite, all waves in C are finite. An infinite wave represents a non-terminating computation.

Definition 7 *A wavefront of a wave W, denoted by $F(W)$, is defined by following condition : $e_{i,j} \in F(W)$ iff $((e_{i,j} \in W) \wedge (\forall k \mid k > j, e_{i,k} \notin W))$.*
$F(W_0) = \{e_{i,0} \forall i \in N\}$ is the initial wavefront.

A wavefront is the set of last events of the wave on each node. For a finite wave, the wavefront has $|N|$ events and the global state after the execution of each event on the wavefront represents a ground state. The wavefronts of the waves of Figure 1 are easily observed in Figure 1. It can be shown that the waves in a computation form a lattice. The proof of the following theorem is left as an exercise. [8] shows a similar result for the set of all the system states in a computation.

Theorem 1 *Waves of a computation form a lattice.*

Definition 8 *The arity of a set of events S is the number of distinct $i \mid e_{i,j} \in S$.*

The arity of any wave is $|N|$. The arity of any wavefront is the number of nodes on which the events of the wave are finite in number and is in the range [0,|N|].

Definition 9 *A molecule L is a non-empty set of events such that*
(1) $e_{i,j} \in L \Rightarrow j > 0$
(2) L is a dipole set.
(3) $\exists W \mid \forall e_{i,j} \in L, ((e_{i,j-1} \in L) \vee (e_{i,j-1} \in F(W)))$
L is said to begin on wavefront $F(W)$.

These requirements mean that a molecule is any set of events such that every send/receive in it has its corresponding receive/send in the molecule, and if two events on a node belong to the same molecule, all intervening events on that node also belong to it. In addition, the molecule must begin on a wavefront. Thus, the only way the rest of the computation affects the molecule is in the determination of the node states at the start of the molecule execution. The only way the molecule affects the rest of the computation is in the determination of the node states at the end of the molecule execution. From Definitions 6 and 9, it follows that every wave without its dummy events is a molecule. Note that a molecule is never a wave. Figure 2 shows some molecules in the computation of Figure 1.

Lemma 1 *For any wave W, $\overline{W} = C - W$ is a molecule (if $W \subset C$).*

Proof : We prove that $C - W$ satisfies the three properties of a molecule.

$C - W$ never contains any $e_{i,j} \mid j = 0$. This satisfies (1). W and C are dipole sets. So $C - W$ is also a dipole set. This satisfies property (2). Consider any $e_{i,j} \in \overline{W}$. $e_{i,j-1} \in \overline{W} \vee e_{i,j-1} \in W$. $e_{i,j-1} \in W \Longrightarrow e_{i,j-1} \in F(W)$. In either case, property (3) is satisfied. □

From the definition of a molecule, it follows that it is independent of the rest of the computation in the following sense.

1. No messages sent within the molecule are received outside it and no messages sent outside it are received within it. (property (2))

2. Once a node stops executing events in a molecule, it may not resume executing events in the same molecule at any later time. (property (3))

Definition 10 *An atom l is a molecule(Definition 9) such that $\nexists S \mid S \subset l$ and S is a molecule.*

Atoms and molecules can be infinite sets. From Definitions 9 and 10, it is evident that every atom is a molecule but not vice-versa. An atom is any molecule that does not have a proper subset that is a molecule, and thus possesses indivisibility. The atom has minimal arity and minimal size among molecules containing any subset of the events contained in the atom.

We now prove several other properties of the atom and then establish that each event in a computation belongs to a unique atom.

Definition 11 *The pre-cut of an atom l, $R(l)$ is a set defined by the condition : $e_{i,j} \in R(l)$ iff $(e_{i,j} \notin l \wedge e_{i,j+1} \in l)$.*

The pre-cut of an atom is a subset of some wavefront and has arity-of-l events.

Lemma 2 *If L is a molecule and W is a wave such that L begins on $F(W)$, then $W \cup L$ is a wave.*

Proof : We show that $W' = W \cup \mathrm{L}$ satisfies the definition of a wave.
(1) is satisfied as W is a wave and we are doing a set union.
(2) is satisfied as W and L are dipole sets and their union is also a dipole set.
(3) is satisfied by the following reasoning. Since L begins on $F(W)$, by Definitions 6 and 9, $W \cap \mathrm{L} = \phi$. For any $e_{i,j} \in W'$, $e_{i,j} \in W \vee e_{i,j} \in \mathrm{L}$. In the first case, property (3) holds because W satisfies the property. In the second case, $e_{i,j-1} \in \mathrm{L} \vee e_{i,j-1} \in W$. Whichever be true, $e_{i,j-1} \in W'$ and property (3) is satisfied. □

Lemma 2 indicates that when the execution of a molecule is suffixed to the execution of the wave on which it begins, the resulting execution is an execution of a wave. Thus, the execution of a molecule transforms the global state of the DS from one ground state to another. Lemma 2 in conjunction with properties (2) and (3) of molecules shows that the molecule can qualify as a unit of computation.

Lemma 3 *Given an atom l and a wave W, $(l \subset W) \vee (l \cap W = \phi)$, i.e., an atom is either contained fully within a wave or the intersection of the two is the null set.*

Proof : Given l and W, let $S = l \cap W \neq \phi$ We show that S satisfies the properties of a molecule.

$\forall i \in N$, $e_{i,0} \notin l$. Hence, $\forall i \in N$, $e_{i,0} \notin S$. So S satisfies property (1).

l and W are dipole sets. So S is also a dipole set. So S satisfies property (2).

Now consider any $e_{i,j} \mid e_{i,j} \in S$. $(e_{i,j} \in W \wedge e_{i,j} \in l) \implies e_{i,j-1} \in W \wedge (e_{i,j-1} \in l \vee e_{i,j-1} \in R(l))$, where $R(l) \subseteq F$ for some pre-determined wavefront. So S satisfies property (3).

But by the definition of an atom, no proper subset of l can be a molecule. So $S = l$ and hence, $l \subseteq W$. □

Lemma 4 *For any two atoms l_1 and l_2, $l_1 \cap l_2 = \phi$, i.e., no two atoms overlap.*

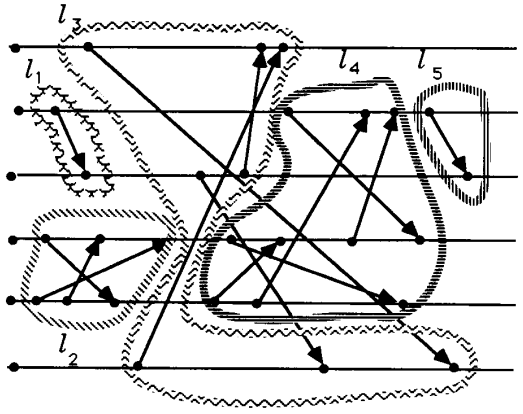

Figure 3: Five atoms in the computation of Figure 1. Horizontal lines represent time lines of nodes.

Proof : Let $R(l_1) \subseteq F(W_1)$. By Lemma 3, $(l_2 \subset W_1) \vee (l_2 \cap W_1 = \phi)$.

If $l_2 \subseteq W_1$, then $l_1 \cap l_2 = \phi$.

If $l_2 \cap W_1 = \phi$, then let $W_1 \cup l_1 = W_2$. By Lemma 3, $(l_2 \subset W_2) \vee (l_2 \cap W_2 = \phi)$. If $l_2 \cap W_2 = \phi$, then $l_1 \cap l_2 = \phi$ because $l_1 \subset W_2$. If $l_2 \subseteq W_2$, then $l_2 \subseteq (W_2 - W_1 = l_1)$. This is not possible by definition. □

A node can be executing within at most one atom at any real-time.

Lemma 5 *$e_{i,j}$ is in an atom if $j \neq 0$.*

Proof : Define $l = \bigcap \{\mathrm{L} \mid e_{i,j} \in \mathrm{L}\}$. Notice that there is at least one such L, namely, $C - W_0$ (recall that W_0 is the initial wavefront). For each molecule L with $e_{i,j} \in \mathrm{L}$, let W_{L} be a wave such that L begins on W_{L}.

We now show that l is a molecule. Clearly, l is a dipole set. Define $W = \bigcup \{W_{\mathrm{L}} \mid e_{i,j} \in \mathrm{L}\}$. We want to show that l begins on W. Suppose $e_{m,n} \in l$. We must show that either $e_{m,n-1} \in l$ or $e_{m,n-1} \in F(W)$. Suppose $e_{m,n-1} \notin l$. By definition of l, there exists a molecule $\mathrm{L}' \mid (e_{i,j} \in \mathrm{L}') \wedge (e_{m,n-1} \notin \mathrm{L}')$. Since $e_{m,n} \in l \subseteq \mathrm{L}'$, we must have $e_{m,n-1} \in F(W_{\mathrm{L}'})$. Therefore, $e_{m,n-1} \in W_{\mathrm{L}'} \subseteq W$ implying $e_{m,n-1} \in F(W)$.

Now to see that l is an atom, suppose L'' is a molecule and $\mathrm{L}'' \subseteq l$. We must show that $l \subseteq \mathrm{L}''$. Let W'' be a wave such that L'' begins on $F(W'')$. Note that $e_{i,j}$ cannot be in W'', otherwise $W'' - W_0$ is a molecule containing $e_{i,j}$, implying $\mathrm{L}'' \subseteq l \subseteq W'' - W_0 \subset W''$. Similarly, $e_{i,j}$ cannot be in the molecule $C - (W'' \cup \mathrm{L}'')$. Therefore, $e_{i,j} \in \mathrm{L}''$. By the definition of l, $l \subseteq \mathrm{L}''$. □

Theorem 2 *Each non-dummy event belongs to a unique atom.*

Proof : Follows from Lemmas 4 and 5. □

Figure 3 shows all the atoms in the computation of Figure 1.

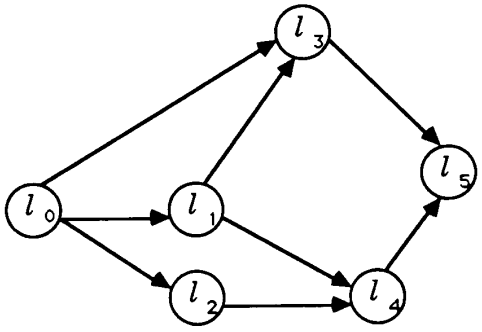

Figure 4: Atoms in G' corresponding to Figure 3.

4 Ordering of Basic Units

The ordering among the non-dummy events in C is a DAG $G = (V, E)$ where $V = C - \{e_{i,0} \mid i \in N\}$ and $E = \{(e_{i,j}, e_{k,l}) \mid e_{i,j} \rightarrow e_{k,l}\}$. E conveys causality within the computation. Analogous to the "happens before" relationship among events, we can define a relationship that conveys causality among atoms in the computation. G can be transformed to a DAG G' in which the nodes are atoms and the edges represent the causality relationship among these atoms. Define $l_0 = \{e_{i,0} \forall i \in N\}$, the dummy initial atom for the purpose of defining G'. $G' = (V', E')$ where
$(1) V' = \{l_0\} \bigcup \{atoms\ in\ the\ computation\}$.
$(2) E' = \{(l_1, l_2) \mid (l_1, l_2 \in V') \bigwedge (R(l_2) \bigcap l_1 \neq \phi)\}$.

Figure 4 shows the graph G' for the atoms in Figure 3. There is an edge (l_1, l_2) in G' iff $\exists i \in N | e_{i,j} \in l_1 \bigwedge e_{i,j+1} \in l_2$. Note that $\forall W | l_2 \subset W$, we have $l_1 \subset W$; using an inductive argument, if there is a path from l_1 to l_2, every wave that contains l_2 contains l_1.

A wavefront in G can be characterized as follows : Partition V' into two sets S_1 and S_2 such that $l_0 \in S_1 \bigwedge (\forall (l_1, l_2) \in E', l_2 \in S_1 \Longrightarrow l_1 \in S_1)$. Any ground state of the computation is a state along such a partition. This dependence is captured by the "occurs before" relation.

Definition 12 *For two atoms l_1 and l_2, l_1 "occurs before" l_2, denoted by $l_1 \mapsto l_2$ if either $R(l_2) \bigcap l_1 \neq \phi$ or $\exists l_3 | l_1 \mapsto l_3 \bigwedge R(l_2) \bigcap l_3 \neq \phi$.*

Theorem 3 *The "occurs before" relation is a strict partial ordering (transitive, irreflexive and anti-symmetric).*

Proof : Transitivity follows from definition.

"Occurs before" is irreflexive because (a) $R(l_1) \bigcap l_1 = \phi$ and (b) $\nexists l_2 \mid l_1 \mapsto l_2 \bigwedge R(l_1) \bigcap l_2 \neq \phi$. To prove (b), assume that such a l_2 exists. Let $R(l_1) \subseteq W_1$. Note that $l_1 \bigcap W_1 = \phi$. Since $l_2 \bigcap R(l_1) \neq \phi$, $l_2 \bigcap W_1 \neq \phi$. By Lemma 3, $l_2 \subset W_1$. However, if we let $l_1 \mapsto l_2$, then by applying the definition of $\mapsto$ recursively and using Lemma 3 each time, we see that $l_2 \subseteq \overline{W_1}$. Hence, l_2 cannot exist.

To show anti-symmetry, let us presume the contrary. If $l_1 \mapsto l_2 \bigwedge l_2 \mapsto l_1$, then using transitivity, $l_1 \mapsto l_1$ which cannot be since "occurs before" is irreflexive. So $\mapsto$ is anti-symmetric. □

An atom is executed in a computation only after all the atoms that occur before it have begun execution. Thus, the computation induces a partial order over all its atoms. From Lemma 2, it follows that the state of the DS after a prefix of this partial order is a ground state.

5 Detecting Ground States in a Computation

For any computation, we would ideally like to detect the atoms as they occur without disturbing the underlying computation, so that we can use the properties of the corresponding ground states. Detecting atoms (and hence ground states) as they occur involves high overhead of system resources and messages because it involves detecting a state over an undetermined subset of the system nodes without interfering with the underlying computation. However, we can create ground states by creating artificial molecules in the computation. It has been realized in [11] that to create a ground state for a distributed database, the computation would have to be affected or else parts of the database would need to be replicated. The conversation scheme [16] requires complete synchronization among participating nodes to create a ground state. It creates an instantaneous ground state at the cost of restricted interprocess communication. Instead of freezing the nodes, the transaction model of [5] uses a non-intrusive scheme to create a ground state of a distributed database. A copy of the database is created whenever a ground state is required. This copy can be discarded when the ground state has been computed.

A distributed algorithm that creates and records artificial ground states in an asynchronously communicating system with FIFO channels is given below. It assumes that one process is running on each node and that the network topology is such that there is a path from the initiator process to every other process. It can be spontaneously invoked by any process when the latest wavefront is needed. It freezes the computation to the extent that once a process receives a marker due to the algorithm, then until it sends markers and records its state, it does not send any messages. It is based on Chandy-Lamport's snapshot algorithm [3] and differs in two ways. (1) A process records its state when it has received a marker on each channel incident on and directed towards itself. (2) A process sends a marker on each channel incident on and directed away from itself and then records its state, and between the two, it does not send any messages.

- To initiate the algorithm, process p follows the Marker-Sending Rule when it is not already executing the algorithm.
- **Marker-Sending Rule:** For each channel incident on and directed away from p, p sends a marker. It

does not send any messages till the Recording Rule is executed.

- **Marker-Receiving Rule:** If a marker received is the first marker on any channel, p follows the Marker-Sending Rule. p does not accept a message on a channel on which a marker is received until the Recording Rule is followed.
- **Recording Rule :** p records the local state when a marker has been received on each channel incident on and directed towards p.

Theorem 4 *The above algorithm gives a ground state.*

Proof :
(Correctness) For the two process states recorded by processes across the ends of a channel, the channel state recorded is $\emptyset$. Let c be a channel from process p to process q. p sends a marker to q and does not send any message to q until p records its state. Due to FIFO channels, when q receives the marker from p, it has received all messages sent by p before p sent the marker. q records its state after it receives the marker from p and before receiving more messages from p. Thus, state of c is $\emptyset$ for process states recorded by p and q. This holds for every (p, q) pair. The rest of the proof is similar to the correctness of the algorithm in [3] and is left as an exercise.
(Termination) Each process eventually records its state. A process follows the Sending Rule when it gets its first marker. Hence, as long as there is a path from the initiator to every other node, each process follows the Sending Rule and a marker is sent on each channel. Because markers are not lost, they eventually reach and all processes follow the Recording Rule. □

The algorithm requires P messages where P is the number of edges in the graph of the DS topology. This algorithm can be extended to non-FIFO systems by using counters or "flush" messages [1].

A system with synchronous communication is a restricted case of an asynchronously communicating system. The algorithm of Bougé [2] is a CSP implementation that detects ground states in a synchronous system.

6 Significance of the Basic Unit

The basic unit of computation exists inherently in all computations. A computation which is a particular run of a distributed program is broken down into unique basic units at run-time based on the happens before relation of the events of the computation. All computations that are isomorphic [4] to each other with respect to all the system nodes can be broken down into the same unique atoms with the same occurs before relation because they exhibit the same happens before relation among their events. At the completion of execution of each basic unit, there is a wavefront. The global state of the system corresponding to this wavefront is a ground state. There is always a latest ground state in

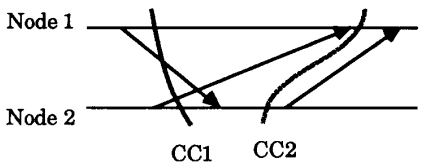

Figure 5(a)

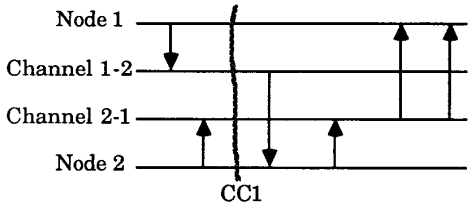

Figure 5(b)

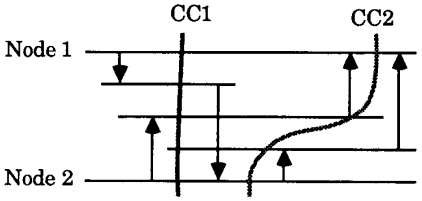

Figure 5(c)

Figure 5: (a) CC1 and CC2 are consistent cuts in an asynchronous computation. (b) If the above computation is represented with each channel as a node, only CC1 can be represented as a wavefront. (c)If each message is represented by a node, CC1 and CC2 can be represented as wavefronts.

the system. It advances each time the execution of a basic unit is completed. All the ground states through which a computation takes the DS are totally ordered in real-time in that computation.

Two possible computations of a program need not have the same C, but even if they do, they may comprise of different sets of basic units. Even if the two possible computations comprise of the same set of basic units, they may lead to different orderings in real-time among the occurrence of the same set of ground states. All these observations point to the richness of possible scenarios that are possible in a distributed computation. To detect the basic units in a computation as it unfolds necessarily requires observing the computation (so far) a posteori. This can have high overheads depending on the diversity of possible scenarios. As seen in the last section, it is even difficult to create the basic units artificially.

We would like to observe the system at points on the nodes where each node has finished computing some basic unit because all the channels are empty in the resulting state. Any wavefront (except W_0) is exactly such a set of observation points which identify a ground state. A ground state "isolates the past from the future". The cut identified by a ground state is stronger than a consistent cut [3, 15] in

an asynchronous system because the consistent cut allows message sends to be included in the cut even if the corresponding receives are not. A DS with synchronous communication [7, 12] restricts the concurrency as compared to a DS with asynchronous communication and allows each channel to have only one message in transit at any time. A computation that is isomorphic to a given computation with synchronous communication will have all communications represented as vertical lines in a timing diagram. Due to the semantics of synchronous communication, any global state that includes a send (receive) would also include its corresponding receive (send). Hence, every consistent cut in such a system corresponds to a wavefront and a global state is always a ground state.

Our approach to analysing a DS can capture the consistent cut [15] and the global state [3] in an asynchronous system also. A ground state is a restricted global state. We can represent a global state with possibly non-empty channels as the global state along a wavefront as follows : Represent each channel as a "channel" node. An event on a "channel" node takes place whenever a processor node sends or receives a message associated with that channel. For a message send, we have an instantaneous "message" transmission from a source processor node to the corresponding "channel" node. For a message receive, there is an instantaneous "message" transmission from the corresponding "channel" node to the destination processor node. The representation is similar to the one obtained using the I/O automata model [12]. Every wavefront in this representation denotes a global state with possibly non-empty channels in the DS. Every global state in the DS in which each channel has (not surprisingly) at most one message in transit can be represented as a wavefront in this representation. Figure 5(b) illustrates this representation for an asynchronous computation of Figure 5(a). To represent every global state as a wavefront, treat each channel as a number of "channel" nodes, one for each message sent on the channel. Figure 5(c) shows this representation for the computation in Figure 5(a). Each wavefront in this extended representation denotes a consistent cut across processor nodes and vice-versa.

Observe from Section 4 that any wavefront in a computation denotes some "consistent" cut in the graph G' and vice-versa if each node in G' is treated as an event and each edge in G' is treated as a message. Thus, we are directly supporting the concept of the consistent cut at a different level of atomicity.

The structure of the basic units depends on the concurrency in the DS. A DS with asynchronous communication over non-FIFO channels offers more concurrency than does the same system over FIFO channels. The increased concurrency permits more communication patterns and a greater variety of basic units are possible. In a synchronously communicating DS, the restrictions imposed by the synchronous nature of the computation reduce the multiplicity in communication patterns in the DS (from what it was under asynchronous communication) and the basic units of computation are simplified in terms of their composition. In such a system, each send and its receive form an atom as shown in Figure 6. Special topologies that serve to restrict the communication pattern also constrain the composition of the basic units in the computation. For a unidirectional token-passing ring topology with asynchronous communication over FIFO channels, each atom comprises of a send and its receive.

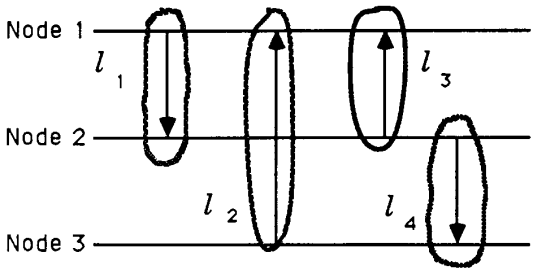

Figure 6: Atoms in a synchronously communicating sytem.

Reasoning about a general DS (with asynchronous communication over non-FIFO channels) is difficult because the reasoning has to be correct in spite of the vast number of possible scenarios. We see that by imposing restrictions such as having synchronous communication, FIFO channels, a special topology and a locking protocol, we restrict the possible concurrency in the DS. This introduces some discipline in the structure of the basic units and prevents some of the scenarios that would have been possible otherwise. This achieves simpler reasoning about the DS.

The ground state that exists along a wavefront is different from the global state at global termination [13]. For global termination, the following predicates should be true for the system : (a) there are no messages in transit, (b) each node has finished execution, (c) each node will not initiate communication. Along a wavefront, only (a) needs to be true.

6.1 Applications

6.1.1 Reasoning about Stable Properties

If all the computation messages are about a predicate x, then x takes a value on a stable basis only at a wavefront, e.g., termination can occur only on a wavefront. This is because on any other consistent cut, messages about x are still in transit. Knowledge of x attained along a wavefront is stronger than concurrent knowledge [15] because no information is in transit; each node knows everything intended for it to know.

If only some of the computation messages are about predicate x, then x takes a value on a stable basis only on a wavefront with respect to messages pertaining to x. To create such a wavefront, only the computation with respect to x need be affected. This concept needs to be developed.

6.1.2 Distributed Program Debugging

By observing the DS in a ground state, debugging [14] can be simplified. Monitoring the values of concerned predicates at snapshots along wavefronts gives knowledge about the concerned predicates in ground states. To draw an analogy to a uniprocessor system, it is reasonable to see the state of variables after each high-level instruction and not, say, between the micro-code instructions that execute the high-level instruction. Similarly it makes sense to know the values of variables at the boundary of (and not midway through) some basic unit in a distributed system. This boundary is always along a wavefront.

6.1.3 Checkpointing and Recovery in Distributed Databases

Nodes can recover independently from failure using pessimistic logging. If optimistic logging is performed and checkpoints are taken along a consistent cut [8, 9], rollback/redoing of events even before the checkpoints may be needed because such checkpoints have message sends without the corresponding receives. However, when checkpoints are taken at a wavefront, no node ever has to rollback/redo events before the checkpoint. This was first realized in the conversation scheme [16] and used in [5]. The latest checkpoint for the DS can be found by forcing a ground state. Alternatively, if it is feasible to detect individual molecules, only the nodes concerned in executing the molecule can advance their local checkpoints and do not have to coordinate with the rest of the nodes. The nodes can perform optimistic logging or none at all, and do not have to retain any log before the latest checkpoint.

7 Conclusions

We have identified a basic unit of computation in a DS. The phenomenon of basic units occurs naturally in all computations and is independent of the system model or the synchrony in communication. The structure of the basic units depends on the concurrency permitted in the system. The basic units in a computation are formed at run-time and are not easy to detect.

This work provides an understanding of the richness of possible scenarios that are possible in a distributed computation and the dependence of this richness on the structure of the basic units. It shows that the complexity of the structure of basic units depends on the level of concurrency in the DS which clearly determines the level of difficulty of reasoning. Thus the prevention of certain possible scenarios provides simplicity of reasoning in DSs.

The global state of the system after the execution of a unit of computation has been shown to have useful properties. Hence it is worthwhile to artificially create units of computation. We proposed a way of doing so.

References

[1] M. Ahuja. *Flush Primitives for Asynchronous Distributed Systems.* Information Processing Letters, 5-12, 34(1), 1990.

[2] L. Bougé. *Repeated Snapshots in Distributed Systems with Synchronous Communications and their Implementations in CSP.* Theoretical Computer Science, 145-169, 49, 1987.

[3] M. Chandy, L. Lamport. *Distributed Snapshots : Determining Global States Of Distributed Systems.* ACM Transactions on Distributed Systems, 63-75, 3(1), Feb. 1985.

[4] M. Chandy, J. Misra. *How Processes Learn.* Distributed Computing, 40-52, 1, 1986.

[5] M. Fischer, N. Griffeth, N. Lynch. *Global States in a Distributed System.* IEEE Trans. Software Engineering, 8(3), 198-202, May 1982.

[6] J. Hintikka. *Knowledge and Belief.* Cornell University Press, 1962.

[7] C. A. R. Hoare. *Communicating Sequential Processes.* Communications of the ACM,666-677, 21(8), August 1978.

[8] D. Johnson, W. Zwaenepoel. *Recovery in Distributed Systems Using Optimistic Message Logging and Checkpointing.* Proceedings of the 7th ACM Symposium on Principles of Distributed Computing, 171-181, 1988.

[9] R. Koo, S. Toueg. *Checkpointing and Rollback-Recovery for Distributed Systems.* IEEE Transactions on Software Engineering, 23-31, 13(1), Jan. 1987.

[10] L. Lamport. *Time, Clocks, and the Ordering of Events in a Distributed System.* CACM, 558-565, 21(7), July 1978.

[11] H. F. Li, K. Venkatesh, T. Radhakrishnan. *Global States of Distributed Systems.* Tech. Report, Concordia University, Montreal, Canada, 1986.

[12] N. Lynch, M. Tuttle. *An Introduction to Input/Output Automata.* MIT/LCS/TM-373, Nov. 1988.

[13] F. Mattern. *Algorithms for Distributed Termination Detection.* Distributed Computing, 2:161-175, 1987.

[14] B. Miller, J. Choi. *Breakpointing and Halting in Distributed Programming.* Proc. of the 8th Intl. Conference on Distributed Computing Systems, 316-323, 1988.

[15] P. Panengaden, K. Taylor. *Concurrent Common Knowledge: A New Definition Of Agreement for Asynchronous Systems.* TR 88-915, Computer Science, Cornell University, 1988.

[16] B. Randell. *System Structure for Software Fault Tolerance.* IEEE Transactions on Software Engineering, 1(2), 220-232, June 1975.