Logical Time in Distributed Software Systems

I. A. Zhuklinets * and D. A. Khotimsky**

* Mozhaiskii Military Space-Engineering University, St. Petersburg, Russia
e-mail: jkl@solaris.ru

** Bell Labs - Lucent Technologies, Holmdel, New Jersey, USA
e-mail: dkhotimsky@lucent.com

Abstract—This paper presents a survey of implementation of logical time in asynchronous distributed systems. We provide an argument that justifies the use of logical time as a mechanism for detecting causal relationships between events. Further, we formally introduce the notion of a logical time system (a logical clock) and proceed to discuss the properties of the scalar, vector, and matrix clocks. Finally, we consider the modifications of the vector clock that reduce the average communication overhead while retaining the property of isomorphism.

1. INTRODUCTION

The theory of distributed and parallel computations owes its intensive development to the rapid advances in telecommunications infrastructure (e.g., adaptation of the conventional voice networks to data communication, creation of the high-speed digital networks), as well as to the use of this infrastructure in offering of new information services. The need for distributed software systems emerges in various application domains, such as distributed databases, distributed information processing, distributed computations and modeling, and automatic process and mobility control.

The persistent efforts in the field of distributed and parallel computations laid the foundations of the theory and clearly outlined the main directions of research. These research activities are primarily focused on solving pragmatic problems belonging to the two interconnected areas. The first area encompasses specification, analysis, and verification of the distributed software systems; the other one is concerned with development of the techniques and tools allowing to overcome the inherent difficulties of the asynchronous process interaction. While the issues of the first area have been, to some extent, addressed in numerous Russian scientific and technical publications, the issues of the second area have received much less attention.

The characteristic features of asynchronous distributed systems include the absence of a common reference periodic process (i.e., global physical time), indeterminism and inconsistency of the relative process execution speeds, uncertainty in determining the epochs when the messages are sent and the order in which they arrive, unbounded message transmission delays.

At present, one of the major instruments (if not the only one) that can help to effectively solve interaction and synchronization problems is the logical clock. The concept of logical time was first introduced by L. Lamport in his landmark paper, published in 1978 [1]. Its objective was to formalize the notion of one event happening before another in a network system where the real physical time is not available or its use is impractical for some reason. In the following years, the concept of logical time attracted the attention of numerous researchers and software developers. It was extended, generalized, and refined many times.

The goal of this survey is to present a systematic account of the problems connected with the use of logical time in asynchronous distributed systems. The paper is structured as follows. Section 2 reviews some basic concepts, introduces logical relations between events associated with the notion of causal precedence, and gives the definition of an event structure for a distributed system with the totally ordered event sets. Further, this section explains the rationale behind the logical (or subjective) time as a mechanism for determining logical relations between events. Section 3 gives the formal definition of a logical time system (logical clock) and elaborates on the properties of scalar, vector, and matrix systems. Section 4 deals with the modifications of the vector clock that allow to reduce the average communication overhead while preserving the isomorphism property. Section 5 lists the examples of applications for which it has been possible to obtain efficient solutions based on the logical clock and outlines the range of problems related to logical time that fall outside the scope of this survey.

---

1 This is a revised version of the English translation of the article by I.A. Zhuklinets and D.A. Khotimsky originally published in Programming and Computer Software, 2000, vol. 26, no. 6, under the title “Logical Time in Distributed Software Systems.” The article is re-published due to errors made in the original translation. Our apologies to the authors of this paper and the readers of the journal.
2. RELATIONS IN THE SET OF EVENTS AND LOGICAL TIME

A set of processes that execute in spatially distinct nodes and can exchange information between each other is referred to as a distributed system. A process may be associated with a program, a group of programs, or a single thread in a multithread software complex. This paper focuses on fault-free asynchronous distributed systems with unicast message passing. A message-passing distributed system is modeled by a directed graph

\[ D = (P, C), \]

where \( P \) is the set of nodes, which correspond to the processes of the system, and \( C \) is the set of edges corresponding to the communication channels. Such systems are also called network systems (as opposed to models with shared memory and rendez-vous). Without loss of generality, we assume that the graph under consideration is fully connected and every edge represents a full-duplex channel with random independent finite delays.

In the general case, a process \( P \) is defined by the set \( S \) of states, the subset \( S_i \) of at least one initial state, the set \( M \) of incoming and outgoing messages including the empty message \( \emptyset \), and the function \( Trans \) of transitions and outputs that maps each state \( s \in S \) along with incoming message \( m \in M \) to a pair \((s', m')\), where \( s' \) is the new state of the process and \( m' \) is the outgoing message:

\[ P = (S, S_0, M, Trans). \]

A state transition represents an event. An ordered sequence of events of process \( P \) constitutes its computational history \( (h_i) \) (or just history for the sake of brevity):

\[ h_i = (e_{i0}, e_{i1}, \ldots, e_{ij}, \ldots), \]

where event \( e_{ij} \) is said to immediately precede event \( e_{i,j+1} \). The union of the events of all processes in a distributed system constitutes the event set \( E \) of this system:

\[ E = \bigcup_{i=1}^{n} E_i, \quad \text{where} \quad E_i = \{e_{ij}\}_{j=0, 1, \ldots}. \quad (1) \]

Event \( e_{ij} \in E \), for any \( j \), will be referred to as a proper event of process \( P_i \). All the events in an asynchronous distributed system are classified into two groups: internal events and communication events. Communication events are further subdivided into two classes: sending a message and receiving a message. Let \( e \) be some event in the system. We shall use the following notation:

- \( \text{Proc}(e) \) is the identifier of the process to which event \( e \) belongs;
- \( Type(e) \) is the event type, taking one of the four values:
  - \( INIT \) (the event of process initialization),
  - \( LOCL \) (local, or internal, event),
  - \( SEND \) (sending a message),
  - \( RECV \) (receiving a message).

Graphically, an execution of a distributed system can be represented with a time diagram where each horizontal line corresponds to the time axis of a process. Proper events of the process \( P \) are denoted by dots on the \( i \)-th time axis. Each send message event is connected to the corresponding receive message event with a directed line segment. An example of an execution of a distributed system with three processes is shown in Fig. 1.

Consider the logical relations in the event set \( E \) of a distributed system. The trivial equivalence relation in the set \( E \) is called the coincidence relation \( (a \equiv b) \). An event \( a \) is an elementary cause of the event \( b \) if \( a \) and \( b \) are not coincident and one of the following holds:

- \( a \) and \( b \) are proper events of the same process, and event \( a \) immediately precedes event \( b \) in the computational history of this process; or
- \( a \) and \( b \) are the events of sending and receiving the same message, respectively.

The causal precedence relation in the set \( E \) (denoted as \( a \rightarrow b \), a causally precedes \( b \)) is the transitive closure of the relation “is an elementary cause.” Two distinct events \( a \) and \( b \) are called concurrent \( (a \parallel b) \) if neither \((a \rightarrow b)\) nor \((b \rightarrow a)\).

With respect to the terminology used here, we would like to emphasize the difference between the notion of concurrency, on the one hand, and the notions of parallelism, simultaneity and independence, on the other hand.

The term parallelism is usually applied to a certain type of computer system architecture and computational structure. It does not exclude the communication between distinct processes. Therefore, two processes executing in parallel may contain a pair of events that are concurrent, as well as events that are characterized by an arbitrary direction of causal precedence between them.

Simultaneity of two events implies that both of them can be mapped to some external reference process or,