Universal Constructs in Distributed Computations

Ajay D. Kshemkalyani[1] and Mukesh Singhal[2]

[1] Dept. of EECS, University of Illinois at Chicago, Chicago, IL 60607-7053, USA
`ajayk@eecs.uic.edu`
[2] Dept. of CIS, The Ohio State University, Columbus, OH 43210, USA
`singhal@cis.ohio-state.edu`

Abstract. This paper identifies two classes of communication patterns that occur in distributed computations and explores their properties. It first examines local patterns, primarily *IO* and *OI intervals*, that occur at nodes in distributed computations. These local patterns form building blocks that are then used to define the global patterns, termed *segments* and *paths*, that occur across nodes in distributed computations. By controlling the predicates on the local patterns used to define segments and paths, various types of segments and paths can be defined. A number of key concepts and structures characterizing distributed computations are special cases of and are expressed in terms of the patterns identified.

1 Introduction

Analyzing the structure of a distributed computation helps to understand the concurrency and leads to a better design of distributed applications, algorithms, and systems. To this end, this paper identifies two classes of communication patterns that occur in every distributed computation and examines their properties. The first class of patterns consists of local patterns or intervals, primarily *IO* and *OI intervals*, that occur at processes [6]. These local patterns are specified in terms of messages received and messages sent by a process, and are distinguished by the order in which a pair of messages is sent and/or received by a process. Domain-specific predicates can be defined on how the interval at one process is related to the interval at another process. The use of such predicates on intervals at different processes allows intervals to be used as building blocks to formulate the second class of patterns, which is comprised of two global patterns, termed *segments* and *paths*. These global patterns occur across processes in a distributed computation and signify the flow of information and coupling among the events at different processes. Segments and paths generalize causal chains. While a causal chain only captures the causal relation, certain other message sequences also play a significant role in the analysis of a distributed computation. This paper generalizes segments and paths identified in [6]. By controlling the predicates on the intervals used to define segments and paths, different types of segments and paths can be defined.

Several key concepts and structures characterizing distributed computations are special cases of and can be expressed using the identified patterns. These patterns are shown to be useful in areas such as synchronous and causally ordered communication [4], transfer of knowledge [3], concurrency measures [5], necessary and sufficient conditions for a consistent global state [2, 9] which is useful in checkpointing and recovery [1], and distributed deadlock detection [6].

Section 2 gives the system model. Section 3 defines the local patterns and gives their properties. Section 4 defines the global patterns that occur across nodes and shows their applications. Section 5 concludes. The full paper is in [7].

2 System Model

The system is a network of N nodes (sites) with a logical channel between each pair of nodes. The nodes communicate by passing messages over the logical channels and do not share memory. We assume without loss of generality that each node in the system has one process running on it. Hence, nodes are synonymous to processes. Process execution and message transfers are asynchronous. Messages are delivered reliably but not necessarily in the order sent.

The execution of a computation at a node is modeled by three types of events: message send events, message receive events, and internal events. Let s_i^x and r_j^x denote the send and the receive events at which the message with label x is sent at node i and received at node j, respectively. The superscript and/or subscript will be omitted when it is not important. Let $dest(s_i^x)$ denote the destination of the message sent at s_i^x. An execution of a distributed computation associates with each node i a totally ordered set C_i of events. Let $C = \bigcup C_i$ be the possibly infinite set of all events. The state of a node is defined by the values of the variables associated with its computation, which are a function of the history of events executed by it at any time. A distributed computation is represented by the poset $(C, \prec)$, where $\prec$ is the causality relation on C [8].

3 Local Communication Patterns: Intervals at a Node

This section formalizes the local communication patterns that occur at nodes. The next section shows how these patterns are used as building blocks to formulate two global patterns that occur across nodes.

At the time a node i sends a message at s_i, an "outward dependency" gets established at i. At the time a node i receives a message at r_i, an "inward dependency" gets established at i. An *interval* at a node is the period between the times that two such dependencies get established [6]. There are two main types of intervals, shown in Fig. 1 (a) and (b), based on whether the inward dependency is established before the outward dependency or vice-versa. The former interval is an *IO interval* and the latter is an *OI interval.* Analogous to IO and OI intervals, *II intervals* and *OO intervals* can also be defined.

An *interval begins* at node i whenever one of the following two events occurs (see Fig. 1): (a) i receives a message from some node j, or (b) i sends a message to some node k. For the two cases (a) and (b) in which an *interval* begins, the IO or OI *interval completes* when, in case (a), i sends a message to some node k, and in case (b), i receives a message from some node j, respectively. Similar explanations hold for II and OO intervals.

The formation of an interval at a node signifies the participation of the node in global communication patterns that span across nodes. Note that intervals

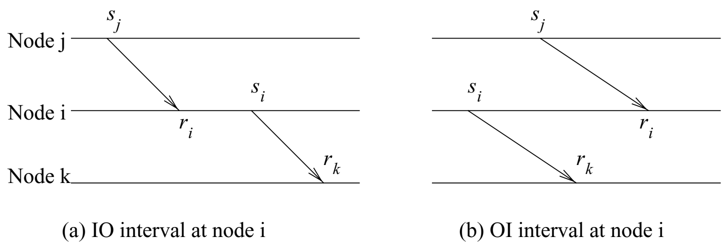

Fig. 1. IO and OI intervals [6].

at a node can overlap. For example, in Fig. 2, the following pairs of intervals overlap at node 3: (i) the OI interval between s_3^3 and r_3^2, the OI interval between s_3^3 and r_3^5; (ii) the IO interval between r_3^2 and s_3^6, the IO interval between r_3^5 and s_3^6; and (iii) the OI interval between s_3^3 and r_3^5, the IO interval between r_3^2 and s_3^6. There are numerous intervals at each process in the computation.

Each node has a set of application-specific semantic-defined "distinguished" events that are identified by monotonically increasing functions such as the sequence number of the events at the node. An example of a distinguished event is a checkpointing of the local state of a node. The time span from the xth to the $(x + 1)$th distinguished event at node i is called the xth duration at i. IO or OI intervals of interest to an application are those that satisfy a certain application-specific relationship on the durations in which the send and receive events identifying the interval occur. Each duration x at node i is associated with a predicate $\Phi_{i,x}$ which is true only during that duration. The duration at node i in which an event e_i occurs is denoted by $D(e_i)$. A send event s_i and a receive event r_i can be related at a node i in one of the following ways:

1. $D(s_i) - D(r_i) = 0$. Events s_i and r_i belong to the same duration and identify an IO or an OI interval, based on whether $r_i \prec s_i$, or vice-versa, resp..
2. $D(s_i) - D(r_i) > 0$. In this case, events s_i and r_i identify an IO interval.
3. $D(s_i) - D(r_i) < 0$. In this case, events s_i and r_i identify an OI interval.

The semantics attached to an IO or OI interval can be of the following types (classification of global communication patterns uses these semantics):

1. No semantics is attached to the events of an OI or an IO interval. No conditions are imposed on the relation between $D(s_i)$ and $D(r_i)$.
2. The s_i and r_i events of an interval satisfy constraints on $D(s_i)$ and $D(r_i)$, the local durations to which they belong. For example, events of an IO interval are in two different local durations, but the events of an OI interval are in the same duration.
3. The events of an interval satisfy certain constraints on the durations they belong to and on the durations of the events in their causal past.

4 Global Communication Patterns: Paths and Segments

This section defines global patterns that span nodes in a computation. It then shows that several key concepts and structures characterizing distributed computations are special cases of and can be expressed using these patterns.

Domain-specific predicates can be defined on how the IO or OI interval at one node is related to the IO or OI interval at another node. The use of such predicates on IO and OI intervals at different nodes allows IO and OI intervals to be used as building blocks to formulate the global patterns: segments and paths. These global patterns occur across different nodes and signify a sequence of message exchanges such that any two adjacent messages in the sequence are related at a node by an IO or an OI interval. By controlling the predicates (or conditions) on the IO and OI intervals used to define segments and paths, different types of segments and paths can be defined.

Based on the semantics attached to the events identifying IO and OI intervals, three versions of segments and paths are presented. The first version (Sect. 4.1) is for a general computation where no restrictions are imposed and any s_i and any r_i events at a node i participate in OI and IO intervals. This version has applications in characterizing distributed computations by identifying structures like a crown, deriving concurrency measures, and analyzing knowledge transfer. In the second version (Sect. 4.2), distinguished events are assigned values of a monotonically nondecreasing function. This version has applications in characterizing global checkpoints. In the last version (Sect. 4.3), the distinguished events signify participation in a stable property. This version has applications in characterizing stable properties like distributed deadlocks.

Before defining the global patterns, we introduce some primitive predicates (conditions) on a distributed computation. The global patterns for various semantic models are defined using these conditions. In a computation, at any instant, there could have existed a sequence $\langle s_{i_1}, s_{i_2}, \ldots, s_{i_n} \rangle$ of send events on nodes $i_j \in \{i_1, i_2, \ldots, i_n\}$ satisfying a combination of the following conditions (henceforth, $i_j \in \{i_1, i_2, \ldots, i_n\}$):

(C1) Convey predicate to successor: $D(s_{i_j}) = x_{i_j}$ and $dest(s_{i_j}) = i_{j+1}$, for $1 \leq j \leq n-1$.

(C2) Predicate conveyed from predecessor: A node i_j (except for $j = 1$) has received the message sent by i_{j-1} at $s_{i_{j-1}}$ before s_{i_j}.

(C3) No local violation of predecessor's predicate: Each node i_j (except for $j = 1$) has not invalidated the predicate $\Phi_{i_{j-1}, x_{i_{j-1}}}$ at node i_{j-1}.

(C4) No violation of predecessor's predicate: A node i_j (except for $j = 1$) has not received any message, in the causal past of which i_{j-1}'s predicate $\Phi_{i_{j-1}, x_{i_{j-1}}}$ got invalidated.

(C5) Local predicate valid in duration: Each node i_j is in its x_{i_j}th duration and $\Phi_{i_j, x_{i_j}}$ is currently true.

(C6) Duration of send event does not occur before duration of receive event: $D(s_{i_j}) \geq D(r_{i_j})$.

4.1 Segments and Paths for General Computations

In a general computation, no semantics is attached to the events of an interval and no constraints are imposed on the relation between $D(s_i)$ and $D(r_i)$.

Definition 1. *A* "segment" *for a general computation, denoted $S_g(s_{i_1}, r_{i_{n+1}})$, is a sequence of events $\langle s_{i_1}, s_{i_2}, \ldots, s_{i_n} \rangle$ satisfying (C1) $\bigwedge$ (C2).*

Every event in a segment occurs at a node that has sent a message to the node at which the successor event in the segment occurs. (Henceforth, a reference to "a node on a segment/path" will mean "a node with an event on a segment/path".) Moreover, when a node i_j sends the message at s_{i_j} (as per (C1)), the message sent at the previous event $s_{i_{j-1}}$ in the sequence has been received (as per (C2)). Therefore, a segment denotes a sequence of nodes such that the dependencies on their successor nodes in the segment are created sequentially. That is, $\forall i_j$:$1 \leq j < n$ :: $s_{i_j} \preceq s_{i_{j+1}}$. A segment thus denotes a causal chain of messages in which the events signify completed IO intervals.

For a sequence of events $\langle s_{i_1}, s_{i_2}, \ldots, s_{i_n} \rangle$ such that $dest(s_{i_j}) = i_{j+1}$ for $1 \leq i \leq n-1$, it may happen that $\exists j$: $s_{i_j} \not\preceq s_{i_{j+1}}$, that is, node i_{j+1} has an OI interval. A *path* is defined next to capture such a sequence of events.

Definition 2. *A* "path" *for a general computation, denoted $P_g(s_{i_1}, r_{i_{n+1}})$, is a sequence of events $\langle s_{i_1}, s_{i_2}, \ldots, s_{i_n} \rangle$ satisfying (C1).*

The formation of an interval at a node signifies the participation of the node in a path or a segment. In a path, successive messages are related by either an IO or an OI interval. In a segment, successive messages are related only by IO intervals. Thus, the successive events in a sequence at which outward dependencies are established satisfy a weaker causal relationship in a path than in a segment. A path may contain several segments; a segment is always a path.

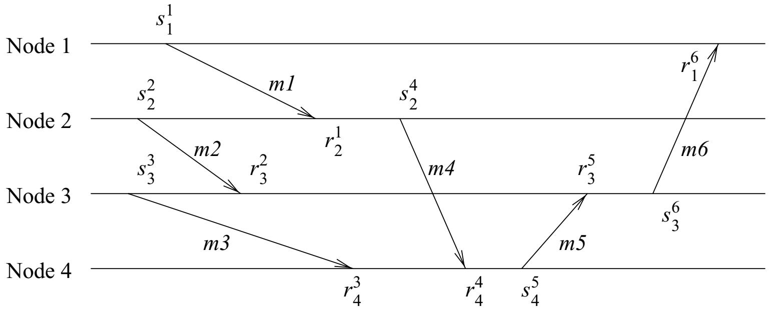

Fig. 2. An example computation.

Fig. 2 gives examples of paths and segments. Some segments are: $\langle\, s_1^1, s_2^4, s_4^5, s_3^6 \,\rangle$, $\langle\, s_2^2, s_3^6 \,\rangle$, $\langle\, s_3^3, s_4^5, s_3^6 \,\rangle$, and all subsequences of the above. By definition, each segment is a path. The following are some paths with at least one OI interval: $\langle s_1^1, s_2^2, s_3^3 \,\rangle$, $\langle\, s_1^1, s_2^2, s_3^6 \,\rangle$, $\langle\, s_2^2, s_3^3, s_4^5, s_3^6 \,\rangle$. Subsequences of these are also paths.

A *maximal path* is a path which cannot be extended by the addition of a send event at either end. For example, in Fig. 2, $\langle\, s_1^1, s_2^2, s_3^3, s_4^5, s_3^6 \,\rangle$ is a maximal path. The longest maximal path in the computation is $\langle\, s_2^2, s_3^3, s_4^5\ s_3^6, s_1^1, s_2^4 \,\rangle$ which happens to consist of all the messages in the computation. A *maximal segment* is defined likewise. In Fig. 2, $\langle\, s_1^1, s_2^4, s_4^5, s_3^6 \,\rangle$ is a maximal segment.

Maximal segments and maximal paths are useful concepts in analyzing properties of a distributed computation. A maximal segment is a causal chain that signifies the maximum length of the serial execution of “thread of control” represented by the segment. On the other hand, a maximal path whose events are related by OI intervals at nodes provides a measure of the concurrency in a computation. The higher the number of OI intervals, the higher the concurrency in the computation. The ratio of the average of the sizes of maximal paths to the average of the sizes of maximal segments in a distributed computation is a good indicator of the concurrency in the computation [5].

We next show how segments and paths can be used to express some communication patterns that are important in analyzing distributed computations.

Realizable Synchronous Computations: The Crown Criterion. Charron-Bost et al. observed that a distributed algorithm designed to run correctly on asynchronous systems (called *A-computations*) may not run correctly on synchronous systems – an algorithm that runs on an asynchronous system may *deadlock* on a synchronous system [4].

A-computations that can be realized under synchronous communication are called *Realizable with Synchronous Communication* (RSC) computations. Formally, a computation C is RSC if there exists a non-separated linear extension of the poset $(C, \prec)$. A non-separated linear extension of $(C, \prec)$ is a linear extension of $(C, \prec)$ such that for each pair of send event s and corresponding receive event r, the interval $\{ x \in C \mid s \prec x \prec r \}$ is empty. [4] showed that RSC computations are a proper subset of causally ordered computations, which are a proper subset of FIFO computations.

Charron-Bost et al. [4] developed a criterion (called the *crown criterion*) to determine if an A-computation can be realized on a system with synchronous communication. This criterion uses a structure called *crown*, defined next.

Definition 3. *Let C be a computation. A crown of size k in C is a sequence $\langle$ (s^i,r^i), $i \in \{$ 0, ..., k-1 $\}$ $\rangle$ of pairs of corresponding send and receive events such that:* $s^0 \prec r^1$, $s^1 \prec r^2$, ... $s^{k-2} \prec r^{k-1}$, $s^{k-1} \prec r^0$.

Charron-Bost et al. [4] showed that a computation is RSC iff it contains no crown.

Fig. 3 shows a crown having six pairs of corresponding send and receive events (s^i, r^i), $i \in [0, 5]$. There is also a causal chain from s^i to $r^{(i+1) mod\, 6}$, for $i \in [0, 5]$. Defn. 3 specifies the constraints between s^i and $r^{(i+1) mod\, k}$, for $i \in [0, k-1]$. Each such constraint simply represents a segment $S_g(s^i,\, r^{(i+1) mod\, k})$. We next define crowns in terms of segments.

Definition 4. *In terms of segments, a crown of size k in a computation is a sequence $\langle$ (s^i,r^i), $i \in \{$ 0, ..., k-1 $\}$ $\rangle$ of pairs of corresponding send and receive events such that $\forall i \in [0, k-1], S_g(s^i, r^{(i+1) mod\, k})$. (To simply notation, the crown will also be expressed by just the conditions $\{S_g(s^i, r^{(i+1) mod\, k}) : i \in [0, k-1]\}$.)*

Example 1 (Crown in Fig. 3): $CROWN = \{S_g(s^i, r^{(i+1) mod\, k}) : i \in [0, 5]\}$.
Refinement of Defn. 4: Defn. 4 expresses a crown of size k in terms of k segments. A crown of size k can generally be expressed in terms of less than k segments and paths.

A segment $S_g(s^i,\, r^j)$ such that events s^i and r^j lie on the same node is called a *local* segment. Note that in Fig. 3, (i) segments $S_g(s^2,\, r^3)$ and $S_g(s^3,\, r^4)$ are

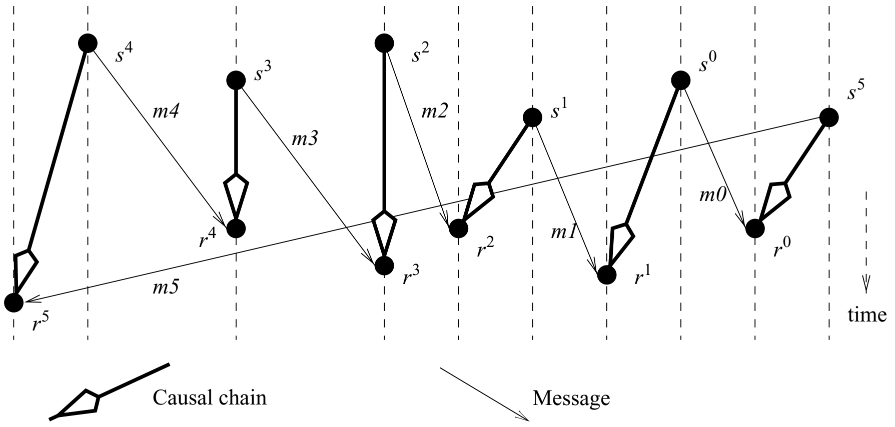

Fig. 3. A crown of size 6.

local segments and (ii) these two segments are connected by message (s^3,r^3). In this situation, segments $S_g(s^2, r^3)$ and $S_g(s^3, r^4)$ can be represented by path $\langle s^4, s^3, s^2 \rangle$. Consequently, the conditions represented by segments $S_g(s^2, r^3)$ and $S_g(s^3, r^4)$ in the expression of the crown can be equivalently stated in terms of path $\langle s^4, s^3, s^2 \rangle$ (which happens to contain only OI intervals). Given a crown, we present an algorithm that replaces clusters of local segments connected by messages, by equivalent paths. This algorithm compacts consecutive segments into paths wherever possible. (Such paths consist of OI intervals only and a cyclic path with OI intervals only is always a crown.)
A Crown-Compaction Algorithm:

1. $CR\_ALT = CROWN$;
2. Identify each maximal sequence of consecutive integers, modulo k, from x to y satisfying $\forall\ j \in [x, (y) mod\ k]$, s^j and $r^{(j+1) mod\ k}$ occur on the same node. For each such sequence, do the following.
   (a) $CR\_ALT = CR\_ALT \setminus \{S_g(s^i, r^{(i+1) mod\ k}) : i \in [x, (y) mod\ k]\}$
   (b) $CR\_ALT = CR\_ALT \bigcup \{\langle s^{(y+1) mod\ k}, s^y, s^{(y-1) mod\ k}, \ldots\ldots s^{(x+1) mod\ k}, s^x \rangle\}$

Example 1 (contd.): In the crown in Fig. 3, s^2 and r^3 lie on the same node, and s^3 and r^4 lie on the same node. As there is a range of consecutive integers $[x, y] = [2, 3]$ such that $\forall i \in [2, 3]$, s^i and $r^{(i+1) mod\ k}$ lie on the same node, segments $S_g(s^2, r^3)$ and $S_g(s^3, r^4)$ can be replaced by path $\langle s^4, s^3, s^2 \rangle$. Hence, $CR\_ALT = \{S_g(s^i, r^{(i+1) mod\ k}) : i \in \{0, 1, 4, 5\}\} \bigcup \{\langle s^4, s^3, s^2 \rangle\}$.

Thus, a crown which is an example of various structures in distributed computations can be expressed more compactly in terms of paths and segments.

Knowledge Transfer. Knowledge in distributed systems refers to the states of the nodes and is defined as temporal and spatial predicates over the variables of the nodes. Knowledge plays a significant role in the evaluation of global predicates, debugging, monitoring, establishing breakpoints, evaluating triggers, industrial process control, and controlling a distributed execution [11].

Knowledge is transferred among nodes through send and receive events [3]; the extent of knowledge dissemination is determined by the message communication pattern among nodes and is identified by the causality relation between events. A segment from event e_i to event e_j signifies the flow of knowledge of node i's state preceding event e_i to all the events following e_j. In Fig. 2, messages forming segment $\langle s_1^1, s_2^4, s_4^5 \rangle$ transfer the knowledge about the local state of node 1 just before event s_1^1 to event r_3^5. A path with an OI interval denotes a disrupted transfer of knowledge among the nodes along the path. In Fig. 2, knowledge about the local state of node 1 just before event s_1^1 is not transferred to event r_3^2. The knowledge transfer is disrupted at node 2 due to the OI interval formed by s_2^2 and r_2^1. Thus, paths and segments are useful tools to identify the extent of knowledge transfer.

4.2 Segments and Paths for Monotonically Nondecr. Functions

In distributed computations with monotonically nondecreasing functions at nodes (e.g., the local clock time at the occurrence of an event [8]), the distinguished events at a node are associated with monotonically nondecreasing values.

The definition of a segment for a monotonically nondecreasing function (Defn. 5) is the same as for a general function (Defn. 1). The definition of a path for a monotonically nondecreasing function (Defn. 6) differs from the corresponding Defn. 2 in that the events of an OI interval must belong to the same duration.

Definition 5. *A* "segment" *for a monotonically nondecreasing computation, denoted $S_m(s_{i_1}, r_{i_{n+1}})$, is a sequence of events $\langle s_{i_1}, s_{i_2}, \ldots, s_{i_n} \rangle$ satisfying (C1) $\bigwedge$ (C2).*

Definition 6. *A* "path" *for a monotonically nondecreasing function, denoted $P_m(s_{i_1}, r_{i_{n+1}})$, is a sequence of events $\langle s_{i_1}, s_{i_2}, \ldots, s_{i_n} \rangle$ satisfying (C1) $\bigwedge$ $(C6)$.*

A *closed path* for a monotonically nondecreasing function is a path $P_m(s_{i_1}, r_{i_{n+1}})$ such that events s_{i_1} and $r_{i_{n+1}}$ occur at the same node (i.e., $i_1 = i_{n+1}$).

Necessary and Sufficient Conditions for a Global Snapshot: Zigzag Paths. Checkpointing is used in fault-tolerant computing [1], and parallel and distributed debugging [11]. Each node can take local checkpoints asynchronously; a consistent global checkpoint is constructed by chosing a local checkpoint from each node. Checkpoints are the "distinguished events" which demarcate consecutive durations at nodes. The xth duration (or xth *checkpoint interval*) at a node denotes the computation from its xth to its $x+1$th checkpoint.

An important problem is to determine if an arbitrary set of local checkpoints belongs to a consistent global checkpoint [2]. Netzer and Xu used the zigzag path, a generalization of Lamport's causality relation [8], and showed that two local checkpoints cannot lie on a consistent global checkpoint iff a zigzag path exists between the checkpoints [9]. Let $C_{i,x}$ denote the xth local checkpoint at node i and let $e_{i,x}$ denote the event of taking $C_{i,x}$.

Definition 7. *A zigzag path exists from $C_{i,x}$ to $C_{j,y}$ iff there are messages m_1, m_2, …, m_n ($n > 1$) such that*

1. *m_1 is sent by node i after $C_{i,x}$*
2. *if m_k ($1 \leq k < n$) is received at node r, then m_{k+1} is sent by r in the same or a later checkpoint interval*
3. *m_n is received by process j before $C_{j,y}$.*

In Fig. 4, $m1$, $m2$, and $m3$ form a zigzag path from checkpoint C_{11} at node 1 to checkpoint C_{32} at node 3. Likewise, $m4$, $m5$, and $m6$ form a zigzag path from checkpoint C_{12} at node 1 to checkpoint C_{42} at node 4. Note from Defn. 7 that a zigzag path is a chain of messages that are connected by OI or IO intervals at nodes. Thus, a zigzag path is nothing but a "path" (Defn. 6) and can be expressed using paths as follows:

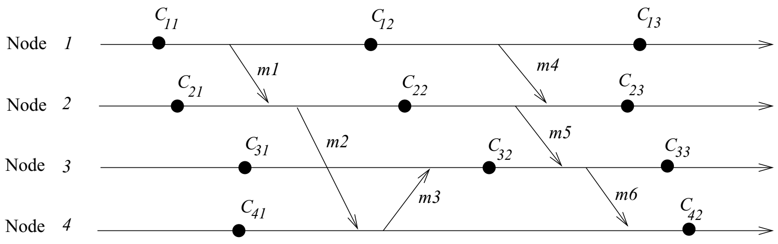

Fig. 4. Zigzag paths.

Definition 8. *A zigzag path exists from $C_{i,x}$ to $C_{j,y}$ iff $\exists$ P_m = $\langle s_{i_1}, s_{i_2}, \ldots, s_{i_n} \rangle$ such that (i) $e_{i,x} \prec s_{i_1}$, and (ii) a message sent at s_{i_n} to j is received before $e_{j,y}$.*

A checkpoint is on a Z-cycle iff there is a zigzag path from the checkpoint to itself. In Fig. 4, checkpoint C_{32} lies on a Z-cycle consisting of messages $m6$ and $m3$. A checkpoint can be part of a consistent snapshot iff it is not involved in a Z-cycle. Observe that a Z-cycle is nothing but a closed path.

4.3 Segments and Paths for Stable Properties

A stable property is a property of the system state such that once it becomes true, it continues to hold unless there is external intervention [10]. Examples of such properties are deadlocks, termination of a computation, etc. In this section, segments and paths are defined for stable properties, with special emphasis on deadlocks [6]. A similar approach can be used for other stable properties.

Conditions for Deadlocks. We consider deadlocks in the request-reply model. In this model, a process sends a request and blocks until it receives a reply to its request. "Distinguished" events at a node are the events at which a node sends a request and blocks waiting for a reply. The predicate $\Phi_{i,x}$ stands for "node i is blocked on the request it sent at its xth distinguished event". This predicate becomes true at the start of the duration between two distinguished events and becomes false on the receipt of the reply at some time before the next distinguished event. In this context, a segment and a path are defined next [6].

Definition 9. *A* "segment" *in the request-reply model, denoted $S_d(s_{i_1}, r_{i_{n+1}})$, is a sequence of events $\langle s_{i_1}, s_{i_2}, \ldots, s_{i_n} \rangle$ satisfying the following conditions :*
(I) (C1) $\bigwedge$ (C2) $\bigwedge$ (C3) $\bigwedge$ (C4). /* *conditions on distinguished events.* */
(II) (C5). /* *conditions when the system is observed.* */

Definition 10. *A* "path" *in the request-reply model, denoted $P_d(s_{i_1}, r_{i_{n+1}})$, is a sequence of events $\langle s_{i_1}, s_{i_2}, \ldots, s_{i_n} \rangle$ satisfying the following conditions:*
(I) (C1) $\bigwedge$ ((C2) $\Longrightarrow$ ((C3) $\bigwedge$(C4))). /* *conditions on distinguished events.* */
(II) (C5). /* *conditions when the system is observed.* */

Condition (C2) indicates that the request sent at $s_{i_{j-1}}$ has been received before s_{i_j}. Condition (C3) indicates that i_j has not sent back a reply to i_{j-1} and thus has not invalidated $\Phi_{i_{j-1}, x_{i_{j-1}}}$. By Condition (C4), i_j has not received a message indicating that i_{j-1} got unblocked, i.e., $\Phi_{i_{j-1}, x_{i_{j-1}}}$ got invalidated. As no node in a distributed system has instantaneous knowledge of the entire system, while declaring a segment/path, it must be ensured that the nodes that are believed to be blocked are still blocked; Condition (C5) asserts this. A detailed explanation of Defns. 9 and 10 is given in [6].

Paths in which each node is blocked waiting for a reply from its successor and the last node never receives a reply denote deadlocks.

Definition 11. *A* closed path *is a path $P_d(s_{i_1}, r_{i_{n+1}})$ such that events s_{i_1} and $r_{i_{n+1}}$ occur at the same node (i.e., $i_1 = i_{n+1}$).*

A closed path denotes a deadlock because no node with an event on the closed path will ever receive a reply and get unblocked. A closed path has at least one OI interval, i.e., at least two segments. Condition (C5) helps to ensure that false deadlocks are not detected.

5 Conclusion

We identified two classes of universal communication patterns in distributed computations. IO, OI, II and OO intervals are local patterns that occur at nodes, whereas paths and segments are global patterns which occur across nodes in a distributed computation and are defined in terms of the local patterns. It is seen that the communication patterns identified are universal to all distributed computations. We showed that a number of key concepts and structures characterizing distributed computations are special cases of the proposed patterns and can be expressed using these patterns. By controlling the predicates on local patterns used to define segments and paths, different types of segments and paths can be defined to address the needs of other applications also.

References

[1] B. Bhargava, S.R. Lian, Checkpointing and rollback recovery in distributed systems – an optimistic approach, *Proc. 7th IEEE SRDS*, 3-12, Oct. 1988.

[2] K. M. Chandy, L. Lamport, Distributed snapshots: Global states of a distributed system, *ACM Trans. Comput. Systems*, 3(1):63-75, 1985.

[3] K. M. Chandy, J. Misra, How processes learn, *Distributed Computing*, 1, 40-52, 1986.

[4] B. Charron-Bost, F. Mattern, G. Tel, Synchronous, asynchronous, and causally ordered communication, *Distributed Computing*, 9(4):173–191, 1996.

[5] C. J. Fidge, A simple run-time concurrency measure, In: T. Bossomaier et al. (Eds.), *The Transputer in Australasia (ATOUG-3)*, 92-41, IOS Press, 1990.

[6] A. D. Kshemkalyani, M. Singhal, On characterization and correctness of distributed deadlock detection, *Journal of Parallel and Distributed Computing*, 22(1), 44-59, July 1994. (Tech. Rep. TR-06/90-TR15, Ohio State Univ., 1990.)

[7] A. D. Kshemkalyani, M. Singhal, Universal constructs in distributed computations, *Technical Report 29.2136*, IBM, March 1996.

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

[9] R. Netzer, J. Xu, Necessary and sufficient conditions for consistent global snapshots, *IEEE Trans. on Parallel and Distributed Systems*, 6(2):165-169, 1995.

[10] A. Schiper, A. Sandoz, Strong stable properties in distributed systems, *Distributed Computing*, 8:93-103, 1994.

[11] M. Spezialetti, R. Gupta, Debugging distributed programs through the detection of simultaneous events, *Proc. 14th IEEE ICDCS*, 634-641, June 1994.